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 :

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 subsequen
t 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.
```

2.1.2 Example Data point

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

```
2381599,3,2005-09-12
525356,2,2004-07-11
1910569,4,2004-04-12
2263586,4,2004-08-20
2421815,2,2004-02-26
1009622,1,2005-01-19
1481961,2,2005-05-24
401047,4,2005-06-03
2179073,3,2004-08-29
1434636,3,2004-05-01
93986,5,2005-10-06
1308744,5,2005-10-29
2647871,4,2005-12-30
1905581,5,2005-08-16
2508819,3,2004-05-18
1578279,1,2005-05-19
1159695,4,2005-02-15
2588432,3,2005-03-31
2423091,3,2005-09-12
470232,4,2004-04-08
2148699,2,2004-06-05
1342007,3,2004-07-16
466135,4,2004-07-13
2472440,3,2005-08-13
1283744,3,2004-04-17
1927580,4,2004-11-08
716874,5,2005-05-06
4326,4,2005-10-29
```

2.2 Mapping the real world problem to a Machine Learning Problem

2.2.1 Type of Machine Learning Problem

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

2.2.2 Performance metric

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

2.2.3 Machine Learning Objective and Constraints

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

In [3]:

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

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

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

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

3. Exploratory Data Analysis

3.1 Preprocessing

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

In [8]:

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

```
Reading ratings from combined_data_1.txt...
Done.

Reading ratings from combined_data_2.txt...
Done.

Reading ratings from combined_data_3.txt...
Done.

Reading ratings from combined_data_4.txt...
Done.

Time taken : 0:09:53.069898
```

In [9]:

```
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 [10]:

```
df.head()
```

Out[10]:

	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 [11]:

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

Out[11]:

```
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 [12]:

```
# 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 [13]:

```
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 [14]:

```
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 [15]:

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

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

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

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

In [16]:

```
# 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 [18]:

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

```
Test data
--------------------------------------------------

Total no of ratings : 20096102
```

```
Total No of Users    : 349312
Total No of movies   : 17757
```

3.3 Exploratory Data Analysis on Train data

In [19]:

```
# 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 [39]:

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

plt.show()
```

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

In [21]:

```
# 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()
```

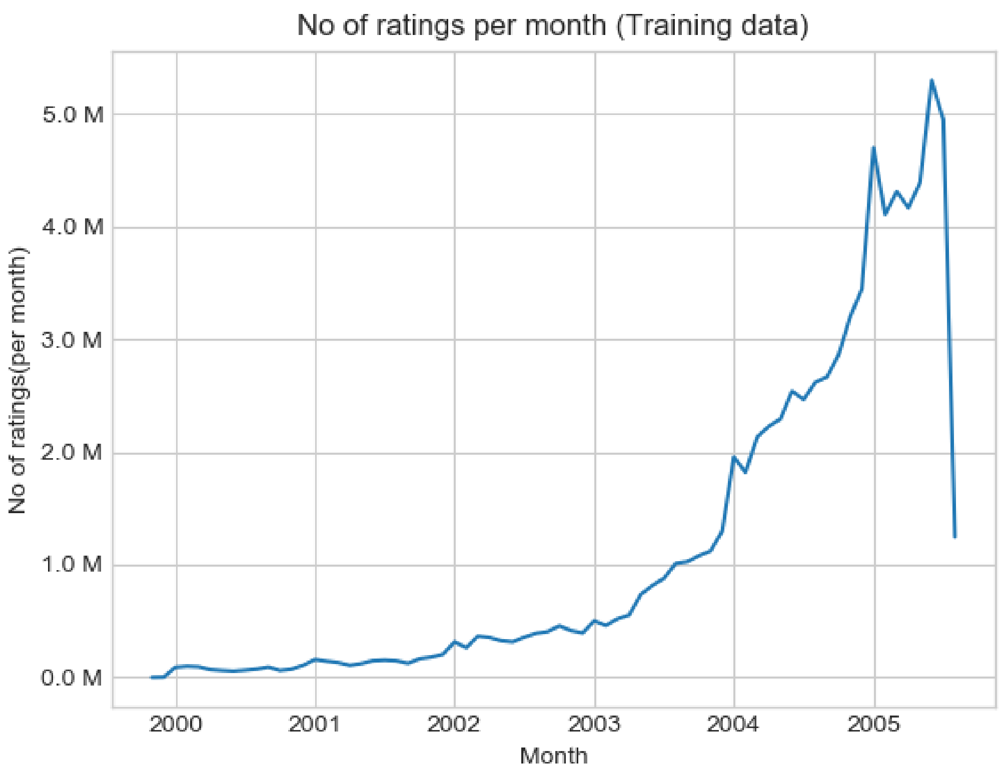
Out[21]:

	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 [43]:

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



3.3.3 Analysis on the Ratings given by user

In [46]:

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

```
no_of_rated_movies_per_user.head()
```

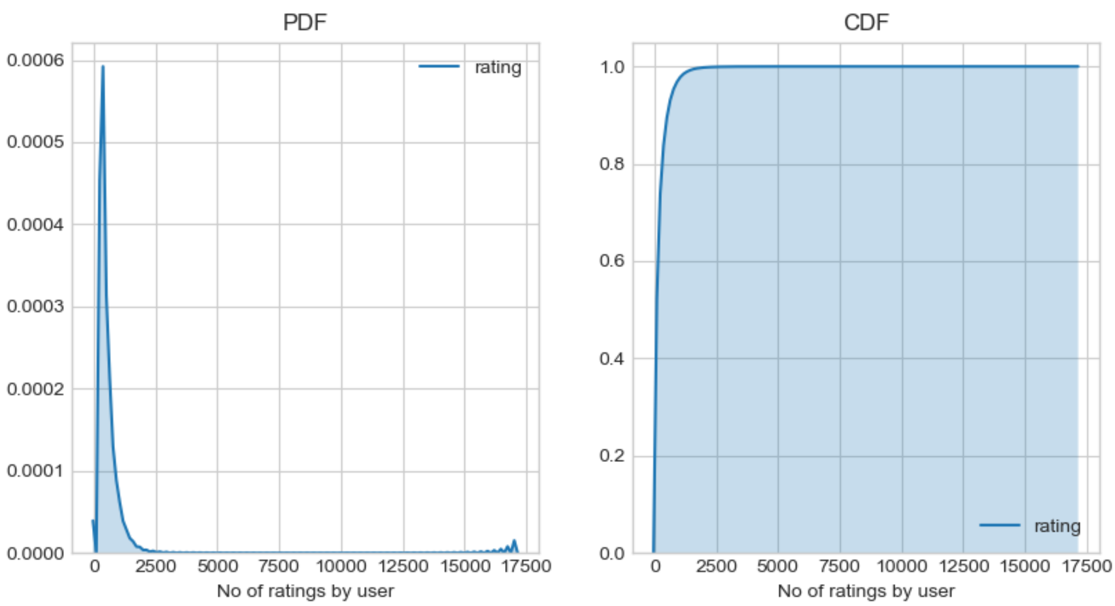
Out[46]:

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

In [24]:

```
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()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



In [25]:

```
no_of_rated_movies_per_user.describe()
```

Out[25]:

```
count    405041.000000
mean        198.459921
std         290.793238
min           1.000000
25%          34.000000
50%          89.000000
75%         245.000000
```

```
75%          245.000000
max        17112.000000
Name: rating, dtype: float64
```

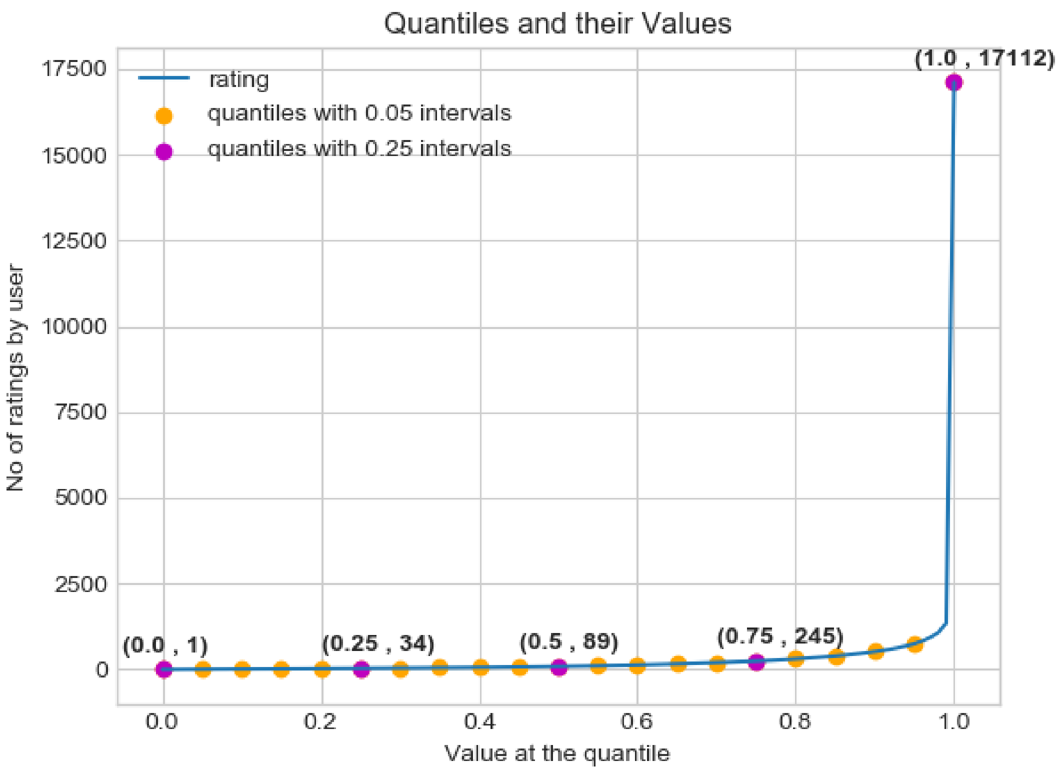
> *There, is something interesting going on with the quantiles..*

In [50]:

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

In [51]:

