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. Each subsequent line in the
file corresponds to a rating from a customer and its date in the 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.
```

2.2 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 movi
e.
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 [ ]:

```
```

In [2]:

```
# this is just to know how much time will it take to run this entire ipython notebook

%matplotlib inline

import warnings
warnings.filterwarnings("ignore")

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

from sklearn import cross_validation, metrics   #Additional scklearn functions
from sklearn.grid_search import GridSearchCV   #Perforing grid search
from sklearn.model_selection import TimeSeriesSplit
```

3. Exploratory Data Analysis

3.1 Preprocessing

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

In [4]:
```
start = datetime.now()
if not os.path.isfile('data.csv'):
    # Create a file 'data.csv' before reading it
    # Read all the files in netflix and store them in one big file('data.csv')
    # We re reading from each of the four files and appendig each rating to a global file 'train.csv'
    data = open('data.csv', mode='w')
    
    row = list()
    files=['combined_data_1.txt','combined_data_2.txt',
           'combined_data_3.txt', 'combined_data_4.txt']
    for file in files:
        print("Reading ratings from {}...".format(file))
        with open(file) as f:
            for line in f:
                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
```

In [ ]:

In [5]:
```
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 [6]:
```
df.head()
```

Out[6]:

	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 [7]:
```
df.describe()['rating']
```

Out[7]:
```
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 [8]:
```
# 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 [9]:
```
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 [10]:
```
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 [11]:
```
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 d`isk 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 [12]:
```
# 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 [13]:
```
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
```

Observations

- In the total data there are about 100480507 ratings given by around 480189 users of the netflix and the total number of movies were 17770.
- By splitting the data into 80:20 the train data consists of 80384405 ratings which is considerably good amount of data for

exploration and training the models.

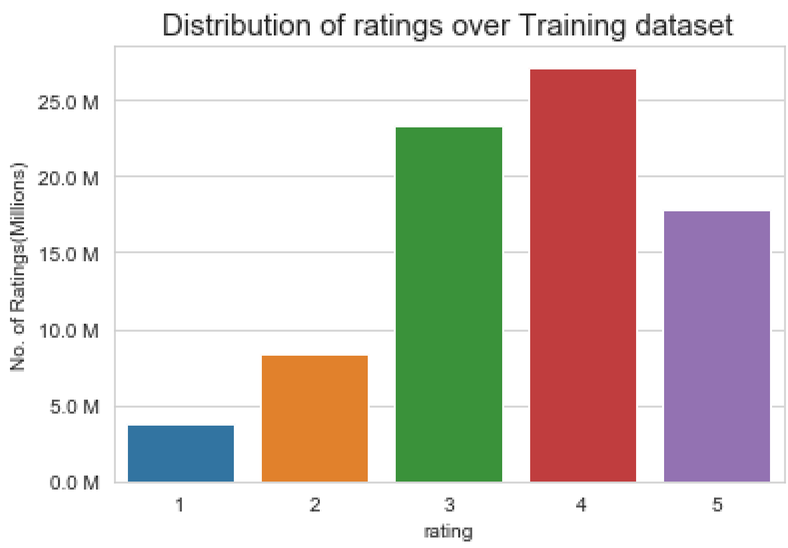
- So I took the training data for the further exploratory data analysis to properly understand the data.

3.3 Exploratory Data Analysis on Train data

In [14]:
```python
# 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 [15]:
```python
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()
```



- The above distribution of the ratings are quite interesting and gives important info about users behaviour
- The number of users given ratings (3 & 4) are fairly high as compared to the other ratings
- Very less number of users give (1) as rating to a movie
- So the users tends to give ratings on a higher side

New feature day_of_week is introduced

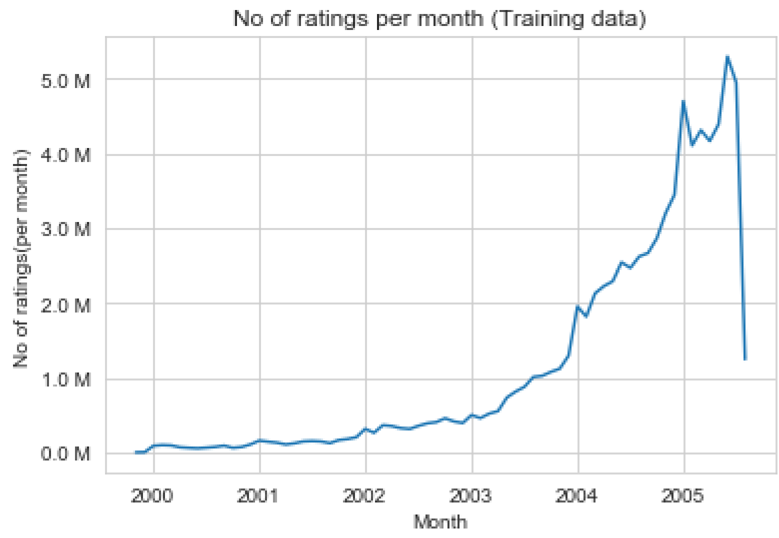
In [16]:
```python
# It is used to skip the warning ''SettingWithCopyWarning''..
pd.options.mode.chained_assignment = None  # default='warn'

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

train_df.tail()
```

Out[16]:

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

3.3.2 Number of Ratings per a month

In [17]:
```python
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()
```



- From the above time-series graph the ratings per month increased exponentially after 2003 which also indicates the growth of the Netflix

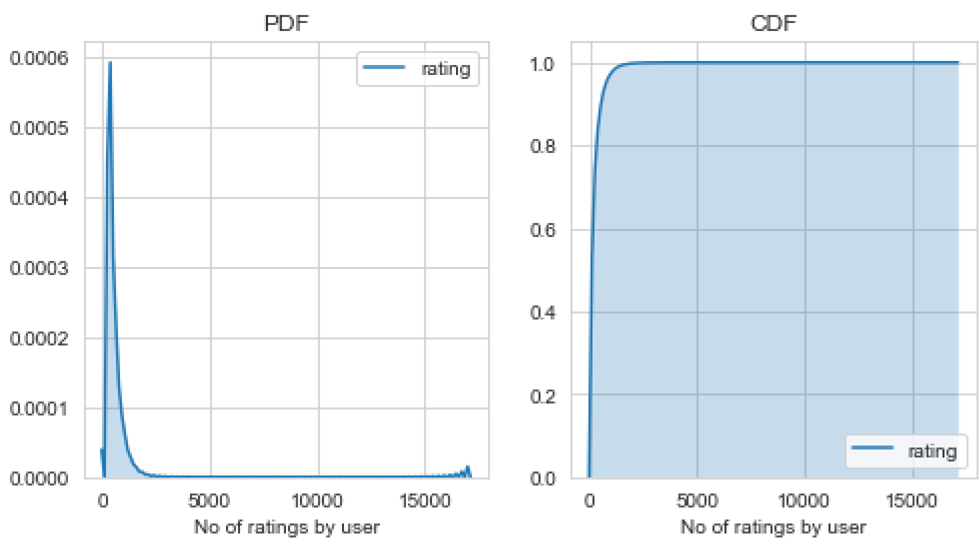
3.3.3 Analysis on the Ratings given by user

In [18]:
```
no_of_rated_movies_per_user = train_df.groupby(by='user')['rating'].count().sort_values(ascending=False)

no_of_rated_movies_per_user.head()
```

Out[18]:
```
user
305344     17112
2439493    15896
387418     15402
1639792     9767
1461435     9447
Name: rating, dtype: int64
```

- The above numbers indicate the total number of ratings given by an user.
- For a single user the number of ratings are higher and seems to be unrealistic.
- So let's explore further and understand the data properly

In [19]:
```
import warnings
warnings.filterwarnings("ignore")
import statsmodels.nonparametric.api as smnp
fig = plt.figure(figsize=plt.figaspect(.5))

ax1 = plt.subplot(121)

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

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

plt.show()
```

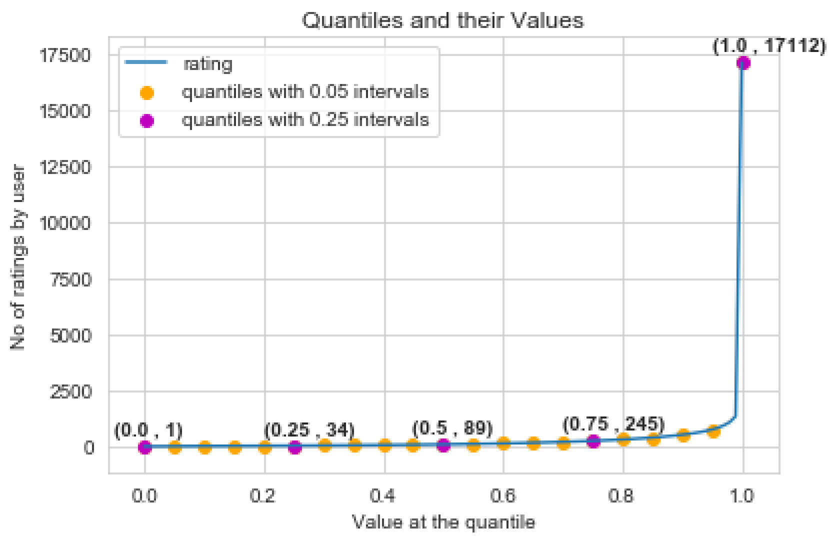


In [20]:
```
no_of_rated_movies_per_user.describe()
```

Out[20]:
```
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
```

In [21]:
```
quantiles = no_of_rated_movies_per_user.quantile(np.arange(0,1.01,0.01), interpolation='higher')
```

In [22]:
```
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 in
tervals")
# quantiles with 0.25 difference
plt.scatter(x=quantiles.index[::25], y=quantiles.values[::25], c='m', label = "quantiles with 0.25 int
ervals")
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 [23]:
```
quantiles[::5]
```

Out[23]:
```
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
```

In [24]:
```
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
```

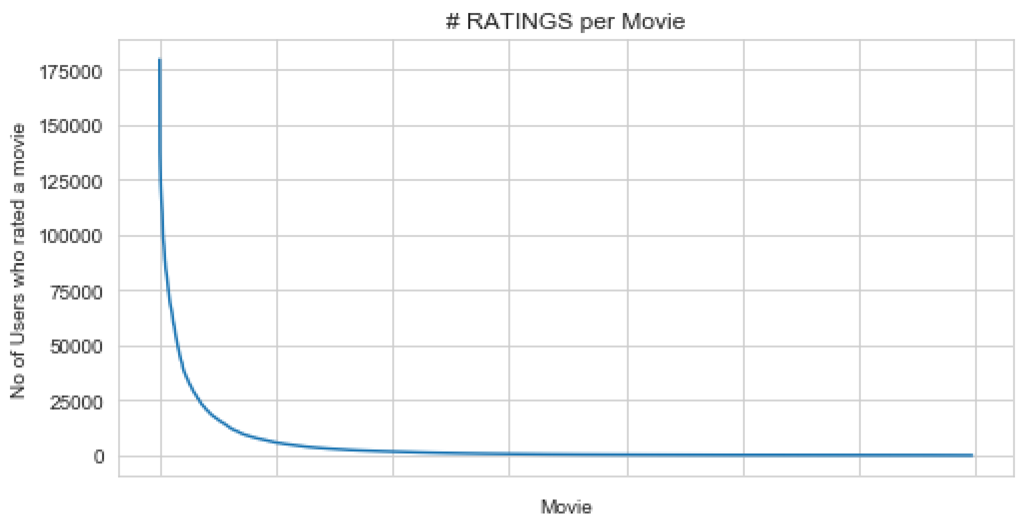
Observation

- By studying the above graphs and Quantiles interesting aspects of user ratings are understood and they are as follows:

- The mean value of rating is around 198 ratings.
- About 50% of users gave more than 89 nummbers of ratings.
- About 90% of users of users gave more than 15 numbers of ratings.

In [ ]:

In [25]:
```
no_of_ratings_per_movie = train_df.groupby(by='movie')['rating'].count().sort_values(ascending=False)

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

plt.show()
```

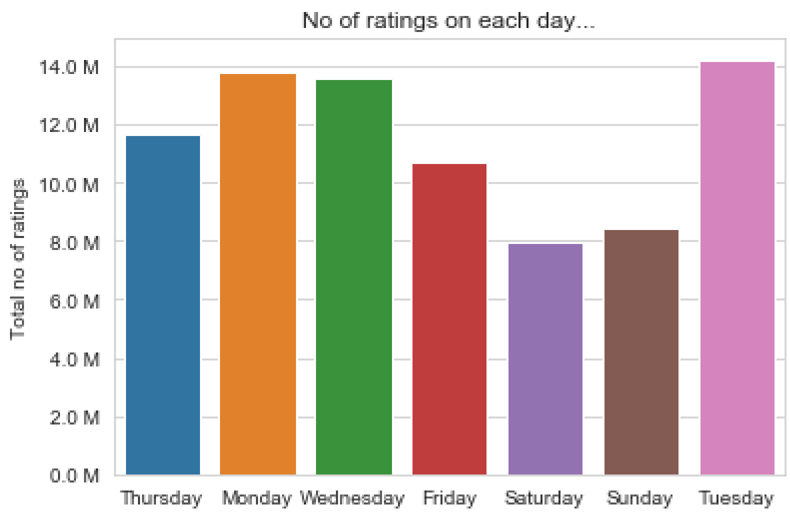


- **It is very skewed.. just like number of ratings given per user.**

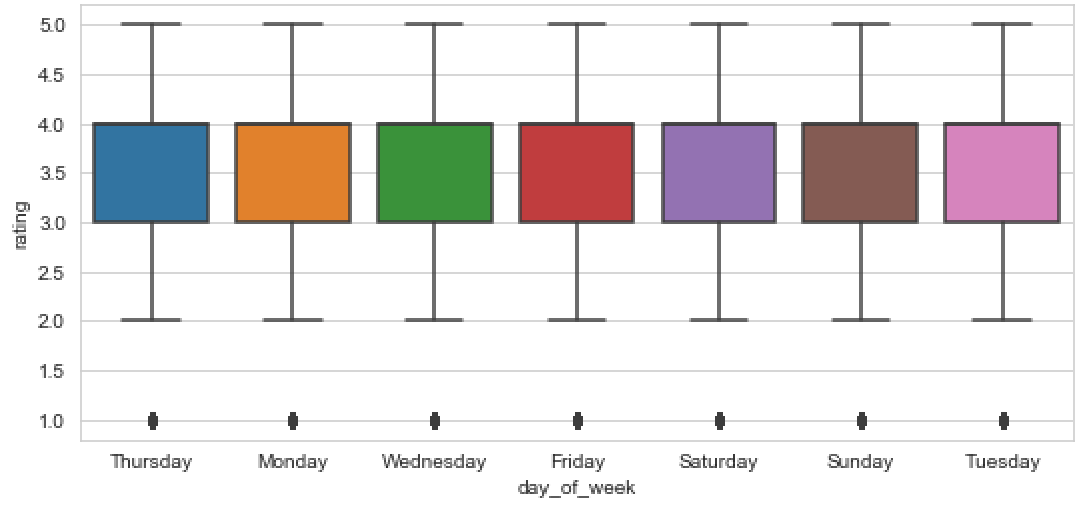
```
- There are some movies (which are very popular) which are rated by huge number of users.

- But most of the movies(like 90%) got some hundereds of ratings.
```

3.3.5 Number of ratings on each day of the week

In [26]:
```
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 [27]:
```
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)
```




```
0:01:01.749536
```

In [28]:

```
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
```

OBSERVATION

- The self-made feature Day_of_the_week has same distributions which can be observed by using the Box-plot figure
- The average ratings per day_of_the_week is around 3.57 to 3.59 which is very close to each other.
- Therefore this feature is not that important because it does not properly describre the train data.

3.3.6 Creating sparse matrix from data frame

In [29]:

```
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..
0:00:03.002459
```

In [30]:

```
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 [31]:
```
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..
0:00:00.933857
```

The Sparsity of Test data Matrix

In [32]:
```
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 %
```

In [33]:
```
# 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 [34]:
```
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[34]:
```
{'global': 3.582890686321557}
```

3.3.7.2 finding average rating per user

In [35]:
```
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
```

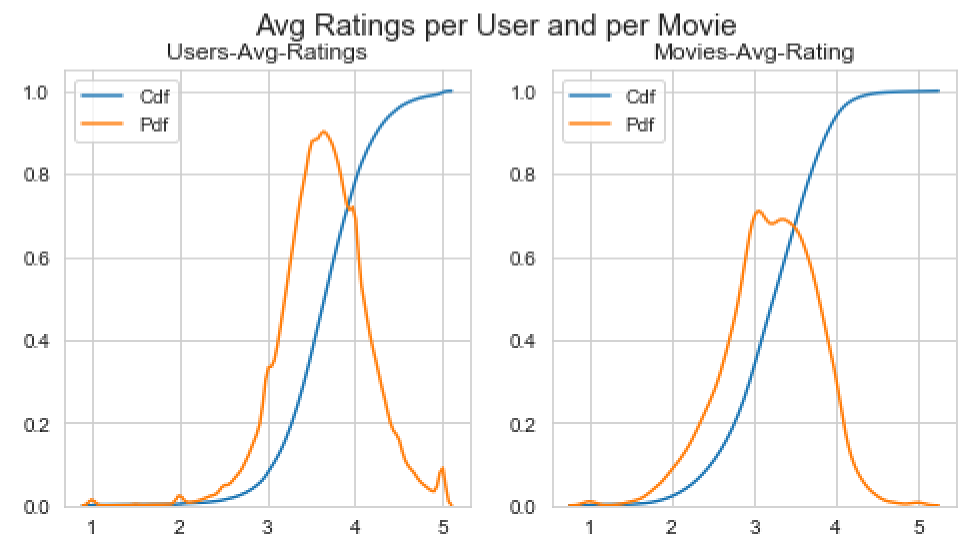
3.3.7.3 finding average rating per movie

In [36]:
```
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 [37]:
```
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)
```




```
0:00:45.674876
```

In [ ]:

In [ ]:

In [ ]:

3.4.2 Computing Movie-Movie Similarity matrix

In [38]:
```
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
0:00:37.781746
```

In [39]:
```
m_m_sim_sparse.shape
```

Out[39]:
```
(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 [40]:
```
movie_ids = np.unique(m_m_sim_sparse.nonzero()[1])
```

In [41]:
```
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]
```

```
0:00:31.072817
```

Out[41]:
```
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], dtype=int64)
```

3.4.3 Finding most similar movies using similarity matrix

In [42]:
```
# First Let's load the movie details into soe dataframe..
# movie details are in 'netflix/movie_titles.csv'

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

movie_titles.head()
```

```
Tokenization took: 2.99 ms
Type conversion took: 48.37 ms
Parser memory cleanup took: 0.00 ms
```

Out[42]:

	year_of_release	title
movie_id		
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

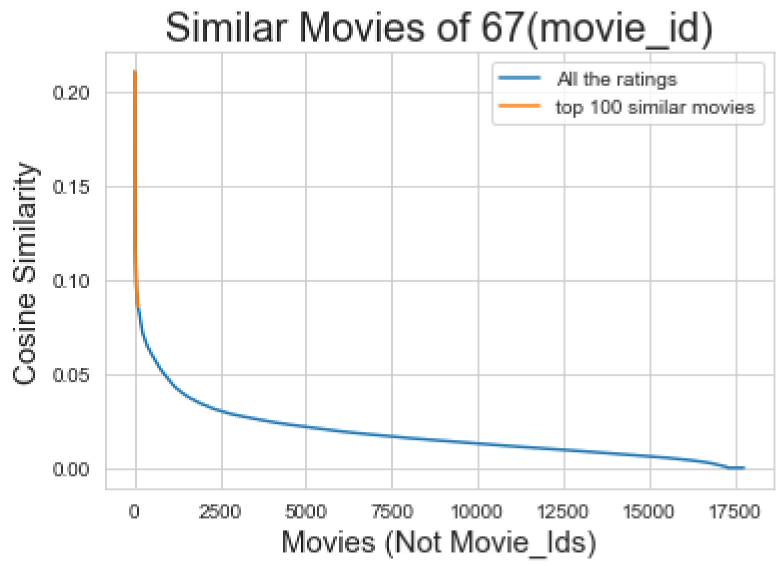
Similar Movies for 'Vampire Journals'

In [43]:
```
mv_id = 67

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

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

print("\nWe have {} movies which are similar to this  and we will get only top most..".format(m_m_sim_
sparse[:,mv_id].getnnz()))
```

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

It has 270 Ratings from users.

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

In [44]:
```
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 sim
ilarity (ie.,1)
                                               # and return its indices(movie_ids)
```

In [45]:
```
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()
```



In [46]:
```
movie_titles.loc[sim_indices[:10]]
```

Out[46]:

	year_of_release	title
movie_id		
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 [47]:
```
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.in1d(row_ind, sample_users),
                  np.in1d(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_movie
s)))
        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 test data from the test data

In [48]:
```
start = datetime.now()

path = "sample_test_sparse_matrix.npz"
if os.path.isfile(path):
    print("It is present in your pwd, getting it from disk....")
    # just get it from the disk instead of computing it
    sample_test_sparse_matrix = sparse.load_npz(path)
    print("DONE..")
else:
    # get 5k users and 500 movies from available data
    sample_test_sparse_matrix = get_sample_sparse_matrix(test_sparse_matrix, no_users=20000, no_movies
=2000,
                                                 path = "sample_test_sparse_matrix.npz")
print(datetime.now() - start)
```

```
It is present in your pwd, getting it from disk....
DONE..
0:00:00.336777
```

4.1.2 Build sample train data from the train data

In [49]:
```
start = datetime.now()
path = "sample_train_sparse_matrix.npz"
if os.path.isfile(path):
    print("It is present in your pwd, getting it from disk....")
    # just get it from the disk instead of computing it
    sample_train_sparse_matrix = sparse.load_npz(path)
    print("DONE..")
else:
    # get 10k users and 1k movies from available data
    sample_train_sparse_matrix = get_sample_sparse_matrix(train_sparse_matrix, no_users=25000, no_movi
es=3000,
                                             path = path)

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

```
It is present in your pwd, getting it from disk....
DONE..
0:00:00.285007
```

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

In [50]:
```
sample_train_averages = dict()
```

4.2.1 Finding Global Average of all movie ratings

In [51]:
```
# 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[51]:
```
{'global': 3.5875813607223455}
```

4.2.2 Finding Average rating per User

In [52]:
```
sample_train_averages['user'] = get_average_ratings(sample_train_sparse_matrix, of_users=True)
print('\nAverage rating of user 1515220 :',sample_train_averages['user'][1515220])
```