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

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


plt.show()
```



In [28]:

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

Out[28]:

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

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

In [29]:

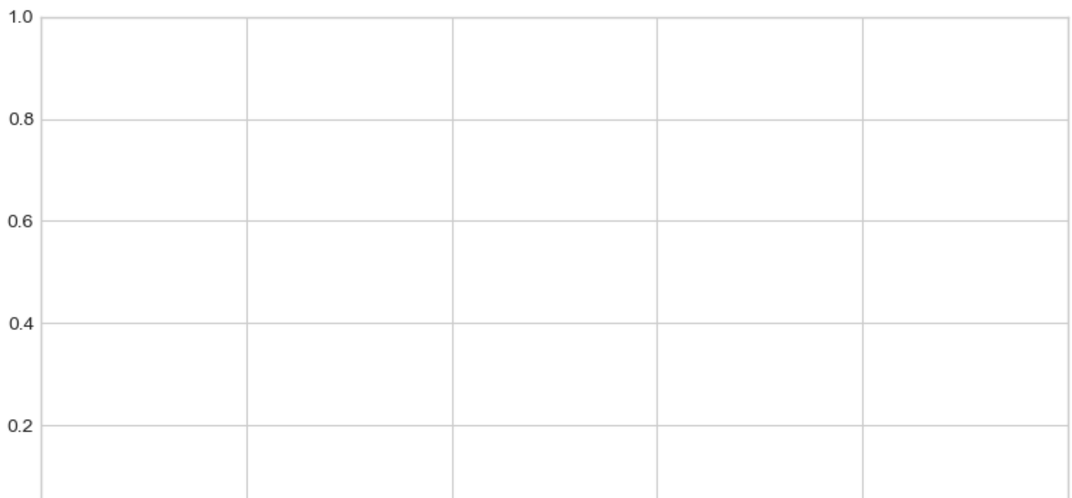
```
print('\n No of ratings at last 5 percentile : {}\n'.format(sum(no_of_rated_movies_per_user>= 749)
) )
```

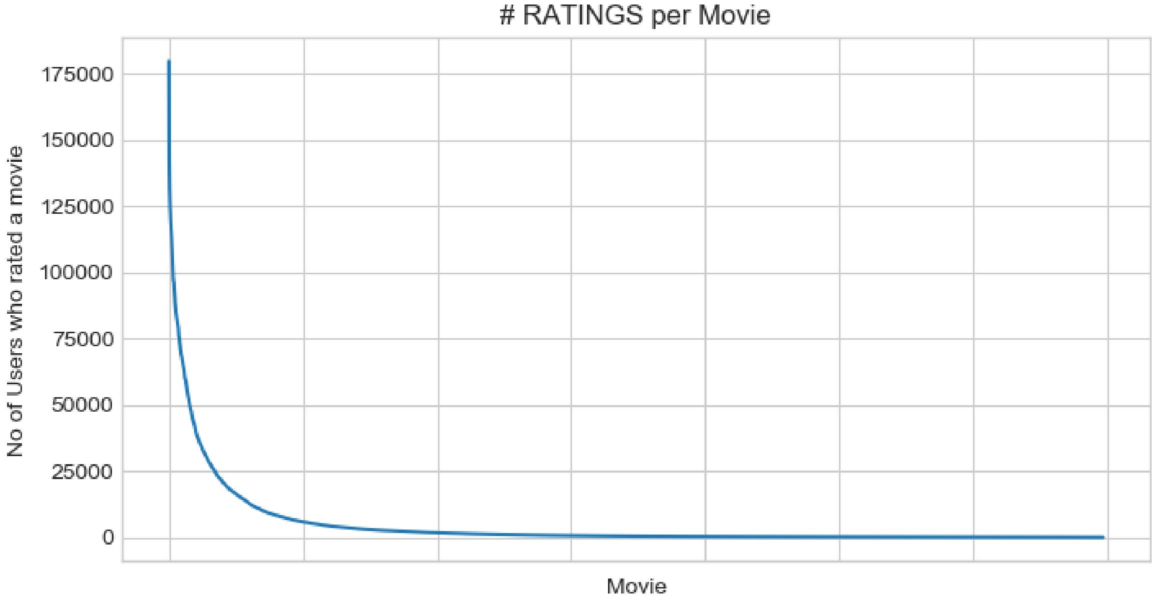
```
 No of ratings at last 5 percentile : 20305
```

3.3.4 Analysis of ratings of a movie given by a user

In [57]:

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

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

plt.show()
```





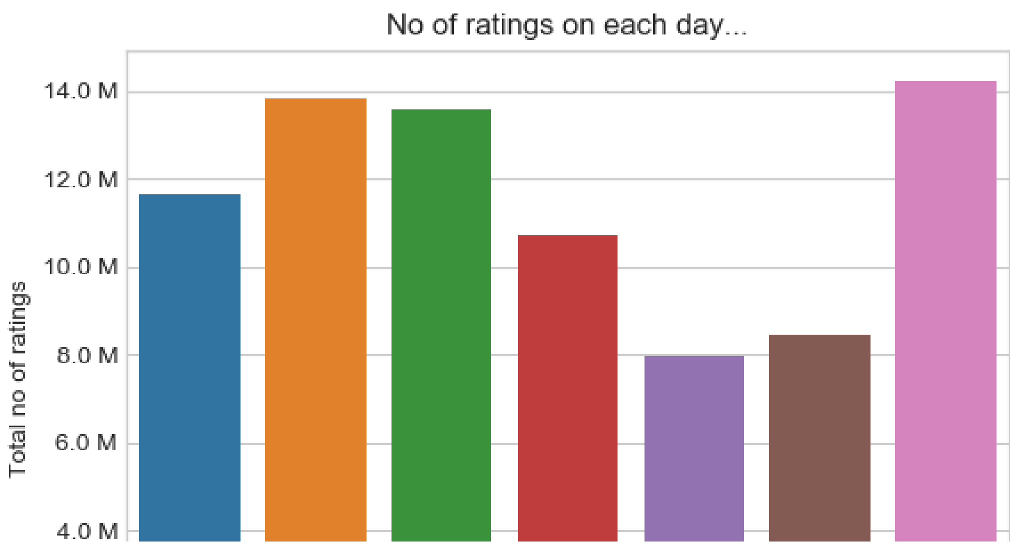
- **It is very skewed.. just like nunmber of ratings given per user.**

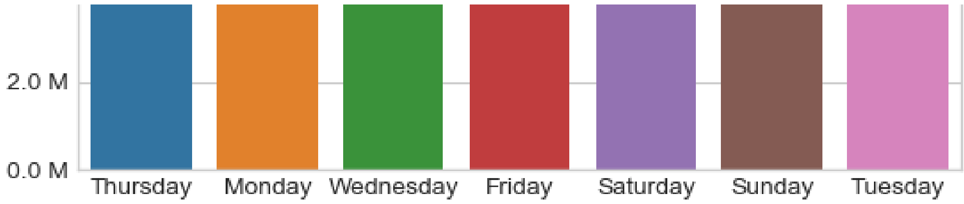
```
- There are some movies (which are very popular) which are rated by huge number of users.

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

3.3.5 Number of ratings on each day of the week

In [31]:

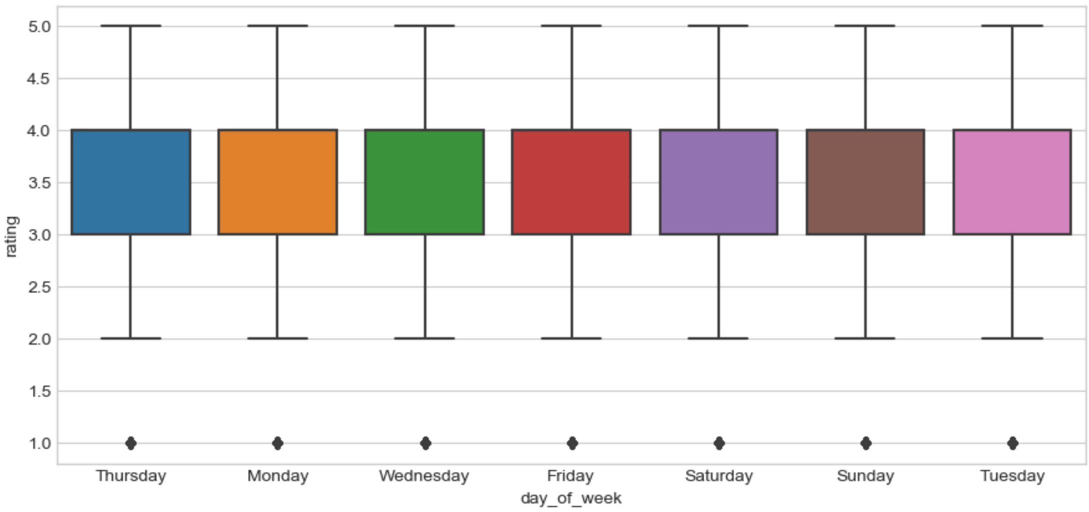
```
fig, ax = plt.subplots()
sns.countplot(x='day_of_week', data=train_df, ax=ax)
plt.title('No of ratings on each day...')
plt.ylabel('Total no of ratings')
plt.xlabel('')
ax.set_yticklabels([human(item, 'M') for item in ax.get_yticks()])
plt.show()
```





In [32]:

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




```
0:00:25.477118
```

In [58]:

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

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

3.3.6 Creating sparse matrix from data frame

3.3.6.1 Creating sparse matrix from train data frame

In [66]:

```
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..
0:00:05.587454
```

The Sparsity of Train Sparse Matrix

In [67]:

```
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 [62]:

```
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..
0:00:07.326708
```

0:00:07.326708

The Sparsity of Test data Matrix

In [65]:

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

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

```
Sparsity Of Test matrix : 99.95731772988694 %
```

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

In [10]:

```
# 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 [69]:

```
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[69]:

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

3.3.7.2 finding average rating per user

In [70]:

```
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

3.3.7.3 finding average rating per movie

In [71]:

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

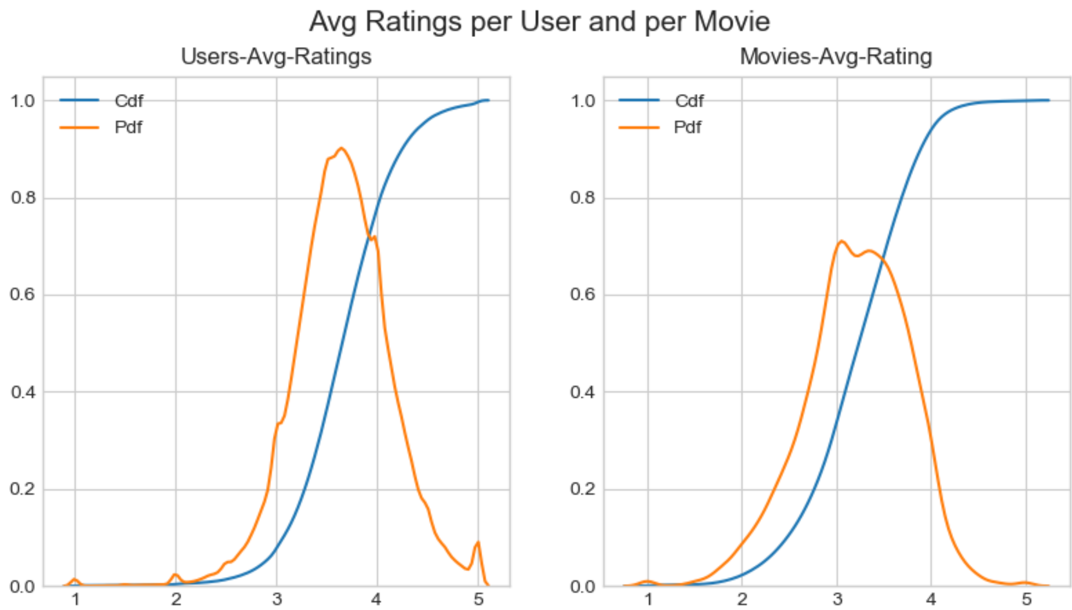
```
 AVerage rating of movie 15 : 3.3038461538461537
```

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

In [72]:

```
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)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-t
uple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[s
eq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will r
esult either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```




```
0:01:19.181116
```

3.3.8 Cold Start problem

3.3.8 Cold Start problem

3.3.8.1 Cold Start problem with Users

In [73]:

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

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

np.round((new_users/total_users)*100, 2)))
```

```
Total number of Users  : 480189

Number of Users in Train data : 405041

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

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

3.3.8.2 Cold Start problem with Movies

In [74]:

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

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

np.round((new_movies/total_movies)*100, 2)))
```

```
Total number of Movies  : 17770

Number of Users in Train data : 17424

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

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

3.4 Computing Similarity matrices

3.4.1 Computing User-User Similarity matrix

1. Calculating User User Similarity_Matrix is **not very easy**(*unless you have huge Computing Power and lots of time*) because of number of. usersbeing lare.
    - You can try if you want to. Your system could crash or the program stops with **Memory Error**

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

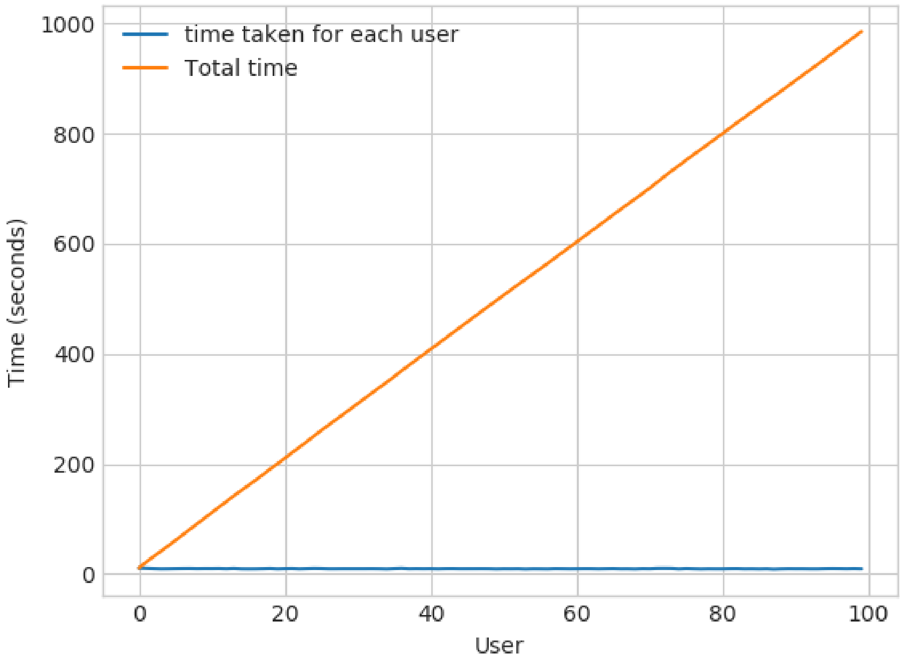
In [75]:

```python
from sklearn.metrics.pairwise import cosine_similarity


def compute_user_similarity(sparse_matrix, compute_for_few=False, top = 100, verbose=False, verb_fo
r_n_rows = 20,
                            draw_time_taken=True):
    no_of_users, _ = sparse_matrix.shape
    # get the indices of  non zero rows(users) from our sparse matrix
    row_ind, col_ind = sparse_matrix.nonzero()
    row_ind = sorted(set(row_ind)) # we don't have to
    time_taken = list() #  time taken for finding similar users for an user..

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

    start = datetime.now()
    temp = 0

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

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

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


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

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

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

In [0]:

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

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




```
--------------------------------------------------------------------------------------------------------
Time taken : 0:16:33.618931
```

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

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

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

In [0]:

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

start = datetime.now()

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

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

```
0:29:07.069783
```

Here,

- \sum \longleftarrow (netflix_svd.**singular_values_**)
- \bigvee^T \longleftarrow (netflix_svd.**components_**)
- \bigcup is not returned. instead **Projection_of_X** onto the new vectorspace is returned.

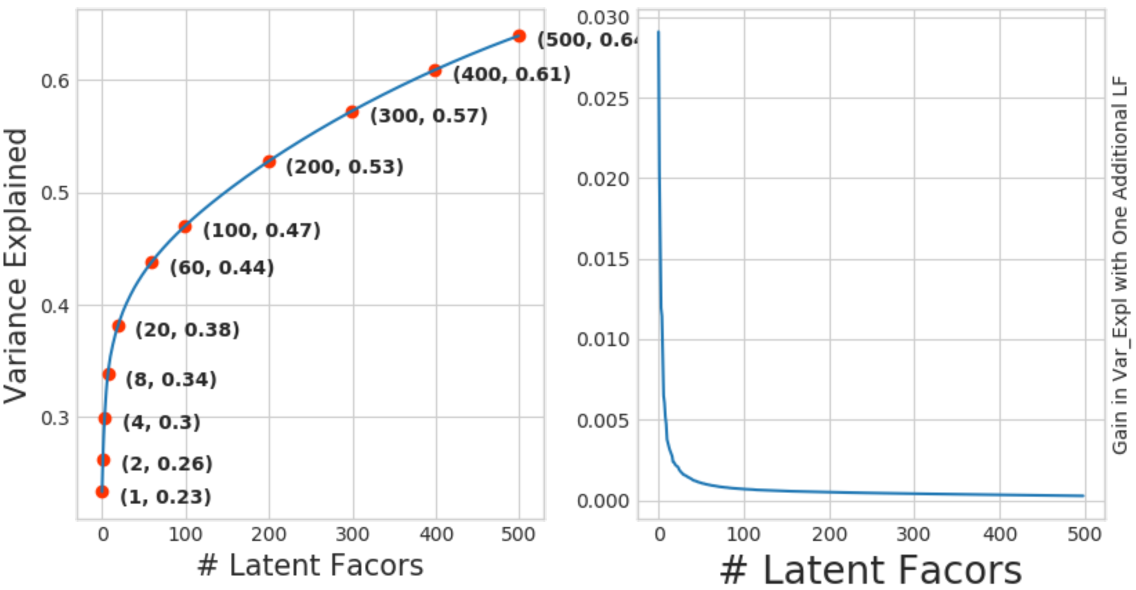
- It uses **randomized svd** internally, which returns **All 3 of them saperately**. Use that instead..

In [0]:

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

In [0]:

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

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

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



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

plt.show()
```



In [0]:

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

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

> I think 500 dimensions is good enough

- By just taking **(20 to 30)** latent factors, explained variance that we could get is **20 %**.
- To take it to **60%**, we have to take **almost 400 latent factors**. It is not fare.

- It basically is the **gain of variance explained**, if we ***add one additional latent factor to it.***

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

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

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

In [0]:

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

```
0:00:45.670265
```

In [0]:

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

Out[0]:

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

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

In [79]:

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

In [80]:

```
trunc_sparse_matrix.shape
```

Out[80]:

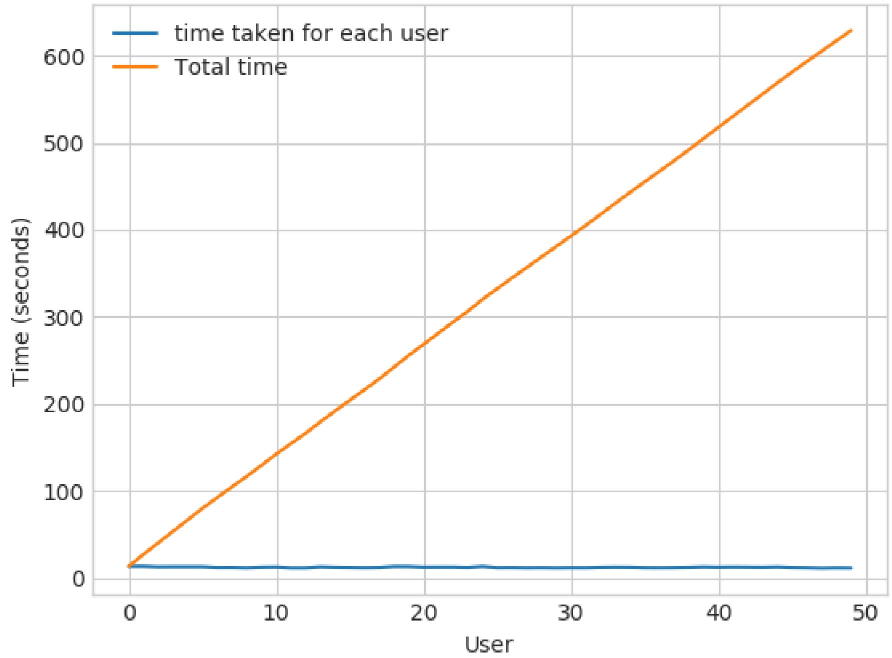
```
(2649430, 500)
```

In [0]:

```
start = datetime.now()
trunc_u_u_sim_matrix, _ = compute_user_similarity(trunc_sparse_matrix, compute_for_few=True, top=50
, verbose=True,
                                                 verb_for_n_rows=10)
```

```
print("-"*50)
print("time:",datetime.now()-start)
```

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




```
--------------------------------------------------
time: 0:10:52.658092
```

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

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

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

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

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

-An alternative is to compute similar users for a particular user, whenenver required (**ie., Run time**)

```
- We maintain a binary Vector for users, which tells us whether we already computed or
not..
- ***If not*** :
    - Compute top (let's just say, 1000) most similar users for this given user, and add
this to our datastructure, so that we can just access it(similar users) without recomputing
```

```
it again.
    -
- ***If It is already Computed***:
    - Just get it directly from our datastructure, which has that information.
    - In production time, We might have to recompute similarities, if it is computed a long
time ago. Because user preferences changes over time. If we could maintain some kind of
Timer, which when expires, we have to update it ( recompute it ).
    -
- ***Which datastructure to use:***
    - It is purely implementation dependant.
    - One simple method is to maintain a **Dictionary Of Dictionaries**.
        -
        - **key    :** _userid_
        - __value__: _Again a dictionary_
            - __key__  : _Similar User_
            - __value__: _Similarity Value_
```

3.4.2 Computing Movie-Movie Similarity matrix

In [81]:

```
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 seems you don't have that file. Computing movie_movie similarity...
Done..
Saving it to disk without the need of re-computing it again..
Done..
It's a  (17771, 17771)  dimensional matrix
0:13:52.977365
```

In [82]:

```
m_m_sim_sparse.shape
```

Out[82]:

```
(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 [83]:

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

In [84]:

In [84]:

```
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]
```

```
0:00:41.247935
```

Out[84]:

```
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

Does Similarity really works as the way we expected...?
Let's pick some random movie and check for its similar movies....