```
Average rating of user 1515220 : 3.923076923076923
```

4.2.3 Finding Average rating per Movie

In [53]:
```
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 : 2.752
```

4.3 Featurizing data

In [54]:
```
print('\n No of ratings in Our Sampled train matrix is : {}\n'.format(sample_train_sparse_matrix.count
_nonzero()))
print('\n No of ratings in Our Sampled test  matrix is : {}\n'.format(sample_test_sparse_matrix.count_
nonzero()))
```

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


 No of ratings in Our Sampled test  matrix is : 136507
```

4.3.1 Featurizing data for regression problem

4.3.1.1 Featurizing train data

In [55]:
```
# 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 [56]:
```
##############################################################
# It took me almost 3 days to prepare this train dataset.#
##############################################################
start = datetime.now()
if os.path.isfile('reg_train.csv'):
    print("File already exists you don't have to prepare again..." )
else:
    print('preparing {} tuples for the dataset..\n'.format(len(sample_train_ratings)))
    with open('reg_train.csv', mode='w') as reg_data_file:
        count = 0
        for (user, movie, rating)  in zip(sample_train_users, sample_train_movies, sample_train_rating
s):
            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_user
s_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_m
atrix.T).ravel()
            top_sim_movies = movie_sim.argsort()[::-1][1:] # we are ignoring 'The User' from its simil
ar 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
```

Reading from the file to make a Train_dataframe

In [57]:
```
reg_train = pd.read_csv('reg_train.csv', names = ['user', 'movie', 'GAvg', 'sur1', 'sur2', 'sur3', 'su
r4', 'sur5','smr1', 'smr2', 'smr3', 'smr4', 'smr5', 'UAvg', 'MAvg', 'rating'], header=None)
reg_train.head()
```

Out[57]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating
0	174683	10	3.587581	5.0	5.0	3.0	4.0	4.0	3.0	5.0	4.0	3.0	2.0	3.882353	3.611111	5
1	233949	10	3.587581	4.0	4.0	5.0	1.0	3.0	2.0	3.0	2.0	3.0	3.0	2.692308	3.611111	3
2	555770	10	3.587581	4.0	5.0	4.0	4.0	5.0	4.0	2.0	5.0	4.0	4.0	3.795455	3.611111	4
3	767518	10	3.587581	2.0	5.0	4.0	4.0	3.0	5.0	5.0	4.0	4.0	3.0	3.884615	3.611111	5
4	894393	10	3.587581	3.0	5.0	4.0	4.0	3.0	4.0	4.0	4.0	4.0	4.0	4.000000	3.611111	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.

4.3.1.2 Featurizing test data

In [58]:

```
# 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 [59]:

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

Out[59]:

```
3.5875813607223455
```

In [60]:

```
start = datetime.now()

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

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

        #--------------------- Ratings of "movie" by similar users of "user" ---------------------
            #print(user, movie)
            try:
                # compute the similar Users of the "user"
                user_sim = cosine_similarity(sample_train_sparse_matrix[user], sample_train_sparse_mat
rix).ravel()
                top_sim_users = user_sim.argsort()[::-1][1:] # we are ignoring 'The User' from its sim
ilar 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



            #--------------------- 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_spar
se_matrix.T).ravel()
                top_sim_movies = movie_sim.argsort()[::-1][1:] # we are ignoring 'The User' from its s
imilar 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_mov
ies_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)%10000 == 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 [61]:
```
reg_test_df = pd.read_csv('reg_test.csv', names = ['user', 'movie', 'GAvg', 'sur1', 'sur2', 'sur3', 's
ur4', 'sur5',
                                                          'smr1', 'smr2', 'smr3', 'smr4', 'smr5',
                                                          'UAvg', 'MAvg', 'rating'], header=None)
reg_test_df.head(4)
```

Out[61]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	sı
0	1129620	2	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587
1	3321	5	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587
2	508584	5	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587
3	731988	5	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587

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

4.3.2 Transforming data for Surprise models

In [ ]:

```python

```

In [62]:

```python
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 [63]:

```python
# 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 [64]:

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

Out[64]:

```
[(1129620, 2, 3), (3321, 5, 4), (508584, 5, 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 [65]:

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

models_evaluation_train, models_evaluation_test
```

Out[65]:

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

> **Utility functions for running regression models**

In [66]:
```
# 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

from sklearn.model_selection import GridSearchCV   #Perforing grid search

def Gridsearch_tuning(xgb_model,param,x_tr,y_tr):

    model = xgb_model

    param_grid=param

    kfold = TimeSeriesSplit(n_splits=5)
    grid_search = GridSearchCV(model, param_grid, scoring='neg_mean_squared_error', n_jobs=-1, cv=kfol
d)
    grid_result = grid_search.fit(x_tr,y_tr)
    # summarize results
    print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
    means = grid_result.cv_results_['mean_test_score']
    stds = grid_result.cv_results_['std_test_score']
    params = grid_result.cv_results_['params']
    for mean, stdev, param in zip(means, stds, params):
        print("%f (%f) with: %r" % (mean, stdev, param))
```

Utility functions for Surprise modes

In [67]:
```python
# 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 rating
s''.
    '''
    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['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

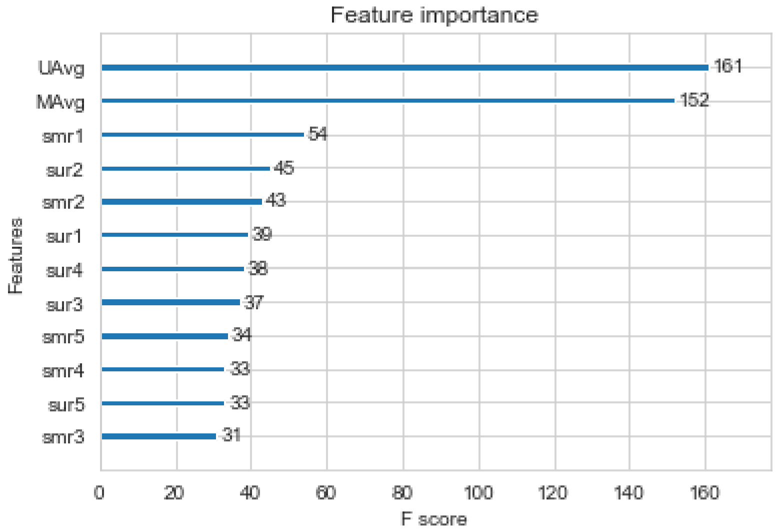
In [68]:
```
import xgboost as xgb
```

In [69]:
```
# 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..
[07:36:35] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:35] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:35] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:35] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:35] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:35] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:35] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:35] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:36] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:36] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:36] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:36] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:36] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:36] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:36] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:36] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:36] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:36] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:36] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:37] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:38] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:38] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:38] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:38] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:38] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:38] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:38] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:38] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:38] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:38] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:38] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:38] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
```

```
[07:36:39] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:39] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:39] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:39] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:39] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:39] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:39] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:39] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:39] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:39] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:39] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:39] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:40] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:40] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:40] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:40] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:40] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:40] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:40] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:40] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:40] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:40] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:40] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:40] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:40] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:41] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:41] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:41] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:41] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:41] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:41] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:41] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:41] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:41] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:41] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:41] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:41] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:42] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:42] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:42] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:42] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:42] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:42] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:42] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
```

```
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:42] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:42] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:42] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:42] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:42] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:43] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:43] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:43] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:43] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:43] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:43] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:43] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
[07:36:43] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.cc:74: tree pruning end, 1 ro
ots, 14 extra nodes, 0 pruned nodes, max_depth=3
Done. Time taken : 0:00:09.333535

Done

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

TEST DATA
------------------------------
RMSE :  1.0898255874531768
MAPE :  34.47749436884959
```



Gridsearch technique over the initial 13 features

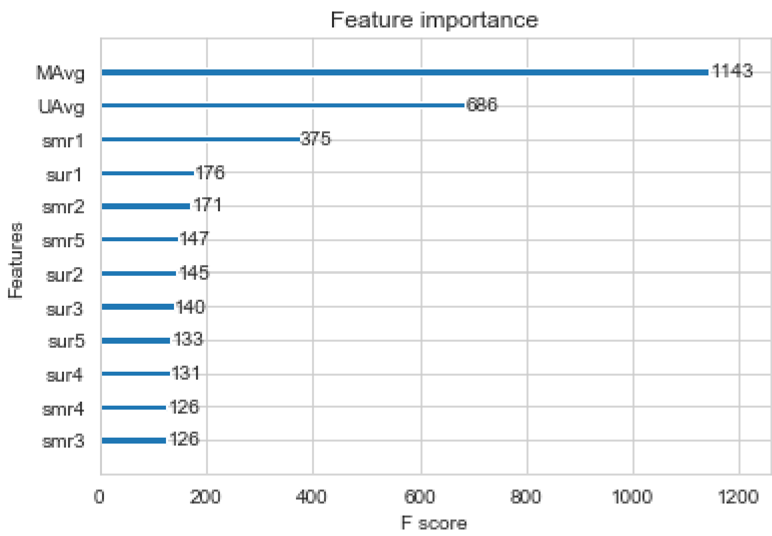
In [70]:
```
%%time
#Tuning the parameters to be given
n_estimators = [100,300,500,700,900,1100,1300] # Total number of base learners
learning_rate = [0.0001, 0.001, 0.01, 0.1] #Total gamma values
Max_depth=[1,2,3] #Depth of the trees

#Creating dictionary of parameters to be considered
param= dict(learning_rate=learning_rate, n_estimators=n_estimators,max_depth=Max_depth)

#Hyperarameter tuning the parameters using Gridsearch cross_validation technique
Gridsearch_tuning(param, x_train, y_train)
```

```
Best: -0.745368 using {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 500}
-10.504771 (0.148415) with: {'learning_rate': 0.0001, 'max_depth': 1, 'n_estimators': 100}
-10.133930 (0.145941) with: {'learning_rate': 0.0001, 'max_depth': 1, 'n_estimators': 300}
-9.777626 (0.143525) with: {'learning_rate': 0.0001, 'max_depth': 1, 'n_estimators': 500}
-9.435236 (0.141174) with: {'learning_rate': 0.0001, 'max_depth': 1, 'n_estimators': 700}
-9.106118 (0.138730) with: {'learning_rate': 0.0001, 'max_depth': 1, 'n_estimators': 900}
-8.789862 (0.136324) with: {'learning_rate': 0.0001, 'max_depth': 1, 'n_estimators': 1100}
-8.485432 (0.133132) with: {'learning_rate': 0.0001, 'max_depth': 1, 'n_estimators': 1300}
-10.501238 (0.145715) with: {'learning_rate': 0.0001, 'max_depth': 2, 'n_estimators': 100}
-10.123634 (0.138107) with: {'learning_rate': 0.0001, 'max_depth': 2, 'n_estimators': 300}
-9.760555 (0.131199) with: {'learning_rate': 0.0001, 'max_depth': 2, 'n_estimators': 500}
-9.411940 (0.124923) with: {'learning_rate': 0.0001, 'max_depth': 2, 'n_estimators': 700}
-9.077431 (0.119020) with: {'learning_rate': 0.0001, 'max_depth': 2, 'n_estimators': 900}
-8.757020 (0.115294) with: {'learning_rate': 0.0001, 'max_depth': 2, 'n_estimators': 1100}
-8.449207 (0.111191) with: {'learning_rate': 0.0001, 'max_depth': 2, 'n_estimators': 1300}
-10.500218 (0.146090) with: {'learning_rate': 0.0001, 'max_depth': 3, 'n_estimators': 100}
-10.120657 (0.139071) with: {'learning_rate': 0.0001, 'max_depth': 3, 'n_estimators': 300}
-9.756088 (0.132491) with: {'learning_rate': 0.0001, 'max_depth': 3, 'n_estimators': 500}
-9.405865 (0.126334) with: {'learning_rate': 0.0001, 'max_depth': 3, 'n_estimators': 700}
-9.069444 (0.120635) with: {'learning_rate': 0.0001, 'max_depth': 3, 'n_estimators': 900}
-8.746172 (0.115219) with: {'learning_rate': 0.0001, 'max_depth': 3, 'n_estimators': 1100}
-8.435833 (0.110192) with: {'learning_rate': 0.0001, 'max_depth': 3, 'n_estimators': 1300}
-8.945703 (0.137510) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 100}
-6.327478 (0.104875) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 300}
-4.565558 (0.079707) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 500}
-3.380229 (0.061259) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 700}
-2.582680 (0.048964) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 900}
-2.044285 (0.040267) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 1100}
-1.679693 (0.033609) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estimators': 1300}
-8.914855 (0.117058) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 100}
-6.273837 (0.083362) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 300}
-4.499942 (0.063916) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 500}
-3.308289 (0.050271) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 700}
-2.505816 (0.039182) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 900}
-1.963202 (0.030638) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 1100}
-1.595940 (0.024345) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estimators': 1300}
-8.905508 (0.117877) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 100}
-6.247740 (0.081909) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 300}
-4.466706 (0.061004) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 500}
-3.270048 (0.044598) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 700}
-2.464795 (0.033138) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 900}
-1.920709 (0.025080) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 1100}
-1.553256 (0.019707) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estimators': 1300}
-2.274944 (0.043579) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 100}
-0.900042 (0.022040) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 300}
-0.821135 (0.023411) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 500}
-0.791372 (0.023354) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 700}
-0.774759 (0.023179) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 900}
-0.764954 (0.023098) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 1100}
-0.758936 (0.023106) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 1300}
-2.195542 (0.034471) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 100}
-0.824037 (0.019728) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 300}
-0.767430 (0.023112) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 500}
-0.754642 (0.023585) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 700}
-0.749864 (0.023484) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 900}
-0.747864 (0.023287) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 1100}
-0.746903 (0.023098) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 1300}
-2.154470 (0.028357) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 100}
-0.795094 (0.020759) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 300}
-0.753017 (0.023484) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 500}
-0.747948 (0.023575) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 700}
-0.746570 (0.023457) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 900}
-0.745930 (0.023312) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 1100}
-0.745617 (0.023259) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 1300}
-0.767599 (0.023340) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 100}
-0.748313 (0.023040) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 300}
-0.747817 (0.022750) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 500}
-0.747758 (0.022691) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 700}
-0.747812 (0.022755) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 900}
-0.747840 (0.022799) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 1100}
-0.747879 (0.022835) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 1300}
-0.749680 (0.023277) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 100}
-0.746150 (0.022890) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 300}
-0.745956 (0.023104) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 500}
-0.745973 (0.023155) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 700}
-0.746090 (0.023357) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 900}
-0.746282 (0.023427) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 1100}
-0.746507 (0.023597) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 1300}
-0.747077 (0.023291) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100}
-0.745460 (0.023526) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 300}
-0.745368 (0.023600) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 500}
-0.745689 (0.023732) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 700}
-0.746000 (0.023941) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 900}
-0.746268 (0.024177) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 1100}
-0.746524 (0.024701) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 1300}
Wall time: 1h 55min 22s
```

Training the model with the tuned hyperparameters

In [70]:
```
# initialize Our first XGBoost model...
Tuned_xgb = xgb.XGBRegressor(max_depth=3,learning_rate=0.1, n_jobs=-1, random_state=15, n_estimators=5
00)
Tuned_train_results, Tuned_test_results = run_xgboost(Tuned_xgb, x_train, y_train, x_test, y_test)

# store the results in models_evaluations dictionaries
models_evaluation_train['Tuned_first_algo'] = Tuned_train_results
models_evaluation_test['Tuned_first_algo'] = Tuned_test_results

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

```
Training the model..
Done. Time taken : 0:00:38.794325

Done

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

TEST DATA
------------------------------
RMSE :  1.0904043649061108
MAPE :  34.459846201416916
```



4.4.2 Suprise BaselineModel

In [71]:
```
from surprise import BaselineOnly
```

In [72]:
```
# options are to specify.., how to compute those user and item biases
bsl_options = {'method': 'sgd',
               'learning_rate': .001
               }
bsl_algo = BaselineOnly(bsl_options=bsl_options)
# run this algorithm.., It will return the train and test results..
bsl_train_results, bsl_test_results = run_surprise(bsl_algo, trainset, testset, verbose=True)


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

```
Training the model...
Estimating biases using sgd...
Done. time taken : 0:00:03.813619

Evaluating the model with train data..
time taken : 0:00:04.763998
---------------
Train Data
---------------
RMSE : 0.9220478981418425

MAPE : 28.6415868708249

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.867936
---------------
Test Data
---------------
RMSE : 1.084696782600206

MAPE : 34.484040979947565

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:00:09.445553
```

4.4.3 XGBoost with initial 13 features + Surprise Baseline predictor

Updating Train Data

In [73]:
```
# add our baseline_predicted value as our feature..
reg_train['bslpr'] = models_evaluation_train['bsl_algo']['predictions']
reg_train.head(2)
```

Out[73]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating	bsl
0	174683	10	3.587581	5.0	5.0	3.0	4.0	4.0	3.0	5.0	4.0	3.0	2.0	3.882353	3.611111	5	3.68139
1	233949	10	3.587581	4.0	4.0	5.0	1.0	3.0	2.0	3.0	2.0	3.0	3.0	2.692308	3.611111	3	3.72015

Updating Test Data

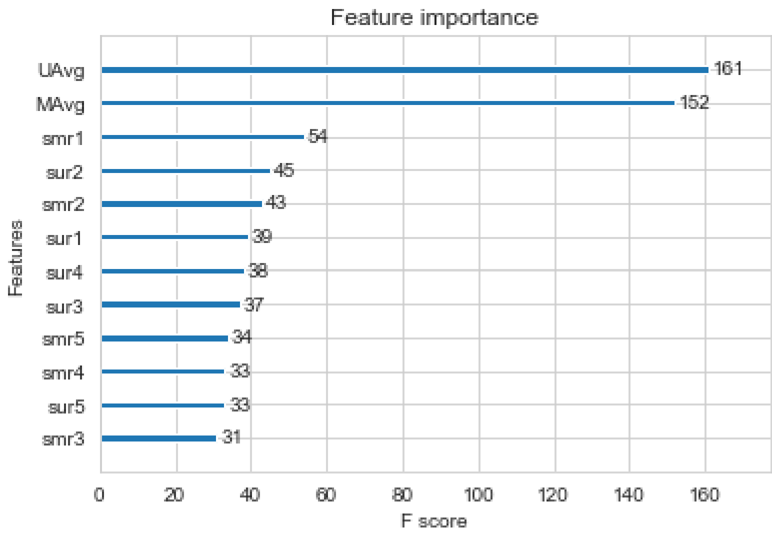
In [74]:
```
# add that baseline predicted ratings with Surprise to the test data as well
reg_test_df['bslpr']  = models_evaluation_test['bsl_algo']['predictions']

reg_test_df.head(2)
```

Out[74]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	s
0	1129620	2	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587
1	3321	5	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587

In [75]:
```
# 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( 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..
Done. Time taken : 0:00:10.109774

Done

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

TEST DATA
------------------------------
RMSE :  1.0898255874531768
MAPE :  34.47749436884959
```



Gridsearch over XGBoost with initial 13 features + Surprise Baseline predictor

In [79]:
```
%%time
#Tuning the parameters to be given
n_estimators = [100,300,500,700,900,1100,1300] # Total number of base learners
learning_rate = [0.0001, 0.001, 0.01, 0.1] #Total gamma values
gamma=[i/10.0 for i in range(0,5)]

#Creating dictionary of parameters to be considered
Param= dict(learning_rate=learning_rate, n_estimators=n_estimators,gamma=gamma)

#Hyperarameter tuning the parameters using Gridsearch cross_validation technique
Gridsearch_tuning(Param, x_train, y_train)
```