In [86]:

```
# 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: 0.00 ms
Type conversion took: 80.00 ms
Parser memory cleanup took: 0.00 ms
```

Out[86]:

	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

In [100]:

```
movie_titles
```

Out[100]:

Out[100]:

	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
6	1997.0	Sick
7	1992.0	8 Man
8	2004.0	What the #$*! Do We Know!?
9	1991.0	Class of Nuke 'Em High 2
10	2001.0	Fighter
11	1999.0	Full Frame: Documentary Shorts
12	1947.0	My Favorite Brunette
13	2003.0	Lord of the Rings: The Return of the King: Ext...
14	1982.0	Nature: Antarctica
15	1988.0	Neil Diamond: Greatest Hits Live
16	1996.0	Screamers
17	2005.0	7 Seconds
18	1994.0	Immortal Beloved
19	2000.0	By Dawn's Early Light
20	1972.0	Seeta Aur Geeta
21	2002.0	Strange Relations
22	2000.0	Chump Change
23	2001.0	Clifford: Clifford Saves the Day! / Clifford's...
24	1981.0	My Bloody Valentine
25	1997.0	Inspector Morse 31: Death Is Now My Neighbour
26	2004.0	Never Die Alone
27	1962.0	Sesame Street: Elmo's World: The Street We Liv...
28	2002.0	Lilo and Stitch
29	2001.0	Boycott
30	2003.0	Something's Gotta Give
...	...	...
17741	2004.0	Ginger Snaps 2: Unleashed
17742	1995.0	Catherine the Great
17743	2003.0	Better Luck Tomorrow
17744	2004.0	NASCAR: Tony Stewart Smoke
17745	2002.0	Russell Simmons Presents Def Poetry: Season 1
17746	1991.0	Godzilla & Mothra: Battle for Earth / Vs. King...
17747	1991.0	Eric Clapton: 24 Nights
17748	2005.0	Dog the Bounty Hunter: The Best of Season 1
17749	1985.0	No End
17750	2005.0	The Hee Haw Collection: Vol. 4
17751	1993.0	Highlander: Season 2

movie_id	year_of_release	title
17752	2003.0	Out of Order
17753	1997.0	Maslin Beach
17754	1999.0	On the Ropes
17755	2003.0	L/R: Licensed by Royalty
17756	1935.0	The 39 Steps
17757	2002.0	Ulysses S. Grant: Warrior / President: America...
17758	1979.0	Prophecy
17759	1972.0	The Big Bird Cage
17760	2004.0	Lightning Bug
17761	2003.0	Levity
17762	1997.0	Gattaca
17763	1978.0	Interiors
17764	1998.0	Shakespeare in Love
17765	1969.0	Godzilla's Revenge
17766	2002.0	Where the Wild Things Are and Other Maurice Se...
17767	2004.0	Fidel Castro: American Experience
17768	2000.0	Epoch
17769	2003.0	The Company
17770	2003.0	Alien Hunter

17770 rows × 2 columns

Similar Movies for 'Vampire Journals'

In [109]:

```
mv_id = 8824

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

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

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

```
Movie -----> Godzilla

It has 855 Ratings from users.

We have 17333 movies which are similarto this  and we will get only top most..
```

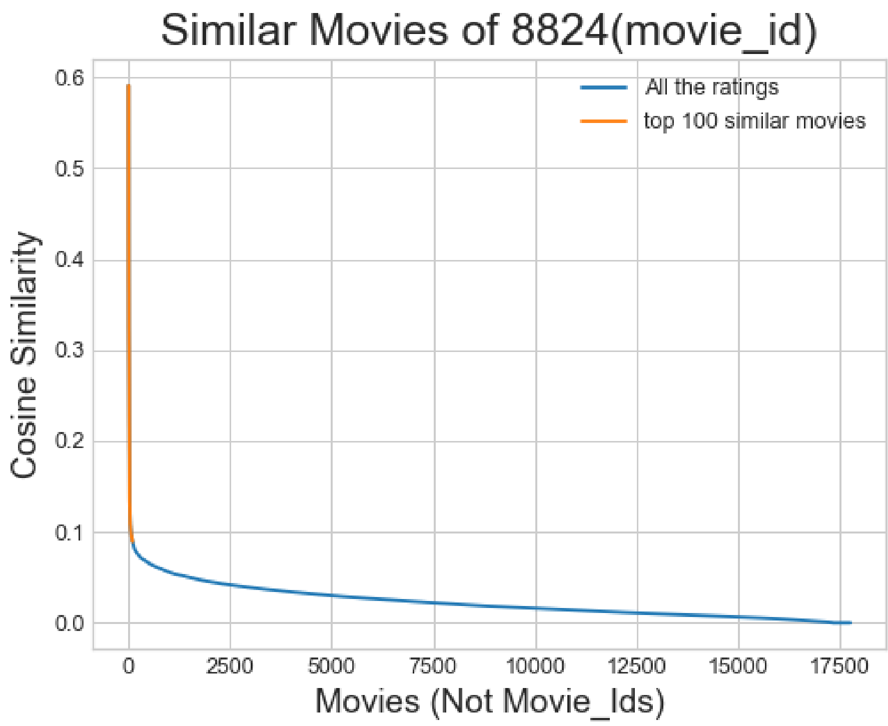
In [110]:

```
similarities = m_m_sim_sparse[mv_id].toarray().ravel()

similar_indices = similarities.argsort()[::-1][1:]

similarities[similar_indices]

sim_indices = similarities.argsort()[::-1][1:] # It will sort and reverse the array and ignore its
similarity (ie.,1)
                                               # and return its indices(movie_ids)
```

In [111]:

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

```
plt.ylabel("Cosine Similarity",fontsize=15)
plt.legend()
plt.show()
```



Top 10 similar movies

In [112]:

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

Out[112]:

	year_of_release	title
movie_id		
12640	2000.0	Godzilla vs. Megaguirus
4461	2002.0	Godzilla Against Mechagodzilla
17746	1991.0	Godzilla & Mothra: Battle for Earth / Vs. King...
15123	1995.0	Godzilla vs. Destroyah / Godzilla vs. Space Go...
7228	1996.0	Gamera 2: Attack of Legion
8656	1993.0	Godzilla vs. Mechagodzilla II
2652	1999.0	Gamera 3: Revenge of Iris
13331	1995.0	Gamera 1: Guardian of the Universe
7140	2003.0	Godzilla: Tokyo S.O.S.
10642	1999.0	Godzilla 2000: Millennium

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

4. Machine Learning Models

In [11]:

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

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

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

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

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

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

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

    return sample_sparse_matrix
```

4.1 Sampling Data

4.1.1 Build sample train data from the train data

In [12]:

```
start = datetime.now()
path = "sample/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=30000, no_m
ovies=5000,
                                             path = path)

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

```
It is present in your pwd, getting it from disk....
DONE..
```

```
DONE..
0:00:00.192025
```

4.1.2 Build sample test data from the test data

In [13]:

```
start = datetime.now()

path = "sample/sample_test_sparse_matrix.npz"
if os.path.isfile(path):
    print("It is present in your pwd, getting it from disk....")
    # just get it from the disk instead of computing it
    sample_test_sparse_matrix = sparse.load_npz(path)
    print("DONE..")
else:
    # get 5k users and 500 movies from available data
    sample_test_sparse_matrix = get_sample_sparse_matrix(test_sparse_matrix, no_users=13000, no_mov
ies=3000,
                                                path = "sample/sample_test_sparse_matrix.npz")
print(datetime.now() - start)
```

```
It is present in your pwd, getting it from disk....
DONE..
0:00:00.046006
```

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

In [14]:

```
sample_train_averages = dict()
```

4.2.1 Finding Global Average of all movie ratings

In [15]:

```
# 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[15]:

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

4.2.2 Finding Average rating per User

In [16]:

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

```
Average rating of user 1515220 : 3.9655172413793105
```

4.2.3 Finding Average rating per Movie

In [17]:

```
sample_train_averages['movie'] =  get_average_ratings(sample_train_sparse_matrix, of_users=False)
print('\n AVerage rating of movie 15153 :',sample_train_averages['movie'][15153])
```

```
AVerage rating of movie 15153 : 2.6458333333333335
```

4.3 Featurizing data

In [18]:

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

```
 No of ratings in Our Sampled train matrix is : 129286


 No of ratings in Our Sampled test  matrix is : 7333
```

4.3.1 Featurizing data for regression problem

4.3.1.1 Featurizing train data

In [19]:

```
# 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 [ ]:

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


            #--------------------- Ratings by "user"  to similar movies of "movie" ----------------
----
            # compute the similar movies of the "movie"
            movie_sim = cosine_similarity(sample_train_sparse_matrix[:,movie].T,
sample_train_sparse_matrix.T).ravel()
            top_sim_movies = movie_sim.argsort()[::-1][1:] # we are ignoring 'The User' from its si
milar users.
            # get the ratings of most similar movie rated by this user..
            top_ratings = sample_train_sparse_matrix[user, top_sim_movies].toarray().ravel()
```

```
            # we will make it's length "5" by adding user averages to.
            top_sim_movies_ratings = list(top_ratings[top_ratings != 0][:5])
            top_sim_movies_ratings.extend([sample_train_averages['user']
[user]]*(5-len(top_sim_movies_ratings)))
        #     print(top_sim_movies_ratings, end=" : -- ")

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

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

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


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

```
preparing 1763797 tuples for the dataset..