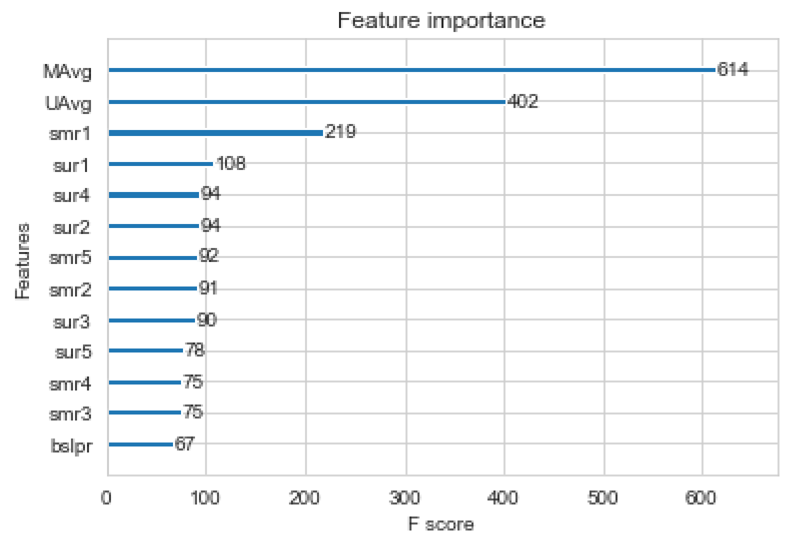
```
Best: -0.745504 using {'gamma': 0.1, 'learning_rate': 0.1, 'n_estimators': 300}
-10.500218 (0.146090) with: {'gamma': 0.0, 'learning_rate': 0.0001, 'n_estimators': 100}
-10.120657 (0.139071) with: {'gamma': 0.0, 'learning_rate': 0.0001, 'n_estimators': 300}
-9.756088 (0.132491) with: {'gamma': 0.0, 'learning_rate': 0.0001, 'n_estimators': 500}
-9.405865 (0.126334) with: {'gamma': 0.0, 'learning_rate': 0.0001, 'n_estimators': 700}
-9.069444 (0.120635) with: {'gamma': 0.0, 'learning_rate': 0.0001, 'n_estimators': 900}
-8.746172 (0.115219) with: {'gamma': 0.0, 'learning_rate': 0.0001, 'n_estimators': 1100}
-8.435833 (0.110192) with: {'gamma': 0.0, 'learning_rate': 0.0001, 'n_estimators': 1300}
-8.905508 (0.117877) with: {'gamma': 0.0, 'learning_rate': 0.001, 'n_estimators': 100}
-6.247740 (0.081909) with: {'gamma': 0.0, 'learning_rate': 0.001, 'n_estimators': 300}
-4.466706 (0.061004) with: {'gamma': 0.0, 'learning_rate': 0.001, 'n_estimators': 500}
-3.270048 (0.044598) with: {'gamma': 0.0, 'learning_rate': 0.001, 'n_estimators': 700}
-2.464795 (0.033138) with: {'gamma': 0.0, 'learning_rate': 0.001, 'n_estimators': 900}
-1.920709 (0.025080) with: {'gamma': 0.0, 'learning_rate': 0.001, 'n_estimators': 1100}
-1.553256 (0.019707) with: {'gamma': 0.0, 'learning_rate': 0.001, 'n_estimators': 1300}
-2.154470 (0.028357) with: {'gamma': 0.0, 'learning_rate': 0.01, 'n_estimators': 100}
-0.795094 (0.020759) with: {'gamma': 0.0, 'learning_rate': 0.01, 'n_estimators': 300}
-0.753017 (0.023484) with: {'gamma': 0.0, 'learning_rate': 0.01, 'n_estimators': 500}
-0.747936 (0.023553) with: {'gamma': 0.0, 'learning_rate': 0.01, 'n_estimators': 700}
-0.746529 (0.023375) with: {'gamma': 0.0, 'learning_rate': 0.01, 'n_estimators': 900}
-0.745934 (0.023304) with: {'gamma': 0.0, 'learning_rate': 0.01, 'n_estimators': 1100}
-0.745633 (0.023259) with: {'gamma': 0.0, 'learning_rate': 0.01, 'n_estimators': 1300}
-0.747064 (0.023279) with: {'gamma': 0.0, 'learning_rate': 0.1, 'n_estimators': 100}
-0.745515 (0.023285) with: {'gamma': 0.0, 'learning_rate': 0.1, 'n_estimators': 300}
-0.745754 (0.023528) with: {'gamma': 0.0, 'learning_rate': 0.1, 'n_estimators': 500}
-0.746022 (0.023597) with: {'gamma': 0.0, 'learning_rate': 0.1, 'n_estimators': 700}
-0.746256 (0.023857) with: {'gamma': 0.0, 'learning_rate': 0.1, 'n_estimators': 900}
-0.746519 (0.024135) with: {'gamma': 0.0, 'learning_rate': 0.1, 'n_estimators': 1100}
-0.746947 (0.024239) with: {'gamma': 0.0, 'learning_rate': 0.1, 'n_estimators': 1300}
-10.500218 (0.146090) with: {'gamma': 0.1, 'learning_rate': 0.0001, 'n_estimators': 100}
-10.120657 (0.139071) with: {'gamma': 0.1, 'learning_rate': 0.0001, 'n_estimators': 300}
-9.756088 (0.132491) with: {'gamma': 0.1, 'learning_rate': 0.0001, 'n_estimators': 500}
-9.405865 (0.126334) with: {'gamma': 0.1, 'learning_rate': 0.0001, 'n_estimators': 700}
-9.069444 (0.120635) with: {'gamma': 0.1, 'learning_rate': 0.0001, 'n_estimators': 900}
-8.746172 (0.115219) with: {'gamma': 0.1, 'learning_rate': 0.0001, 'n_estimators': 1100}
-8.435833 (0.110192) with: {'gamma': 0.1, 'learning_rate': 0.0001, 'n_estimators': 1300}
-8.905508 (0.117877) with: {'gamma': 0.1, 'learning_rate': 0.001, 'n_estimators': 100}
-6.247740 (0.081909) with: {'gamma': 0.1, 'learning_rate': 0.001, 'n_estimators': 300}
-4.466706 (0.061004) with: {'gamma': 0.1, 'learning_rate': 0.001, 'n_estimators': 500}
-3.270048 (0.044598) with: {'gamma': 0.1, 'learning_rate': 0.001, 'n_estimators': 700}
-2.464795 (0.033138) with: {'gamma': 0.1, 'learning_rate': 0.001, 'n_estimators': 900}
-1.920709 (0.025080) with: {'gamma': 0.1, 'learning_rate': 0.001, 'n_estimators': 1100}
-1.553256 (0.019707) with: {'gamma': 0.1, 'learning_rate': 0.001, 'n_estimators': 1300}
-2.154470 (0.028357) with: {'gamma': 0.1, 'learning_rate': 0.01, 'n_estimators': 100}
-0.795094 (0.020759) with: {'gamma': 0.1, 'learning_rate': 0.01, 'n_estimators': 300}
-0.753017 (0.023484) with: {'gamma': 0.1, 'learning_rate': 0.01, 'n_estimators': 500}
-0.747936 (0.023553) with: {'gamma': 0.1, 'learning_rate': 0.01, 'n_estimators': 700}
-0.746529 (0.023375) with: {'gamma': 0.1, 'learning_rate': 0.01, 'n_estimators': 900}
-0.745933 (0.023304) with: {'gamma': 0.1, 'learning_rate': 0.01, 'n_estimators': 1100}
-0.745635 (0.023259) with: {'gamma': 0.1, 'learning_rate': 0.01, 'n_estimators': 1300}
-0.747064 (0.023279) with: {'gamma': 0.1, 'learning_rate': 0.1, 'n_estimators': 100}
-0.745504 (0.023265) with: {'gamma': 0.1, 'learning_rate': 0.1, 'n_estimators': 300}
-0.745747 (0.023379) with: {'gamma': 0.1, 'learning_rate': 0.1, 'n_estimators': 500}
-0.746015 (0.023440) with: {'gamma': 0.1, 'learning_rate': 0.1, 'n_estimators': 700}
-0.746272 (0.023619) with: {'gamma': 0.1, 'learning_rate': 0.1, 'n_estimators': 900}
-0.746614 (0.023745) with: {'gamma': 0.1, 'learning_rate': 0.1, 'n_estimators': 1100}
-0.746836 (0.023919) with: {'gamma': 0.1, 'learning_rate': 0.1, 'n_estimators': 1300}
-10.500218 (0.146090) with: {'gamma': 0.2, 'learning_rate': 0.0001, 'n_estimators': 100}
-10.120657 (0.139071) with: {'gamma': 0.2, 'learning_rate': 0.0001, 'n_estimators': 300}
-9.756088 (0.132491) with: {'gamma': 0.2, 'learning_rate': 0.0001, 'n_estimators': 500}
-9.405865 (0.126334) with: {'gamma': 0.2, 'learning_rate': 0.0001, 'n_estimators': 700}
-9.069444 (0.120635) with: {'gamma': 0.2, 'learning_rate': 0.0001, 'n_estimators': 900}
-8.746172 (0.115219) with: {'gamma': 0.2, 'learning_rate': 0.0001, 'n_estimators': 1100}
-8.435833 (0.110192) with: {'gamma': 0.2, 'learning_rate': 0.0001, 'n_estimators': 1300}
-8.905508 (0.117877) with: {'gamma': 0.2, 'learning_rate': 0.001, 'n_estimators': 100}
-6.247740 (0.081909) with: {'gamma': 0.2, 'learning_rate': 0.001, 'n_estimators': 300}
-4.466706 (0.061004) with: {'gamma': 0.2, 'learning_rate': 0.001, 'n_estimators': 500}
-3.270048 (0.044598) with: {'gamma': 0.2, 'learning_rate': 0.001, 'n_estimators': 700}
-2.464795 (0.033138) with: {'gamma': 0.2, 'learning_rate': 0.001, 'n_estimators': 900}
-1.920709 (0.025080) with: {'gamma': 0.2, 'learning_rate': 0.001, 'n_estimators': 1100}
-1.553256 (0.019707) with: {'gamma': 0.2, 'learning_rate': 0.001, 'n_estimators': 1300}
-2.154470 (0.028357) with: {'gamma': 0.2, 'learning_rate': 0.01, 'n_estimators': 100}
-0.795094 (0.020759) with: {'gamma': 0.2, 'learning_rate': 0.01, 'n_estimators': 300}
-0.753017 (0.023484) with: {'gamma': 0.2, 'learning_rate': 0.01, 'n_estimators': 500}
-0.747936 (0.023553) with: {'gamma': 0.2, 'learning_rate': 0.01, 'n_estimators': 700}
-0.746529 (0.023375) with: {'gamma': 0.2, 'learning_rate': 0.01, 'n_estimators': 900}
-0.745933 (0.023304) with: {'gamma': 0.2, 'learning_rate': 0.01, 'n_estimators': 1100}
-0.745636 (0.023259) with: {'gamma': 0.2, 'learning_rate': 0.01, 'n_estimators': 1300}
-0.747064 (0.023279) with: {'gamma': 0.2, 'learning_rate': 0.1, 'n_estimators': 100}
-0.745544 (0.023267) with: {'gamma': 0.2, 'learning_rate': 0.1, 'n_estimators': 300}
-0.745716 (0.023357) with: {'gamma': 0.2, 'learning_rate': 0.1, 'n_estimators': 500}
-0.745889 (0.023395) with: {'gamma': 0.2, 'learning_rate': 0.1, 'n_estimators': 700}
-0.746196 (0.023593) with: {'gamma': 0.2, 'learning_rate': 0.1, 'n_estimators': 900}
-0.746458 (0.023807) with: {'gamma': 0.2, 'learning_rate': 0.1, 'n_estimators': 1100}
-0.746599 (0.024094) with: {'gamma': 0.2, 'learning_rate': 0.1, 'n_estimators': 1300}
-10.500218 (0.146090) with: {'gamma': 0.3, 'learning_rate': 0.0001, 'n_estimators': 100}
-10.120657 (0.139071) with: {'gamma': 0.3, 'learning_rate': 0.0001, 'n_estimators': 300}
```

```
-9.756088 (0.132491) with: {'gamma': 0.3, 'learning_rate': 0.0001, 'n_estimators': 500}
-9.405865 (0.126334) with: {'gamma': 0.3, 'learning_rate': 0.0001, 'n_estimators': 700}
-9.069444 (0.120635) with: {'gamma': 0.3, 'learning_rate': 0.0001, 'n_estimators': 900}
-8.746172 (0.115219) with: {'gamma': 0.3, 'learning_rate': 0.0001, 'n_estimators': 1100}
-8.435833 (0.110192) with: {'gamma': 0.3, 'learning_rate': 0.0001, 'n_estimators': 1300}
-8.905508 (0.117877) with: {'gamma': 0.3, 'learning_rate': 0.001, 'n_estimators': 100}
-6.247740 (0.081909) with: {'gamma': 0.3, 'learning_rate': 0.001, 'n_estimators': 300}
-4.466706 (0.061004) with: {'gamma': 0.3, 'learning_rate': 0.001, 'n_estimators': 500}
-3.270048 (0.044598) with: {'gamma': 0.3, 'learning_rate': 0.001, 'n_estimators': 700}
-2.464795 (0.033138) with: {'gamma': 0.3, 'learning_rate': 0.001, 'n_estimators': 900}
-1.920709 (0.025080) with: {'gamma': 0.3, 'learning_rate': 0.001, 'n_estimators': 1100}
-1.553256 (0.019707) with: {'gamma': 0.3, 'learning_rate': 0.001, 'n_estimators': 1300}
-2.154470 (0.028357) with: {'gamma': 0.3, 'learning_rate': 0.01, 'n_estimators': 100}
-0.795094 (0.020759) with: {'gamma': 0.3, 'learning_rate': 0.01, 'n_estimators': 300}
-0.753017 (0.023484) with: {'gamma': 0.3, 'learning_rate': 0.01, 'n_estimators': 500}
-0.747936 (0.023553) with: {'gamma': 0.3, 'learning_rate': 0.01, 'n_estimators': 700}
-0.746529 (0.023375) with: {'gamma': 0.3, 'learning_rate': 0.01, 'n_estimators': 900}
-0.745931 (0.023299) with: {'gamma': 0.3, 'learning_rate': 0.01, 'n_estimators': 1100}
-0.745631 (0.023255) with: {'gamma': 0.3, 'learning_rate': 0.01, 'n_estimators': 1300}
-0.747064 (0.023279) with: {'gamma': 0.3, 'learning_rate': 0.1, 'n_estimators': 100}
-0.745548 (0.023274) with: {'gamma': 0.3, 'learning_rate': 0.1, 'n_estimators': 300}
-0.745672 (0.023329) with: {'gamma': 0.3, 'learning_rate': 0.1, 'n_estimators': 500}
-0.745849 (0.023537) with: {'gamma': 0.3, 'learning_rate': 0.1, 'n_estimators': 700}
-0.746211 (0.023697) with: {'gamma': 0.3, 'learning_rate': 0.1, 'n_estimators': 900}
-0.746636 (0.023763) with: {'gamma': 0.3, 'learning_rate': 0.1, 'n_estimators': 1100}
-0.746798 (0.024012) with: {'gamma': 0.3, 'learning_rate': 0.1, 'n_estimators': 1300}
-10.500218 (0.146090) with: {'gamma': 0.4, 'learning_rate': 0.0001, 'n_estimators': 100}
-10.120657 (0.139071) with: {'gamma': 0.4, 'learning_rate': 0.0001, 'n_estimators': 300}
-9.756088 (0.132491) with: {'gamma': 0.4, 'learning_rate': 0.0001, 'n_estimators': 500}
-9.405865 (0.126334) with: {'gamma': 0.4, 'learning_rate': 0.0001, 'n_estimators': 700}
-9.069444 (0.120635) with: {'gamma': 0.4, 'learning_rate': 0.0001, 'n_estimators': 900}
-8.746172 (0.115219) with: {'gamma': 0.4, 'learning_rate': 0.0001, 'n_estimators': 1100}
-8.435833 (0.110192) with: {'gamma': 0.4, 'learning_rate': 0.0001, 'n_estimators': 1300}
-8.905508 (0.117877) with: {'gamma': 0.4, 'learning_rate': 0.001, 'n_estimators': 100}
-6.247740 (0.081909) with: {'gamma': 0.4, 'learning_rate': 0.001, 'n_estimators': 300}
-4.466706 (0.061004) with: {'gamma': 0.4, 'learning_rate': 0.001, 'n_estimators': 500}
-3.270048 (0.044598) with: {'gamma': 0.4, 'learning_rate': 0.001, 'n_estimators': 700}
-2.464795 (0.033138) with: {'gamma': 0.4, 'learning_rate': 0.001, 'n_estimators': 900}
-1.920709 (0.025080) with: {'gamma': 0.4, 'learning_rate': 0.001, 'n_estimators': 1100}
-1.553256 (0.019707) with: {'gamma': 0.4, 'learning_rate': 0.001, 'n_estimators': 1300}
-2.154470 (0.028357) with: {'gamma': 0.4, 'learning_rate': 0.01, 'n_estimators': 100}
-0.795094 (0.020759) with: {'gamma': 0.4, 'learning_rate': 0.01, 'n_estimators': 300}
-0.753017 (0.023484) with: {'gamma': 0.4, 'learning_rate': 0.01, 'n_estimators': 500}
-0.747932 (0.023546) with: {'gamma': 0.4, 'learning_rate': 0.01, 'n_estimators': 700}
-0.746533 (0.023383) with: {'gamma': 0.4, 'learning_rate': 0.01, 'n_estimators': 900}
-0.745935 (0.023308) with: {'gamma': 0.4, 'learning_rate': 0.01, 'n_estimators': 1100}
-0.745633 (0.023259) with: {'gamma': 0.4, 'learning_rate': 0.01, 'n_estimators': 1300}
-0.747063 (0.023278) with: {'gamma': 0.4, 'learning_rate': 0.1, 'n_estimators': 100}
-0.745616 (0.023354) with: {'gamma': 0.4, 'learning_rate': 0.1, 'n_estimators': 300}
-0.745806 (0.023408) with: {'gamma': 0.4, 'learning_rate': 0.1, 'n_estimators': 500}
-0.745964 (0.023454) with: {'gamma': 0.4, 'learning_rate': 0.1, 'n_estimators': 700}
-0.746108 (0.023349) with: {'gamma': 0.4, 'learning_rate': 0.1, 'n_estimators': 900}
-0.746291 (0.023276) with: {'gamma': 0.4, 'learning_rate': 0.1, 'n_estimators': 1100}
-0.746433 (0.023207) with: {'gamma': 0.4, 'learning_rate': 0.1, 'n_estimators': 1300}
Wall time: 5h 15min 59s
```