Done for 10000 rows----- 3:11:02.744759
Done for 20000 rows----- 6:18:25.862770
Done for 30000 rows----- 9:24:54.390753
Done for 40000 rows----- 12:32:15.123534
Done for 50000 rows----- 15:41:37.242553
Done for 60000 rows----- 18:52:12.132380
Done for 70000 rows----- 22:01:23.008506
Done for 80000 rows----- 1 day, 1:08:32.933123
Done for 90000 rows----- 1 day, 4:16:57.222332
Done for 100000 rows----- 1 day, 7:23:54.139880
Done for 110000 rows----- 1 day, 10:34:17.101332
Done for 120000 rows----- 1 day, 13:45:20.448499
Done for 130000 rows----- 1 day, 16:57:22.694247
Done for 140000 rows----- 1 day, 20:09:49.620380
Done for 150000 rows----- 1 day, 23:21:51.209467
Done for 160000 rows----- 2 days, 2:33:37.245251
Done for 170000 rows----- 2 days, 5:46:50.005866
Done for 180000 rows----- 2 days, 8:57:45.761173
Done for 190000 rows----- 2 days, 12:07:46.144364
Done for 200000 rows----- 2 days, 15:18:59.125421
Done for 210000 rows----- 2 days, 18:30:15.232341
Done for 220000 rows----- 2 days, 21:41:33.643891
Done for 230000 rows----- 3 days, 0:52:56.623141
Done for 240000 rows----- 3 days, 4:04:05.890146
Done for 250000 rows----- 3 days, 7:14:37.039224
Done for 260000 rows----- 3 days, 10:25:17.064843
Done for 270000 rows----- 3 days, 13:34:02.439495
Done for 280000 rows----- 3 days, 16:43:50.628264
Done for 290000 rows----- 3 days, 19:55:51.466473
Done for 300000 rows----- 3 days, 23:07:35.885368
Done for 310000 rows----- 4 days, 2:19:10.099014
Done for 320000 rows----- 4 days, 5:28:57.891287
Done for 330000 rows----- 4 days, 8:40:32.049101
Done for 340000 rows----- 4 days, 11:52:05.642745
Done for 350000 rows----- 4 days, 15:04:19.508047
Done for 360000 rows----- 4 days, 18:19:02.161202
Done for 370000 rows----- 4 days, 21:33:48.881769
Done for 380000 rows----- 5 days, 0:45:20.960700
Done for 390000 rows----- 5 days, 3:55:58.398069
```

```
Done for 400000 rows----- 5 days, 7:10:33.663050
Done for 410000 rows----- 5 days, 10:24:28.184517
Done for 420000 rows----- 5 days, 13:35:50.579965
Done for 430000 rows----- 5 days, 16:46:29.674388
Done for 440000 rows----- 5 days, 19:58:15.816145
Done for 450000 rows----- 5 days, 23:10:11.836094
Done for 460000 rows----- 6 days, 2:21:33.434357
Done for 470000 rows----- 6 days, 5:32:50.619737
Done for 480000 rows----- 6 days, 8:44:32.477023
Done for 490000 rows----- 6 days, 11:55:22.383157
Done for 500000 rows----- 6 days, 15:06:01.716330
Done for 510000 rows----- 6 days, 18:16:48.839605
Done for 520000 rows----- 6 days, 21:28:12.082101
Done for 530000 rows----- 7 days, 0:37:36.411758
Done for 540000 rows----- 7 days, 3:48:37.061755
Done for 550000 rows----- 7 days, 6:59:17.027078
Done for 560000 rows----- 7 days, 10:09:50.430711
Done for 570000 rows----- 7 days, 13:17:24.479184
Done for 580000 rows----- 7 days, 16:26:56.094592
Done for 590000 rows----- 7 days, 19:37:22.161195
Done for 600000 rows----- 7 days, 22:47:45.795312
Done for 610000 rows----- 8 days, 1:58:26.060634
Done for 620000 rows----- 8 days, 5:08:49.862863
Done for 630000 rows----- 8 days, 8:18:52.062988
Done for 640000 rows----- 8 days, 11:29:31.127036
Done for 650000 rows----- 8 days, 14:38:53.133419
Done for 660000 rows----- 8 days, 17:49:26.577151
Done for 670000 rows----- 8 days, 21:00:16.799221
Done for 680000 rows----- 9 days, 0:10:53.057782
Done for 690000 rows----- 9 days, 3:20:50.338863
Done for 700000 rows----- 9 days, 6:30:45.698627
Done for 710000 rows----- 9 days, 9:40:31.216994
Done for 720000 rows----- 9 days, 12:47:40.369012
Done for 730000 rows----- 9 days, 15:54:10.181594
Done for 740000 rows----- 9 days, 19:01:17.926429
Done for 750000 rows----- 9 days, 22:09:25.015263
Done for 760000 rows----- 10 days, 1:18:10.385608
Done for 770000 rows----- 10 days, 4:27:00.087956
Done for 780000 rows----- 10 days, 7:33:00.705155
Done for 790000 rows----- 10 days, 10:39:21.947811
Done for 800000 rows----- 10 days, 13:47:09.999103
Done for 810000 rows----- 10 days, 16:53:43.588683
Done for 820000 rows----- 10 days, 20:01:50.299855
Done for 830000 rows----- 10 days, 23:11:13.134534
Done for 840000 rows----- 11 days, 2:18:53.744789
Done for 850000 rows----- 11 days, 5:25:20.518272
Done for 860000 rows----- 11 days, 8:32:36.176376
Done for 870000 rows----- 11 days, 11:39:07.402884
Done for 880000 rows----- 11 days, 14:46:15.010932
Done for 890000 rows----- 11 days, 17:54:49.590144
Done for 900000 rows----- 11 days, 21:03:23.767881
Done for 910000 rows----- 12 days, 0:10:22.027832
Done for 920000 rows----- 12 days, 3:18:22.942500
Done for 930000 rows----- 12 days, 6:25:47.316150
Done for 940000 rows----- 12 days, 9:34:40.472381
Done for 950000 rows----- 12 days, 12:41:22.271509
Done for 960000 rows----- 12 days, 15:48:17.253376
Done for 970000 rows----- 12 days, 18:50:32.070786
Done for 980000 rows----- 12 days, 21:53:12.994317
Done for 990000 rows----- 13 days, 0:56:05.312574
Done for 1000000 rows----- 13 days, 3:58:29.451850
Done for 1010000 rows----- 13 days, 7:00:57.365658
Done for 1020000 rows----- 13 days, 10:03:38.422136
Done for 1030000 rows----- 13 days, 13:06:11.532658
Done for 1040000 rows----- 13 days, 16:08:52.750365
Done for 1050000 rows----- 13 days, 19:12:32.418564
Done for 1060000 rows----- 13 days, 22:14:42.767502
Done for 1070000 rows----- 14 days, 1:16:53.297481
Done for 1080000 rows----- 14 days, 4:18:47.944623
Done for 1090000 rows----- 14 days, 7:21:24.711664
Done for 1100000 rows----- 14 days, 10:23:45.536387
Done for 1110000 rows----- 14 days, 13:25:22.947105
```

1. As per the assignment I have choose 30000 users and 5000 movies because of resource limitation I am unable to process it.I have waited for 14 days at end system got crashed.

In [20]:

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


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

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

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

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


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

```
File already exists you don't have to prepare again...
0:00:00.001001
```

Reading from the file to make a Train_dataframe

In [21]:

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

Out[21]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating
0	53406	33	3.581679	4.0	5.0	5.0	4.0	1.0	5.0	2.0	5.0	3.0	1.0	3.370370	4.092437	4
1	99540	33	3.581679	5.0	5.0	5.0	4.0	5.0	3.0	4.0	4.0	3.0	5.0	3.555556	4.092437	3
2	99865	33	3.581679	5.0	5.0	4.0	5.0	3.0	5.0	4.0	4.0	5.0	4.0	3.714286	4.092437	5
3	101620	33	3.581679	2.0	3.0	5.0	5.0	4.0	4.0	3.0	3.0	4.0	5.0	3.584416	4.092437	5
4	112974	33	3.581679	5.0	5.0	5.0	5.0	5.0	3.0	5.0	5.0	5.0	3.0	3.750000	4.092437	5

- **GAvg** : Average rating of all the ratings
- **Similar users rating of this movie**:
  - sur1, sur2, sur3, sur4, sur5 (top 5 similar users who rated that movie..)
- **Similar movies rated by this user**:
  - smr1, smr2, smr3, smr4, smr5 (top 5 similar movies rated by this user..)
- **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 [22]:

```
# 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 [23]:

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

Out[23]:

```
3.581679377504138
```

In [24]:

```
start = datetime.now()

if os.path.isfile('sample/reg_test.csv'):
    print("It is already created...")
else:
    print('preparing {} tuples for the dataset..\n'.format(len(sample_test_ratings)))
```

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

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

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



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

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

            #-----------------prepare the row to be stores in a file-----------------#
            row = list()
            # add usser and movie name first
            row.append(user)
            row.append(movie)
            row.append(sample_train_averages['global']) # first feature
            #print(row)
            # next 5 features are similar_users "movie" ratings
            row.extend(top_sim_users_ratings)
            #print(row)
            # next 5 features are "user" ratings for similar_movies
            row.extend(top_sim_movies_ratings)
            #print(row)
            # Avg_user rating
            try:
                row.append(sample_train_averages['user'][user])
            except KeyError:
                row.append(sample_train_averages['global'])
```

```
                row.append(sample_train_averages['global'])
            except:
                raise
            #print(row)
            # Avg_movie rating
            try:
                row.append(sample_train_averages['movie'][movie])
            except KeyError:
                row.append(sample_train_averages['global'])
            except:
                raise
            #print(row)
            # finalley, The actual Rating of this user-movie pair...
            row.append(rating)
            #print(row)
            count = count + 1

            # add rows to the file opened..
            reg_data_file.write(','.join(map(str, row)))
            #print(','.join(map(str, row)))
            reg_data_file.write('\n')
            if (count)%1000 == 0:
                #print(','.join(map(str, row)))
                print("Done for {} rows----- {}".format(count, datetime.now() - start))
    print("",datetime.now() - start)
```

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

Reading from the file to make a test dataframe

In [25]:

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

Out[25]:

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

- **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 [26]:

```
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 [27]:

```
# 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 [28]:

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

Out[28]:

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

4.4 Applying Machine Learning models

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

    > **keys** : model names(string)
    >
    > **value**: dict(**key** : metric, **value** : value)

In [29]:

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

models_evaluation_train, models_evaluation_test
```

Out[29]:

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

Utility functions for running regression models

In [30]:

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

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

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


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

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

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

    #######################################
    # get the test data predictions and compute rmse and mape
    print('Evaluating Test data')
    y_test_pred = algo.predict(x_test)
    rmse_test, mape_test = get_error_metrics(y_true=y_test.values, y_pred=y_test_pred)
    # store them in our test results dictionary.
    test_results = {'rmse': rmse_test,
                    'mape' : mape_test,
                    'predictions':y_test_pred}
    if verbose:
        print('\nTEST DATA')
        print('-'*30)
        print('RMSE : ', rmse_test)
        print('MAPE : ', mape_test)

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

Utility functions for Surprise modes

In [31]:

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

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

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

    return actual, pred

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

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

    return rmse, mape*100

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

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

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

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

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

    #store them in the train dictionary
    if verbose:
        print('adding train results in the dictionary..')
    train['rmse'] = train_rmse
    train['mape'] = train_mape
    train['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 [101]:

```
def run_xgboost_hyp(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
    algo.fit(x_train, y_train, eval_metric = 'rmse')
    # from the trained model, get the predictions....
    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)
    y_test_pred = algo.predict(x_test)
    rmse_test, mape_test = get_error_metrics(y_true=y_test.values, y_pred=y_test_pred)
    if verbose:
        #print('\nTEST DATA')

        print('RMSE : ', rmse_test)
        print('MAPE : ', mape_test)
        print('-'*30)

    # return these train and test results...
    return rmse_test,rmse_train
```

In [86]:

```
# prepare Train data
import warnings
warnings.filterwarnings('ignore')
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']
train_rmse=[]
test_rmse=[]
best_rmsc=100.1
estimator= [5, 10, 50, 100, 200]
max_depth=[2, 3, 4, 5, 6, 7]
for est in estimator:
    for dep in max_depth:
        print("Max-depth=",dep)
        print("Estimators=",est)
```

```
        print("Estimators=",est)
        first_xgb = xgb.XGBRegressor(silent=False, n_jobs=13, random_state=15, n_estimators=est,max
_depth=dep)
        rmse_test,rmse_train=run_xgboost_hyp(first_xgb, x_train, y_train, x_test, y_test)
        print("Max-depth=",dep)
        print("Estimators=",est)
        train_rmse.append(rmse_train)
        test_rmse.append(rmse_test)
        if rmse_test<best_rmsc:
            best_rmsc=rmse_test
            best_depth=dep
            best_est=est
print("Best RMSE score",best_rmsc)
print("Best paramater are")
print("Max-depth=",best_depth)
print("Estimators=",best_est)
```

```
Max-depth= 2
Estimators= 5
[12:08:16] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  2.1638187285720933
MAPE :  51.373834662223075
Max-depth= 2
Estimators= 5
Max-depth= 3
Estimators= 5
[12:08:22] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  2.181724813592308
MAPE :  51.85326036033134
Max-depth= 3
Estimators= 5
Max-depth= 4
Estimators= 5
[12:08:29] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  2.1681949652613772
MAPE :  51.49957382785737
Max-depth= 4
Estimators= 5
Max-depth= 5
Estimators= 5
[12:08:37] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  2.128388882447386
MAPE :  50.437828071702974
Max-depth= 5
Estimators= 5
Max-depth= 6
Estimators= 5
[12:08:46] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  2.144227197999755
MAPE :  50.85967294093863
Max-depth= 6
Estimators= 5
Max-depth= 7
Estimators= 5
[12:08:55] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  2.161595820787177
MAPE :  51.32171404562765
Max-depth= 7
Estimators= 5
```

```
Max-depth= 2
Estimators= 10
[12:09:09] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.5612059763007227
MAPE :  38.42165365284793
Max-depth= 2
Estimators= 10
Max-depth= 3
Estimators= 10
[12:09:18] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.5990506932965267
MAPE :  39.11867046747442
Max-depth= 3
Estimators= 10
Max-depth= 4
Estimators= 10
[12:09:29] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.5650355518103993
MAPE :  38.48241035933061
Max-depth= 4
Estimators= 10
Max-depth= 5
Estimators= 10
[12:09:44] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.5368756983786482
MAPE :  37.95156720475861
Max-depth= 5
Estimators= 10
Max-depth= 6
Estimators= 10
[12:09:58] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.5568490962656623
MAPE :  38.325998688796474
Max-depth= 6
Estimators= 10
Max-depth= 7
Estimators= 10
[12:10:11] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.559772099354371
MAPE :  38.3742627441203
Max-depth= 7
Estimators= 10
Max-depth= 2
Estimators= 50
[12:10:36] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.0753828457100834
MAPE :  34.5547343559214
Max-depth= 2
Estimators= 50
Max-depth= 3
Estimators= 50
[12:11:00] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.0776509700525043
```

```
MAPE :  34.363680425256625
Max-depth= 3
Estimators= 50
Max-depth= 4
Estimators= 50
[12:11:31] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.0762727210598162
MAPE :  34.474499675760356
Max-depth= 4
Estimators= 50
Max-depth= 5
Estimators= 50
[12:12:17] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.0782055864851594
MAPE :  34.32783980164519
Max-depth= 5
Estimators= 50
Max-depth= 6
Estimators= 50
[12:13:16] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.08979255518755
MAPE :  33.74380910942254
Max-depth= 6
Estimators= 50
Max-depth= 7
Estimators= 50
[12:14:21] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.1130776814963506
MAPE :  33.02934123719897
Max-depth= 7
Estimators= 50
Max-depth= 2
Estimators= 100
[12:15:43] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.075242167168039
MAPE :  34.590256198860686
Max-depth= 2
Estimators= 100
Max-depth= 3
Estimators= 100
[12:16:34] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.0769599573828592
MAPE :  34.431788329400995
Max-depth= 3
Estimators= 100
Max-depth= 4
Estimators= 100
[12:17:07] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.0752562377090324
MAPE :  34.593572731673525
Max-depth= 4
Estimators= 100
Max-depth= 5
Estimators= 100
[12:18:39] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
```

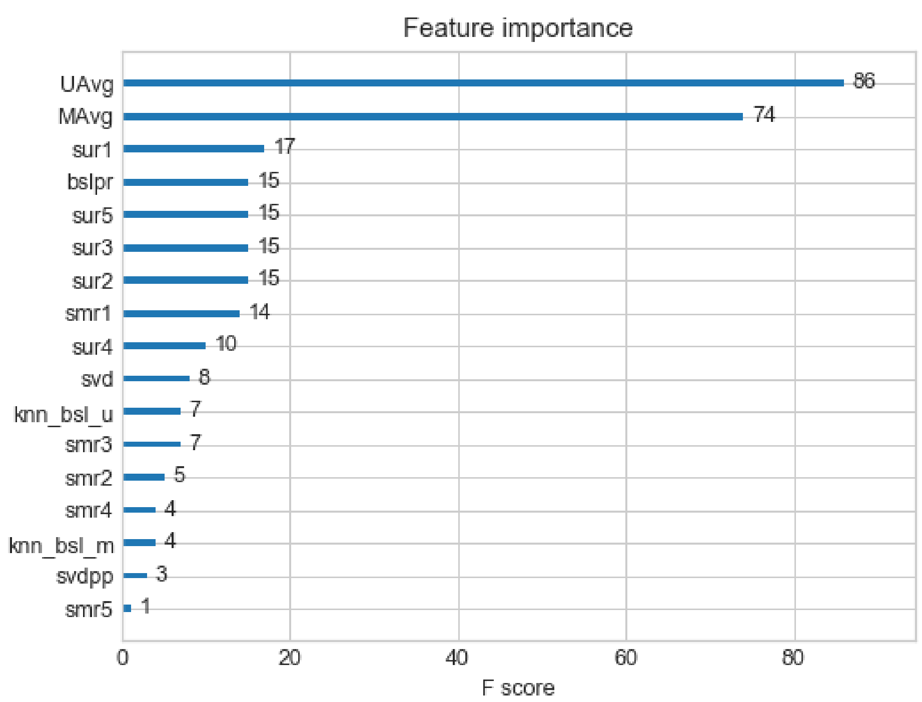
```
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.077182757447147
MAPE :  34.41750848790213
Max-depth= 5
Estimators= 100
Max-depth= 6
Estimators= 100
[12:20:26] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.0878153849422967
MAPE :  33.82799972046007
Max-depth= 6
Estimators= 100
Max-depth= 7
Estimators= 100
[12:22:41] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.1130752179547787
MAPE :  33.031530547257994
Max-depth= 7
Estimators= 100
Max-depth= 2
Estimators= 200
[12:24:00] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.0752680377794044
MAPE :  34.600010653713454
Max-depth= 2
Estimators= 200
Max-depth= 3
Estimators= 200
[12:25:23] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.076522074457759
MAPE :  34.47139586149072
Max-depth= 3
Estimators= 200
Max-depth= 4
Estimators= 200
[12:26:52] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.0752684168297386
MAPE :  34.59918056733446
Max-depth= 4
Estimators= 200
Max-depth= 5
Estimators= 200
[12:29:51] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.0768661719949078
MAPE :  34.44715886306988
Max-depth= 5
Estimators= 200
Max-depth= 6
Estimators= 200
[12:32:37] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
------------------------------
RMSE :  1.0879544076256689
MAPE :  33.82365133844834
Max-depth= 6
Estimators= 200
Max-depth= 7
```

```
Estimators= 200
[12:37:04] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
------------------------------
RMSE :  1.112606463815137
MAPE :  33.050734413510604
Max-depth= 7
Estimators= 200
Best RMSE score 1.075242167168039
Best paramater are
Max-depth= 2
Estimators= 100
```

In [92]:

```
# initialize Our first XGBoost model...
first_xgb = xgb.XGBRegressor(silent=False, n_jobs=13, random_state=15, n_estimators=100,max_depth=2
,learning_rate=0.5)
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..
[13:22:49] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Done. Time taken : 0:00:55.119774

Done

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

TEST DATA
------------------------------
RMSE :  1.0741508440105254
MAPE :  34.86368755778593
```



4.4.2 Suprise BaselineModel

In [34]:

```
from surprise import BaselineOnly
```

Predicted_rating : (baseline prediction)

```
-
http://surprise.readthedocs.io/en/stable/basic_algorithms.html#surprise.prediction_algorithm
seline_only.BaselineOnly
```

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

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

Optimization function (Least Squares Problem)

```
- http://surprise.readthedocs.io/en/stable/prediction_algorithms.html#baselines-estimates-c
onfiguration
```

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

In [35]:

```
# 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 : 0:00:03.028447

Evaluating the model with train data..
time taken : 0:00:03.247819
---------------
Train Data
---------------
RMSE : 0.9347153928678286

MAPE : 29.389572652358183

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.083013
---------------
Test Data
---------------
RMSE : 1.0730330260516174

MAPE : 35.04995544572911
```

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

---------------------------------------------
Total time taken to run this algorithm : 0:00:06.365282
```

4.4.3 XGBoost with initial 13 features + Surprise Baseline predictor

Updating Train Data

In [93]:

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

Out[93]:

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

2 rows × 21 columns

Updating Test Data

In [94]:

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

reg_test_df.head(2)
```

Out[94]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	...	smr4	smr5	
0	808635	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	...	3.581679	3.581679	3.5
1	941866	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	...	3.581679	3.581679	3.5