Training the model with the tuned hyperparameters

In [76]:
```
# initialize Our first XGBoost model...
Tuned_xgb_bsl = xgb.XGBRegressor( n_jobs=-1, random_state=15, learning_rate=0.1,gamma=0.1,n_estimators
=300)
bsl_tr_results, bsl_test_results = run_xgboost(Tuned_xgb_bsl, x_train, y_train, x_test, y_test)

# store the results in models_evaluations dictionaries
models_evaluation_train['Tuned_xgb_bsl'] = bsl_tr_results
models_evaluation_test['Tuned_xgb_bsl'] =  bsl_test_results

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

```
Training the model..
Done. Time taken : 0:00:30.373879

Done

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

TEST DATA
------------------------------
RMSE :  1.0922257160977793
MAPE :  34.369913875810845
```



4.4.4 Surprise KNNBaseline predictor

In [77]:
```
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***)

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

- $\pmb{b_{ui}}$ - *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):

$$\hat{r}_{ui} = b_{ui} + \frac{\sum\limits_{j \in N^k_u(i)} \text{sim}(i, j) \cdot (r_{uj} - b_{uj})}{\sum\limits_{j \in N^k_u(j)} \text{sim}(i, j)}$$

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

4.4.4.1 Surprise KNNBaseline with user user similarities

In [78]:
```
# 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=T
rue)

# 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 : 0:36:03.173417

Evaluating the model with train data..
time taken : 0:16:08.733142
---------------
Train Data
---------------
RMSE : 0.4536279292470732

MAPE : 12.840252350475915

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:01.877863
---------------
Test Data
---------------
RMSE : 1.0850618463554647

MAPE : 34.48062216705011

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:52:13.862529
```

4.4.4.2 Surprise KNNBaseline with movie movie similarities

In [79]:

```
# 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=T
rue)

# 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 : 0:00:14.700325

Evaluating the model with train data..
time taken : 0:01:23.193689
---------------
Train Data
---------------
RMSE : 0.5038994796517224

MAPE : 14.168515366483724

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.965288
---------------
Test Data
---------------
RMSE : 1.0852678745012594

MAPE : 34.48337123552355

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:01:38.859302
```

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 [80]:
```
# 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[80]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating	bsl
0	174683	10	3.587581	5.0	5.0	3.0	4.0	4.0	3.0	5.0	4.0	3.0	2.0	3.882353	3.611111	5	3.6813!
1	233949	10	3.587581	4.0	4.0	5.0	1.0	3.0	2.0	3.0	2.0	3.0	3.0	2.692308	3.611111	3	3.7201!

Preparing Test data

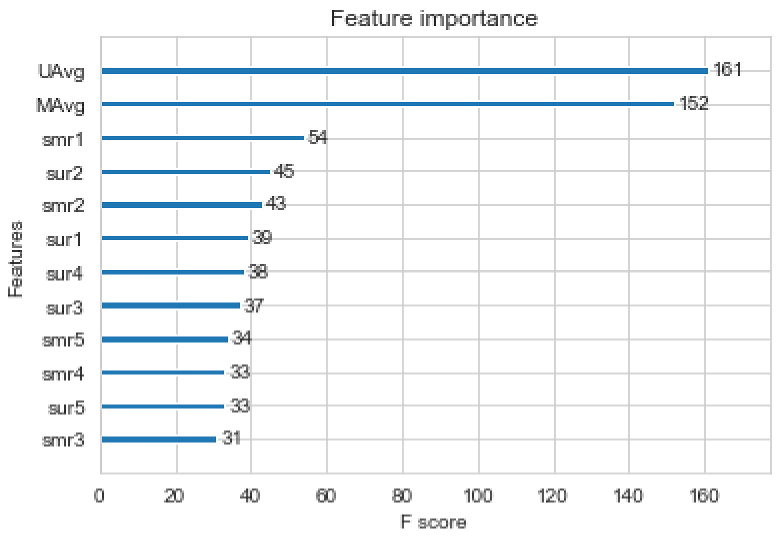
In [81]:
```
reg_test_df['knn_bsl_u'] = models_evaluation_test['knn_bsl_u']['predictions']
reg_test_df['knn_bsl_m'] = models_evaluation_test['knn_bsl_m']['predictions']

reg_test_df.head(2)
```

Out[81]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	sı
0	1129620	2	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587
1	3321	5	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587

In [82]:
```
# 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..
Done. Time taken : 0:00:12.569480

Done

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

TEST DATA
------------------------------
RMSE :  1.0898255874531768
MAPE :  34.47749436884959
```



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

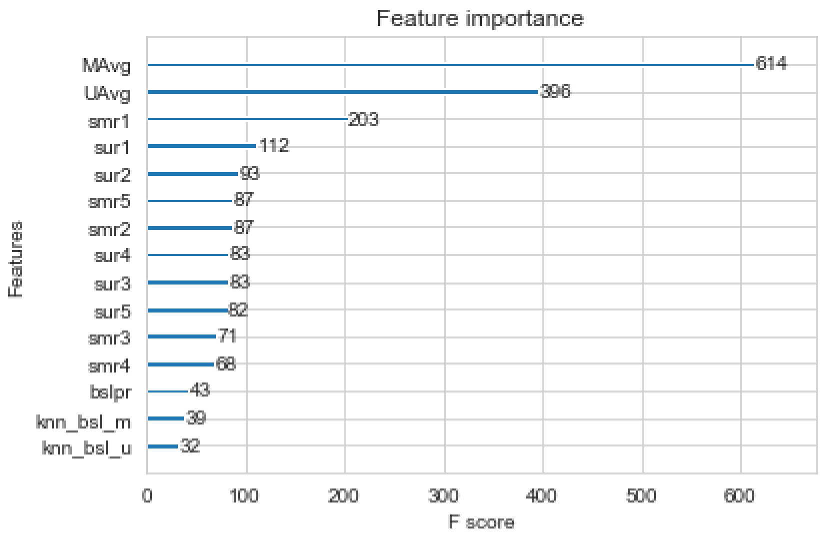
In [83]:
```
%%time
#Tuning the parameters to be given

reg_alpha=[1e-5, 1e-2, 0.1, 1, 100]

model=xgb.XGBRegressor(n_jobs=-1, random_state=15,learning_rate=0.1,gamma=0.1,n_estimators=300)
#Creating dictionary of parameters to be considered
Param_3= dict(reg_alpha=reg_alpha)

#Hyperarameter tuning the parameters using Gridsearch cross_validation technique
Gridsearch_tuning(model,Param_3, x_train, y_train)
```

```
Best: -0.745530 using {'reg_alpha': 1}
-0.745733 (0.023344) with: {'reg_alpha': 1e-05}
-0.745704 (0.023409) with: {'reg_alpha': 0.01}
-0.745611 (0.023613) with: {'reg_alpha': 0.1}
-0.745530 (0.023473) with: {'reg_alpha': 1}
-0.746315 (0.023713) with: {'reg_alpha': 100}
Wall time: 6min 45s
```

Training the model with the tuned hyperparameters

In [84]:
```
# declare the model
Tuned_xgb_knn_bsl = xgb.XGBRegressor(n_jobs=-1, random_state=15,learning_rate=0.1,gamma=0.1,n_estimato
rs=300,reg_alpha=1)
knn_bsl_train_results, knn_bsl_test_results = run_xgboost(Tuned_xgb_knn_bsl, x_train, y_train, x_test,
 y_test)

# store the results in models_evaluations dictionaries
models_evaluation_train['Tuned_xgb_knn_bsl'] = knn_bsl_train_results
models_evaluation_test['Tuned_xgb_knn_bsl'] = knn_bsl_test_results


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

```
Training the model..
Done. Time taken : 0:00:36.421356

Done

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

TEST DATA
------------------------------
RMSE :  1.090692423523049
MAPE :  34.44786909025234
```



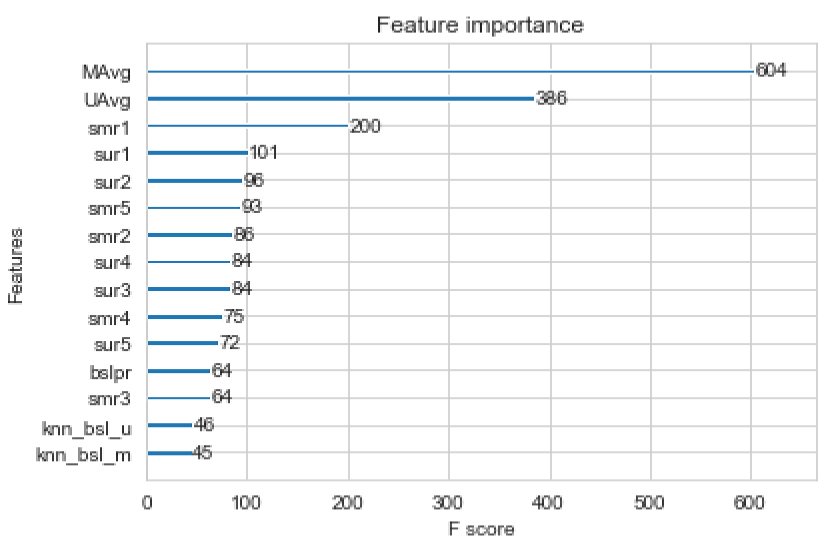
In [85]:
```
%%time
#Tuning the parameters to be given

reg_alpha2=[0, 0.001, 0.005, 0.01, 0.05]

model=xgb.XGBRegressor(n_jobs=-1, random_state=15,learning_rate=0.1,gamma=0.1,n_estimators=300)
#Creating dictionary of parameters to be considered
Param_4= dict(reg_alpha=reg_alpha2)

#Hyperarameter tuning the parameters using Gridsearch cross_validation technique
Gridsearch_tuning(model,Param_4, x_train, y_train)
```

```
Best: -0.745615 using {'reg_alpha': 0.05}
-0.745686 (0.023405) with: {'reg_alpha': 0}
-0.745690 (0.023404) with: {'reg_alpha': 0.001}
-0.745815 (0.023472) with: {'reg_alpha': 0.005}
-0.745704 (0.023409) with: {'reg_alpha': 0.01}
-0.745615 (0.023498) with: {'reg_alpha': 0.05}
Wall time: 6min 39s
```

Training the model with the tuned hyperparameters

In [87]:
```
# declare the model
Tuned2_xgb_knn_bsl = xgb.XGBRegressor(n_jobs=-1, random_state=15,learning_rate=0.1,gamma=0.1,n_estimat
ors=300,reg_alpha=0.05)
knn_bsl2_train_results, knn_bsl2_test_results = run_xgboost(Tuned2_xgb_knn_bsl, x_train, y_train, x_te
st, y_test)

# store the results in models_evaluations dictionaries
models_evaluation_train['Tuned_xgb_knn_bsl'] = knn_bsl2_train_results
models_evaluation_test['Tuned_xgb_knn_bsl'] = knn_bsl2_test_results


xgb.plot_importance(Tuned2_xgb_knn_bsl)
plt.show()
```

```
Training the model..
Done. Time taken : 0:00:36.202536

Done

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

TEST DATA
------------------------------
RMSE :  1.0914801841889183
MAPE :  34.41030100746394
```



4.4.6 Matrix Factorization Techniques

4.4.6.1 SVD Matrix Factorization User Movie intractions

In [97]:
```python
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 [98]:
```python
# 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 : 0:00:40.333576

Evaluating the model with train data..
time taken : 0:00:05.593662
---------------
Train Data
---------------
RMSE : 0.6746731413267192

MAPE : 20.05479554670084

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.878684
---------------
Test Data
---------------
RMSE : 1.0848131688964942

MAPE : 34.42227772904655

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:00:46.805922
```

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

In [93]:
```
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) $
```

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

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

In [94]:
```
# 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 : 0:27:16.019852

Evaluating the model with train data..
time taken : 0:01:09.106627
---------------
Train Data
---------------
RMSE : 0.6641918784333875

MAPE : 19.24213231265533

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:01.115866
---------------
Test Data
---------------
RMSE : 1.0854698955190794

MAPE : 34.387935054377735

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:28:26.242345
```

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

Preparing Train data

In [99]:

```
# 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[99]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	...	smr4	smr5	UAvg	MAvg	rating	bslpr
0	174683	10	3.587581	5.0	5.0	3.0	4.0	4.0	3.0	5.0	...	3.0	2.0	3.882353	3.611111	5	3.681393
1	233949	10	3.587581	4.0	4.0	5.0	1.0	3.0	2.0	3.0	...	3.0	3.0	2.692308	3.611111	3	3.720150

2 rows × 21 columns

Preparing Test data

In [100]:

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

reg_test_df.head(2)
```

Out[100]:

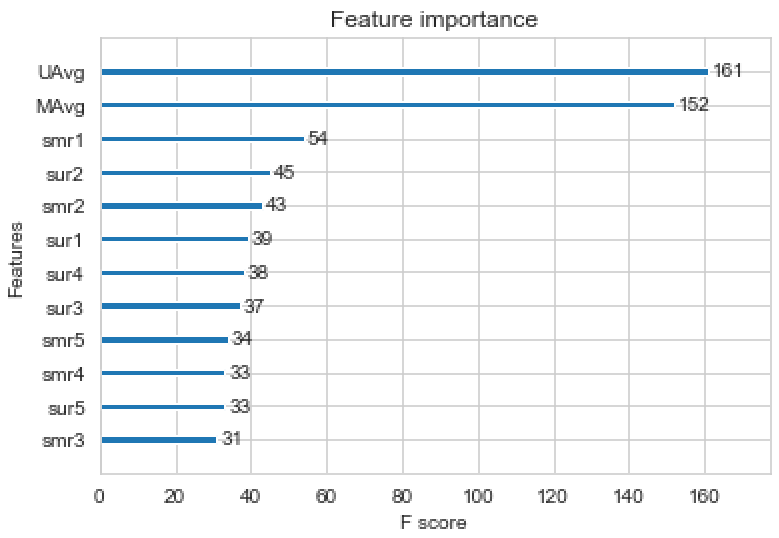
	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	...	smr4	smr5	
0	1129620	2	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	...	3.587581	3.587581	3
1	3321	5	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	...	3.587581	3.587581	3

2 rows × 21 columns

In [101]:

```
# 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..
Done. Time taken : 0:00:14.679801

Done

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

TEST DATA
------------------------------
RMSE :  1.0898255874531768
MAPE :  34.47749436884959
```



Gridsearch over XgBoost with 13 features + Surprise Baseline + Surprise KNNbaseline + MF Techniques

In [107]:

```
subsample=[i/10.0 for i in range(6,11)]
colsample_bytree= [i/10.0 for i in range(6,11)]

#Creating dictionary of parameters to be considered
param_5=dict(subsample=subsample,colsample_bytree=colsample_bytree)

model=xgb.XGBRegressor(n_jobs=-1, random_state=15,learning_rate=0.1,gamma=0.1,n_estimators=300,reg_alp
ha=1)

#Hyperarameter tuning the parameters using Gridsearch cross_validation technique
Gridsearch_tuning(model,param_5, x_train, y_train)
```

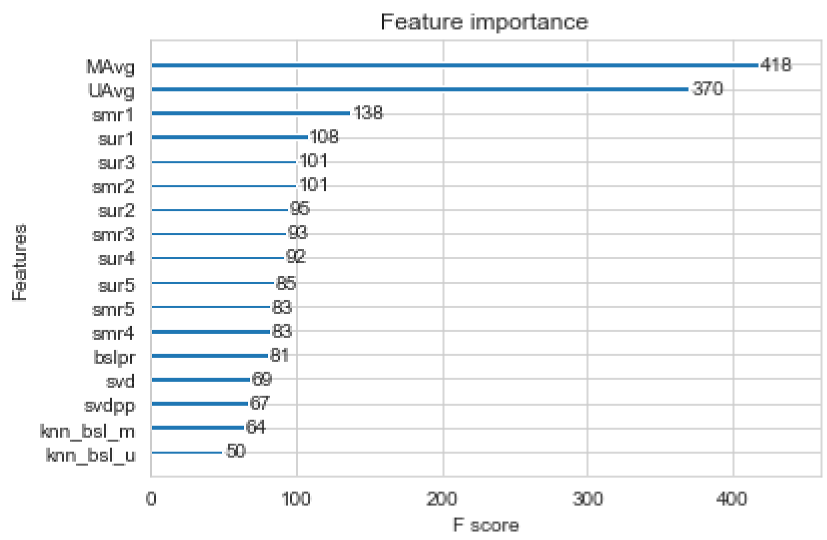
```
Best: -0.744907 using {'colsample_bytree': 0.6, 'subsample': 0.7}
-0.745121 (0.023280) with: {'colsample_bytree': 0.6, 'subsample': 0.6}
-0.744907 (0.023323) with: {'colsample_bytree': 0.6, 'subsample': 0.7}
-0.745282 (0.023616) with: {'colsample_bytree': 0.6, 'subsample': 0.8}
-0.745394 (0.023559) with: {'colsample_bytree': 0.6, 'subsample': 0.9}
-0.745492 (0.023607) with: {'colsample_bytree': 0.6, 'subsample': 1.0}
-0.745349 (0.023623) with: {'colsample_bytree': 0.7, 'subsample': 0.6}
-0.745139 (0.023534) with: {'colsample_bytree': 0.7, 'subsample': 0.7}
-0.745439 (0.023982) with: {'colsample_bytree': 0.7, 'subsample': 0.8}
-0.745333 (0.023846) with: {'colsample_bytree': 0.7, 'subsample': 0.9}
-0.745242 (0.023982) with: {'colsample_bytree': 0.7, 'subsample': 1.0}
-0.745383 (0.023540) with: {'colsample_bytree': 0.8, 'subsample': 0.6}
-0.745364 (0.023553) with: {'colsample_bytree': 0.8, 'subsample': 0.7}
-0.745493 (0.023788) with: {'colsample_bytree': 0.8, 'subsample': 0.8}
-0.745647 (0.023875) with: {'colsample_bytree': 0.8, 'subsample': 0.9}
-0.745521 (0.023853) with: {'colsample_bytree': 0.8, 'subsample': 1.0}
-0.745344 (0.023610) with: {'colsample_bytree': 0.9, 'subsample': 0.6}
-0.745340 (0.023487) with: {'colsample_bytree': 0.9, 'subsample': 0.7}
-0.745796 (0.023526) with: {'colsample_bytree': 0.9, 'subsample': 0.8}
-0.745692 (0.023717) with: {'colsample_bytree': 0.9, 'subsample': 0.9}
-0.745670 (0.023738) with: {'colsample_bytree': 0.9, 'subsample': 1.0}
-0.745613 (0.023179) with: {'colsample_bytree': 1.0, 'subsample': 0.6}
-0.745506 (0.023239) with: {'colsample_bytree': 1.0, 'subsample': 0.7}
-0.745612 (0.023394) with: {'colsample_bytree': 1.0, 'subsample': 0.8}
-0.745490 (0.023502) with: {'colsample_bytree': 1.0, 'subsample': 0.9}
-0.745660 (0.023357) with: {'colsample_bytree': 1.0, 'subsample': 1.0}
```

Training the model with the tuned hyperparameters

In [108]:

```
Tuned_xgb_final = xgb.XGBRegressor(n_jobs=-1, random_state=15,learning_rate=0.1,gamma=0.1,n_estimators
=300,reg_alpha=1,colsample_bytree= 0.6, subsample= 0.7)
xgb_final_train_results, xgb_final_test_results = run_xgboost(Tuned_xgb_final, x_train, y_train, x_tes
t, y_test)

# store the results in models_evaluations dictionaries
models_evaluation_train['xgb_final'] = xgb_final_train_results
models_evaluation_test['xgb_final'] =  xgb_final_test_results


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

```
Training the model..
Done. Time taken : 0:00:39.567212

Done

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

TEST DATA
------------------------------
RMSE :  1.0909527321845127
MAPE :  34.430678585068414
```

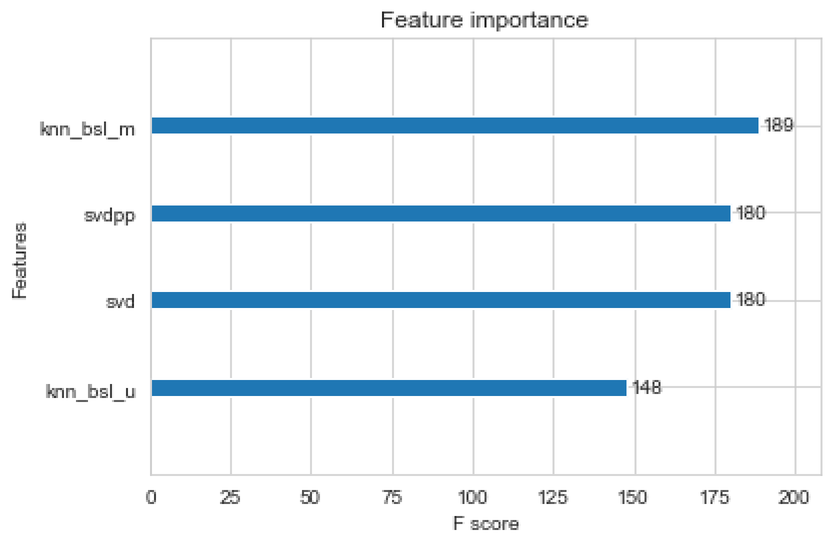


4.4.8 XgBoost with Surprise Baseline + Surprise KNNbaseline + MF Techniques

In [109]:

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

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


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

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

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

```
Training the model..
Done. Time taken : 0:00:07.819102

Done

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

TEST DATA
------------------------------
RMSE :  1.0933435079827543
MAPE :  35.00580733092455
```



Gridsearch over XgBoost with Surprise Baseline + Surprise KNNbaseline + MF Techniques

In [110]:

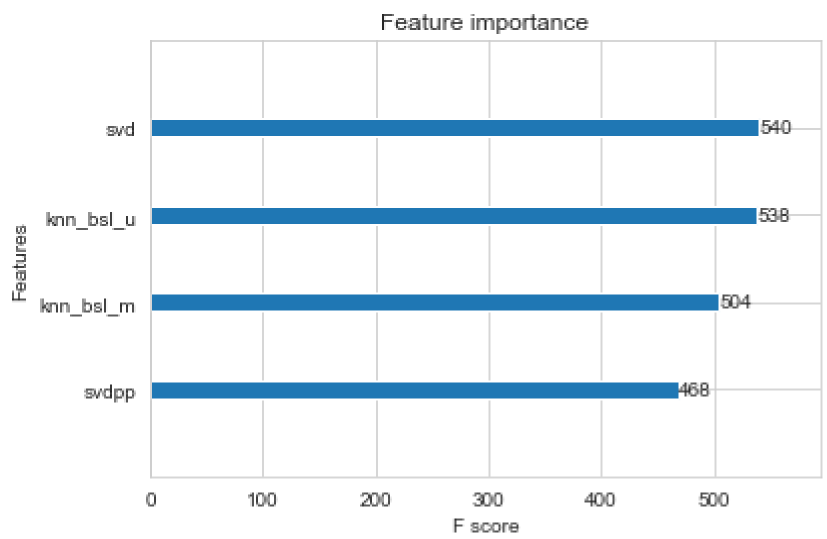
```
subsample=[i/10.0 for i in range(6,11)]
colsample_bytree= [i/10.0 for i in range(6,11)]

#Creating dictionary of parameters to be considered
param_6=dict(subsample=subsample,colsample_bytree=colsample_bytree)

model=xgb.XGBRegressor(n_jobs=-1, random_state=15,learning_rate=0.1,gamma=0.1,n_estimators=300,reg_alp
ha=1)

#Hyperarameter tuning the parameters using Gridsearch cross_validation technique
Gridsearch_tuning(model,param_6, x_train, y_train)
```

```
Best: -1.173900 using {'colsample_bytree': 0.6, 'subsample': 1.0}
-1.174079 (0.032434) with: {'colsample_bytree': 0.6, 'subsample': 0.6}
-1.174096 (0.032440) with: {'colsample_bytree': 0.6, 'subsample': 0.7}
-1.174019 (0.032441) with: {'colsample_bytree': 0.6, 'subsample': 0.8}
-1.173913 (0.032339) with: {'colsample_bytree': 0.6, 'subsample': 0.9}
-1.173900 (0.032381) with: {'colsample_bytree': 0.6, 'subsample': 1.0}
-1.174079 (0.032434) with: {'colsample_bytree': 0.7, 'subsample': 0.6}
-1.174096 (0.032440) with: {'colsample_bytree': 0.7, 'subsample': 0.7}
-1.174019 (0.032441) with: {'colsample_bytree': 0.7, 'subsample': 0.8}
-1.173913 (0.032339) with: {'colsample_bytree': 0.7, 'subsample': 0.9}
-1.173900 (0.032381) with: {'colsample_bytree': 0.7, 'subsample': 1.0}
-1.174276 (0.032417) with: {'colsample_bytree': 0.8, 'subsample': 0.6}
-1.174231 (0.032418) with: {'colsample_bytree': 0.8, 'subsample': 0.7}
-1.174106 (0.032434) with: {'colsample_bytree': 0.8, 'subsample': 0.8}
-1.174039 (0.032399) with: {'colsample_bytree': 0.8, 'subsample': 0.9}
-1.174065 (0.032434) with: {'colsample_bytree': 0.8, 'subsample': 1.0}
-1.174276 (0.032417) with: {'colsample_bytree': 0.9, 'subsample': 0.6}
-1.174231 (0.032418) with: {'colsample_bytree': 0.9, 'subsample': 0.7}
-1.174106 (0.032434) with: {'colsample_bytree': 0.9, 'subsample': 0.8}
-1.174039 (0.032399) with: {'colsample_bytree': 0.9, 'subsample': 0.9}
-1.174065 (0.032434) with: {'colsample_bytree': 0.9, 'subsample': 1.0}
-1.174377 (0.032347) with: {'colsample_bytree': 1.0, 'subsample': 0.6}
-1.174245 (0.032394) with: {'colsample_bytree': 1.0, 'subsample': 0.7}
-1.174236 (0.032465) with: {'colsample_bytree': 1.0, 'subsample': 0.8}
-1.174137 (0.032412) with: {'colsample_bytree': 1.0, 'subsample': 0.9}
-1.174090 (0.032382) with: {'colsample_bytree': 1.0, 'subsample': 1.0}
```

Training the model with the tuned hyperparameters

In [114]:
```python
Tuned_xgb_all_models = xgb.XGBRegressor(n_jobs=-1, random_state=15,learning_rate=0.1,gamma=0.1,n_estim
ators=300,reg_alpha=1,colsample_bytree= 0.6, subsample= 1.0)
Tuned_train_results, Tuned_test_results = run_xgboost(Tuned_xgb_all_models,x_train,y_train,x_test,y_te
st)

# store the results in models_evaluations dictionaries
models_evaluation_train['Tuned_xgb_all_models'] = Tuned_train_results
models_evaluation_test['Tuned_xgb_all_models'] = Tuned_test_results

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

```
Training the model..
Done. Time taken : 0:00:23.619866

Done

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

TEST DATA
------------------------------
RMSE :  1.0934524502733705
MAPE :  35.00429112881574
```



4.5 Comparision between all models

In [116]:
```python
# Saving our TEST_RESULTS into a dataframe so that you don't have to run it again
pd.DataFrame(models_evaluation_test).to_csv('small_sample_results.csv')
models = pd.read_csv('small_sample_results.csv', index_col=0)
models.loc['rmse'].sort_values()
```

```
Out[116]: bsl_algo                 1.084696782600206
          svd                     1.0848131688964942
          knn_bsl_u               1.0850618463554647
          knn_bsl_m               1.0852678745012594
          svdpp                   1.0854698955190794
          first_algo              1.0898255874531768
          xgb_bsl                 1.0898255874531768
          xgb_knn_bsl             1.0898255874531768
          Tuned_first_algo        1.0904043649061108
          xgb_final               1.0909527321845127
          Tuned_xgb_knn_bsl       1.0914801841889183
          Tuned_xgb_bsl           1.0922257160977793
          xgb_all_models          1.0933435079827543
          Tuned_xgb_all_models    1.0934524502733705
          Name: rmse, dtype: object
```

```
In [113]: print("-"*100)
          print("Total time taken to run this entire notebook ( with saved files) is :",datetime.now()-globalsta
          rt)
```

```
----------------------------------------------------------------------------------------------------
Total time taken to run this entire notebook ( with saved files) is : 6:17:44.154886
```

Conclusions

- I have completed both the tasks as instructed and the observations is as follows:
    1. I have taken a sample size of Train set as {25000,3000} and Test set as {20000,2000} for training the different models in the assignment with "rmse" and "mape" as a metric and all the scores are mentioned above properly.
    2. I also have done the hyperparameter tuning by doing gridsearch on every xgboost model.
    3. I have tuned almost 7 parameters of the xgboost regressor model.
    4. The Rmse scores didn't decresed much but the Mape score decreased I think if even larger sample is considered the rmse can be decreased further with even more hyperparameter tuning

```
In [ ]:
```