2 rows × 21 columns

In [102]:

```python
# prepare Train data
import warnings
warnings.filterwarnings('ignore')


# 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']
train_rmse=[]
test_rmse=[]
best_rmsc=100.1
estimator= [5, 10, 50, 100, 200]
max_depth=[2, 3, 4, 5, 6, 7]
for est in estimator:
    for dep in max_depth:
        print("Max-depth=",dep)
        print("Estimators=",est)
```

```
        xgb_bsl = xgb.XGBRegressor(silent=False, n_jobs=13, random_state=15, n_estimators=est,max_d
epth=dep)
        rmse_test,rmse_train=run_xgboost_hyp(xgb_bsl, x_train, y_train, x_test, y_test)
        train_rmse.append(rmse_train)
        test_rmse.append(rmse_test)
        if rmse_test<best_rmsc:
            best_rmsc=rmse_test
            best_depth=dep
            best_est=est
print("Best RMSE score",best_rmsc)
print("Best paramater are")
print("Max-depth=",best_depth)
print("Estimators=",best_est)
```

```
Max-depth= 2
Estimators= 5
[13:53:32] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  2.1638187285720933
MAPE :  51.373834662223075
------------------------------
Max-depth= 3
Estimators= 5
[13:53:38] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  2.181724813592308
MAPE :  51.85326036033134
------------------------------
Max-depth= 4
Estimators= 5
[13:53:44] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  2.1681949652613772
MAPE :  51.49957382785737
------------------------------
Max-depth= 5
Estimators= 5
[13:53:51] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  2.128388882447386
MAPE :  50.437828071702974
------------------------------
Max-depth= 6
Estimators= 5
[13:54:00] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  2.144227197999755
MAPE :  50.85967294093863
------------------------------
Max-depth= 7
Estimators= 5
[13:54:10] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  2.161595820787177
MAPE :  51.32171404562765
------------------------------
Max-depth= 2
Estimators= 10
[13:54:21] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.5612059763007227
MAPE :  38.42165365284793
------------------------------
Max-depth= 3
Estimators= 10
[13:54:29] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.5990506932965267
MAPE :  39.11867046747442
```

```
------------------------------
Max-depth= 4
Estimators= 10
[13:54:39] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.5650355518103993
MAPE :  38.48241035933061
------------------------------
Max-depth= 5
Estimators= 10
[13:54:53] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.5368756983786482
MAPE :  37.95156720475861
------------------------------
Max-depth= 6
Estimators= 10
[13:55:06] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.5568490962656623
MAPE :  38.325998688796474
------------------------------
Max-depth= 7
Estimators= 10
[13:55:30] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.559772099354371
MAPE :  38.3742627441203
------------------------------
Max-depth= 2
Estimators= 50
[13:55:49] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.0753828457100834
MAPE :  34.5547343559214
------------------------------
Max-depth= 3
Estimators= 50
[13:56:20] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.0776509700525043
MAPE :  34.363680425256625
------------------------------
Max-depth= 4
Estimators= 50
[13:57:05] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.0762727210598162
MAPE :  34.474499675760356
------------------------------
Max-depth= 5
Estimators= 50
[13:57:59] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.0782055864851594
MAPE :  34.32783980164519
------------------------------
Max-depth= 6
Estimators= 50
[13:59:16] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.08979255518755
MAPE :  33.74380910942254
------------------------------
Max-depth= 7
Estimators= 50
[14:00:39] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
```

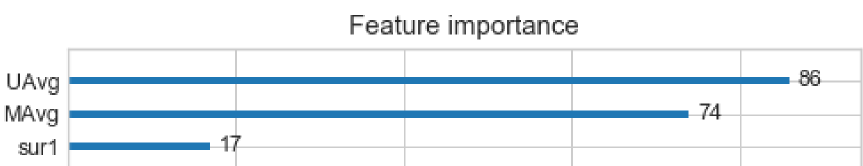
```
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.1130776814963506
MAPE :  33.02934123719897
------------------------------
Max-depth= 2
Estimators= 100
[14:02:12] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.075242167168039
MAPE :  34.590256198860686
------------------------------
Max-depth= 3
Estimators= 100
[14:03:11] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0769599573828592
MAPE :  34.431788329400995
------------------------------
Max-depth= 4
Estimators= 100
[14:04:31] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0752562377090324
MAPE :  34.593572731673525
------------------------------
Max-depth= 5
Estimators= 100
[14:06:17] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.077182757447147
MAPE :  34.41750848790213
------------------------------
Max-depth= 6
Estimators= 100
[14:08:16] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0878153849422967
MAPE :  33.82799972046007
------------------------------
Max-depth= 7
Estimators= 100
[14:10:26] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.1130752179547787
MAPE :  33.031530547257994
------------------------------
Max-depth= 2
Estimators= 200
[14:13:21] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0752680377794044
MAPE :  34.600010653713454
------------------------------
Max-depth= 3
Estimators= 200
[14:14:58] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.076522074457759
MAPE :  34.47139586149072
------------------------------
Max-depth= 4
Estimators= 200
[14:17:00] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0752684168297386
MAPE :  34.59918056733446
------------------------------
Max-depth= 5
```

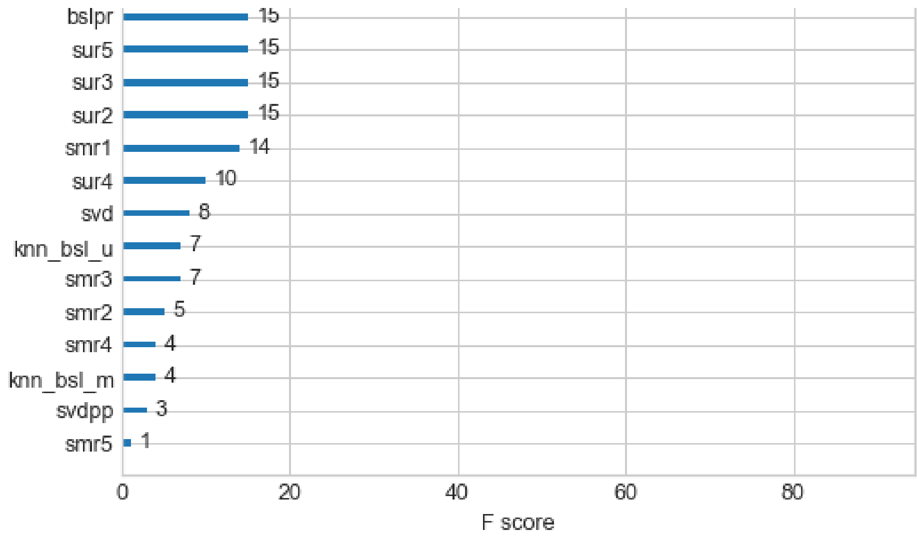
```
Max-depth= 5
Estimators= 200
[14:19:56] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.0768661719949078
MAPE :  34.44715886306988
------------------------------
Max-depth= 6
Estimators= 200
[14:23:28] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.0879544076256689
MAPE :  33.82365133844834
------------------------------
Max-depth= 7
Estimators= 200
[14:27:02] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.112606463815137
MAPE :  33.050734413510604
------------------------------
Best RMSE score 1.075242167168039
Best paramater are
Max-depth= 2
Estimators= 100
```

In [108]:

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

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

# initialize Our first XGBoost model...
xgb_bsl = xgb.XGBRegressor(silent=False, n_jobs=13, random_state=15, n_estimators=100,max_depth=2,learning_rate=0.5)
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..
[14:39:17] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Done. Time taken : 0:00:59.768178

Done

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

TEST DATA
------------------------------
RMSE :  1.0741508440105254
MAPE :  34.86368755778593
```





4.4.4 Surprise KNNBaseline predictor

In [39]:

```
from surprise import KNNBaseline
```

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

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

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

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

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

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

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

4.4.4.1 Surprise KNNBaseline with user user similarities

In [40]:

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

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

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

# Just store these error metrics in our models_evaluation datastructure
models_evaluation_train['knn_bsl_u'] = knn_bsl_u_train_results
models_evaluation_test['knn_bsl_u'] = knn_bsl_u_test_results
```

```
Training the model...
Estimating biases using sgd...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Done. time taken : 0:02:46.128117

Evaluating the model with train data..
time taken : 0:06:57.173944
---------------
Train Data
---------------
RMSE : 0.33642097416508826

MAPE : 9.145093375416348

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.234030
---------------
Test Data
---------------
RMSE : 1.0726493739667242

MAPE : 35.02094499698424

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:09:43.542093
```

4.4.4.2 Surprise KNNBaseline with movie movie similarities

In [41]:

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

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

sim_options = {'user_based' : False,
               'name': 'pearson_baseline',
               'shrinkage': 100,
               'min_support': 2
              }
# we keep other parameters like regularization parameter and learning_rate as default values.
bsl_options = {'method': 'sgd'}


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

knn_bsl_m_train_results, knn_bsl_m_test_results = run_surprise(knn_bsl_m, trainset, testset,
verbose=True)

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

```
Training the model...
Estimating biases using sgd...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Done. time taken : 0:00:05.121294
```

```
Evaluating the model with train data..
time taken : 0:00:36.266434
---------------
Train Data
---------------
RMSE : 0.32584796251610554

MAPE : 8.447062581998374

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:01.189538
---------------
Test Data
---------------
RMSE : 1.0727588326536 83

MAPE : 35.02269653015042

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:00:42.580266
```

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 [109]:

```
# 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[109]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	...	smr4	smr5	UAvg	MAvg	rating	bslpr	kn
0	53406	33	3.581679	4.0	5.0	5.0	4.0	1.0	5.0	2.0	...	3.0	1.0	3.370370	4.092437	4	3.898982	3.9
1	99540	33	3.581679	5.0	5.0	5.0	4.0	5.0	3.0	4.0	...	3.0	5.0	3.555556	4.092437	3	3.371403	3.1

2 rows × 21 columns

Preparing Test data

In [110]:

```
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[110]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	...	smr4	smr5	
0	808635	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	...	3.581679	3.581679	3.5

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	...	smr4	smr5	
1	941866	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	...	3.581679	3.581679	3.5

2 rows × 21 columns

In [111]:

```
# 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']
train_rmse=[]
test_rmse=[]
best_rmsc=100.1
estimator= [5, 10, 50, 100, 200]
max_depth=[2, 3, 4, 5, 6, 7]
for est in estimator:
    for dep in max_depth:
        print("Max-depth=",dep)
        print("Estimators=",est)
        xgb_knn_bsl = xgb.XGBRegressor(silent=False, n_jobs=13, random_state=15, n_estimators=est,m
ax_depth=dep)
        rmse_test,rmse_train=run_xgboost_hyp(xgb_knn_bsl, x_train, y_train, x_test, y_test)
        train_rmse.append(rmse_train)
        test_rmse.append(rmse_test)
        if rmse_test<best_rmsc:
            best_rmsc=rmse_test
            best_depth=dep
            best_est=est
print("Best RMSE score",best_rmsc)
print("Best paramater are")
print("Max-depth=",best_depth)
print("Estimators=",best_est)
```

```
Max-depth= 2
Estimators= 5
[14:40:18] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  2.1638187285720933
MAPE :  51.373834662223075
------------------------------
Max-depth= 3
Estimators= 5
[14:40:25] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  2.181724813592308
MAPE :  51.85326036033134
------------------------------
Max-depth= 4
Estimators= 5
[14:40:32] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  2.1681949652613772
MAPE :  51.49957382785737
------------------------------
Max-depth= 5
Estimators= 5
[14:40:39] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  2.128388882447386
MAPE :  50.437828071702974
------------------------------
Max-depth= 6
Estimators= 5
[14:40:46] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  2.144227197999755
MAPE :  50.85967294093863
```

```
MAPE :  50.65907294095805
------------------------------
Max-depth= 7
Estimators= 5
[14:40:54] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  2.161595820787177
MAPE :  51.32171404562765
------------------------------
Max-depth= 2
Estimators= 10
[14:41:04] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.5612059763007227
MAPE :  38.42165365284793
------------------------------
Max-depth= 3
Estimators= 10
[14:41:11] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.5990506932965267
MAPE :  39.11867046747442
------------------------------
Max-depth= 4
Estimators= 10
[14:41:20] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.5650355518103993
MAPE :  38.48241035933061
------------------------------
Max-depth= 5
Estimators= 10
[14:41:31] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.5368756983786482
MAPE :  37.95156720475861
------------------------------
Max-depth= 6
Estimators= 10
[14:41:42] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.5568490962656623
MAPE :  38.325998688796474
------------------------------
Max-depth= 7
Estimators= 10
[14:41:54] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.559772099354371
MAPE :  38.3742627441203
------------------------------
Max-depth= 2
Estimators= 50
[14:42:07] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0753828457100834
MAPE :  34.5547343559214
------------------------------
Max-depth= 3
Estimators= 50
[14:42:31] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0776509700525043
MAPE :  34.363680425256625
------------------------------
Max-depth= 4
Estimators= 50
[14:43:05] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
```

```
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0762727210598162
MAPE :  34.474499675760356
------------------------------
Max-depth= 5
Estimators= 50
[14:43:54] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0782055864851594
MAPE :  34.32783980164519
------------------------------
Max-depth= 6
Estimators= 50
[14:44:54] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.08979255518755
MAPE :  33.74380910942254
------------------------------
Max-depth= 7
Estimators= 50
[14:45:48] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.1130776814963506
MAPE :  33.02934123719897
------------------------------
Max-depth= 2
Estimators= 100
[14:47:12] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.075242167168039
MAPE :  34.590256198860686
------------------------------
Max-depth= 3
Estimators= 100
[14:47:45] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0769599573828592
MAPE :  34.431788329400995
------------------------------
Max-depth= 4
Estimators= 100
[14:48:54] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0752562377090324
MAPE :  34.593572731673525
------------------------------
Max-depth= 5
Estimators= 100
[14:50:17] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.077182757447147
MAPE :  34.41750848790213
------------------------------
Max-depth= 6
Estimators= 100
[14:52:03] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0878153849422967
MAPE :  33.82799972046007
------------------------------
Max-depth= 7
Estimators= 100
[14:54:35] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.1130752179547787
MAPE :  33.031530547257994
------------------------------
Max-depth= 2
```

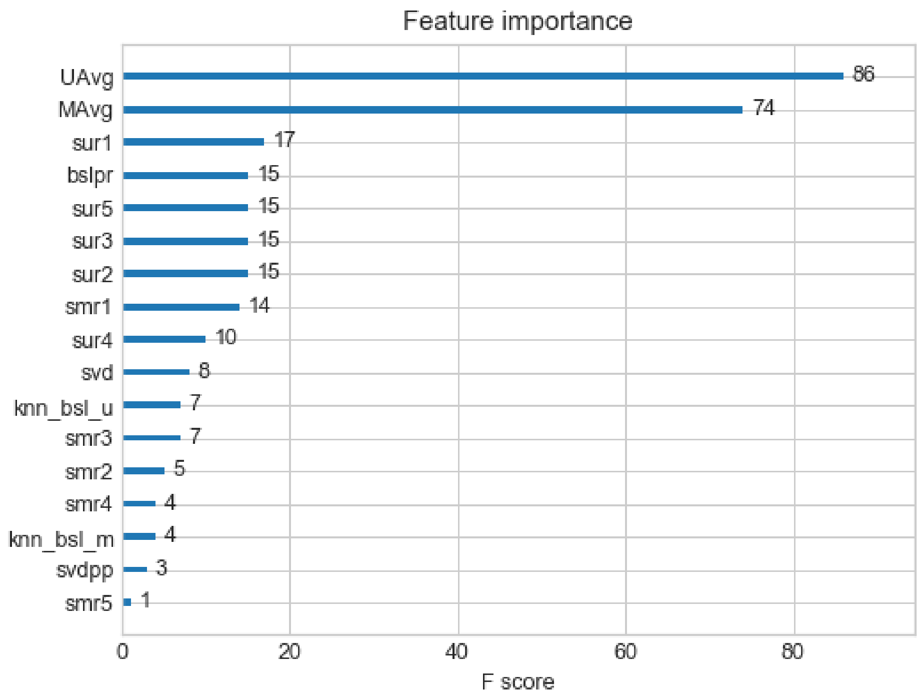
```
Max-depth= 2
Estimators= 200
[14:57:05] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.0752680377794044
MAPE :  34.600010653713454
------------------------------
Max-depth= 3
Estimators= 200
[14:58:46] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.076522074457759
MAPE :  34.47139586149072
------------------------------
Max-depth= 4
Estimators= 200
[15:00:49] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.0752684168297386
MAPE :  34.59918056733446
------------------------------
Max-depth= 5
Estimators= 200
[15:03:53] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.0768661719949078
MAPE :  34.44715886306988
------------------------------
Max-depth= 6
Estimators= 200
[15:07:42] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.0879544076256689
MAPE :  33.82365133844834
------------------------------
Max-depth= 7
Estimators= 200
[15:12:28] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.112606463815137
MAPE :  33.050734413510604
------------------------------
Best RMSE score 1.075242167168039
Best paramater are
Max-depth= 2
Estimators= 100
```

In [115]:

```
# declare the model
xgb_knn_bsl = xgb.XGBRegressor(n_jobs=10, random_state=15,max_depth=2,n_estimators=100,learning_rate=0.5)
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..
[15:23:37] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Done. Time taken : 0:00:56.329301

Done
```

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

TEST DATA
------------------------------
RMSE :  1.0741508440105254
MAPE :  34.86368755778593
```



4.4.6 Matrix Factorization Techniques

4.4.6.1 SVD Matrix Factorization User Movie intractions

In [116]:

```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 [46]:

```
# 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 : 0:00:29.156029

Evaluating the model with train data..
time taken : 0:00:03.989375
---------------
Train Data
---------------
RMSE : 0.6574721240954099

MAPE : 19.704901088660478

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.480061
---------------
Test Data
---------------
RMSE : 1.0726046873826458

MAPE : 35.01953535988152

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:00:33.628462
```

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

In [47]:

```
from surprise import SVDpp
```

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

- Predicted Rating :

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

```
|I_u|^{-\frac{1}{2}} \sum_{j \in I_u}y_j\right) $
```

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

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

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

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

In [48]:

```
# 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 : 0:08:21.098684

Evaluating the model with train data..
time taken : 0:00:23.131172
---------------
Train Data
---------------
RMSE : 0.6032438403305899

MAPE : 17.49285063490268

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.405583
---------------
Test Data
---------------
RMSE : 1.0728491944183447

MAPE : 35.03817913919887

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:08:44.652050
```

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

Preparing Train data

In [117]:

```
# 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[117]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	...	smr4	smr5	UAvg	MAvg	rating	bslpr	kn
0	53406	33	3.581679	4.0	5.0	5.0	4.0	1.0	5.0	2.0	...	3.0	1.0	3.370370	4.092437	4	3.898982	3.9
1	99540	33	3.581679	5.0	5.0	5.0	4.0	5.0	3.0	4.0	...	3.0	5.0	3.555556	4.092437	3	3.371403	3.1

2 rows × 21 columns

Preparing Test data

In [118]:

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

reg_test_df.head(2)
```

Out[118]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	...	smr4	smr5	
0	808635	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	...	3.581679	3.581679	3.5
1	941866	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	...	3.581679	3.581679	3.5

2 rows × 21 columns

In [119]:

```
# 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']

train_rmse=[]
test_rmse=[]
best_rmsc=100.1
estimator= [5, 10, 50, 100, 200]
max_depth=[2, 3, 4, 5, 6, 7]
for est in estimator:
    for dep in max_depth:
        print("Max-depth=",dep)
        print("Estimators=",est)
        xgb_knn_bsl = xgb.XGBRegressor(silent=False, n_jobs=13, random_state=15, n_estimators=est,m
ax_depth=dep)
        rmse_test,rmse_train=run_xgboost_hyp(xgb_knn_bsl, x_train, y_train, x_test, y_test)
        train_rmse.append(rmse_train)
        test_rmse.append(rmse_test)
        if rmse_test<best_rmsc:
            best_rmsc=rmse_test
            best_depth=dep
```

```
            best_depth=dep
            best_est=est
print("Best RMSE score",best_rmsc)
print("Best paramater are")
print("Max-depth=",best_depth)
print("Estimators=",best_est)
```

```
Max-depth= 2
Estimators= 5
[15:25:03] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  2.1638187285720933
MAPE :  51.373834662223075
------------------------------
Max-depth= 3
Estimators= 5
[15:25:08] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  2.181724813592308
MAPE :  51.85326036033134
------------------------------
Max-depth= 4
Estimators= 5
[15:25:15] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  2.1681949652613772
MAPE :  51.49957382785737
------------------------------
Max-depth= 5
Estimators= 5
[15:25:21] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  2.128388882447386
MAPE :  50.437828071702974
------------------------------
Max-depth= 6
Estimators= 5
[15:25:32] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  2.144227197999755
MAPE :  50.85967294093863
------------------------------
Max-depth= 7
Estimators= 5
[15:25:40] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  2.161595820787177
MAPE :  51.32171404562765
------------------------------
Max-depth= 2
Estimators= 10
[15:25:50] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.5612059763007227
MAPE :  38.42165365284793
------------------------------
Max-depth= 3
Estimators= 10
[15:26:02] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.5990506932965267
MAPE :  39.11867046747442
------------------------------
Max-depth= 4
Estimators= 10
[15:26:14] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.5650355518103993
```

```
MAPE :  38.48241035933061
------------------------------
Max-depth= 5
Estimators= 10
[15:26:29] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.5368756983786482
MAPE :  37.95156720475861
------------------------------
Max-depth= 6
Estimators= 10
[15:26:43] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.5568490962656623
MAPE :  38.325998688796474
------------------------------
Max-depth= 7
Estimators= 10
[15:26:55] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.559772099354371
MAPE :  38.3742627441203
------------------------------
Max-depth= 2
Estimators= 50
[15:27:21] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0753828457100834
MAPE :  34.5547343559214
------------------------------
Max-depth= 3
Estimators= 50
[15:27:53] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0776509700525043
MAPE :  34.363680425256625
------------------------------
Max-depth= 4
Estimators= 50
[15:28:44] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0762727210598162
MAPE :  34.474499675760356
------------------------------
Max-depth= 5
Estimators= 50
[15:29:31] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0782055864851594
MAPE :  34.32783980164519
------------------------------
Max-depth= 6
Estimators= 50
[15:30:17] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.08979255518755
MAPE :  33.74380910942254
------------------------------
Max-depth= 7
Estimators= 50
[15:31:21] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.1130776814963506
MAPE :  33.02934123719897
------------------------------
Max-depth= 2
Estimators= 100
[15:32:40] WARNING: C:/Jenkins/workspace/xgboost-
```

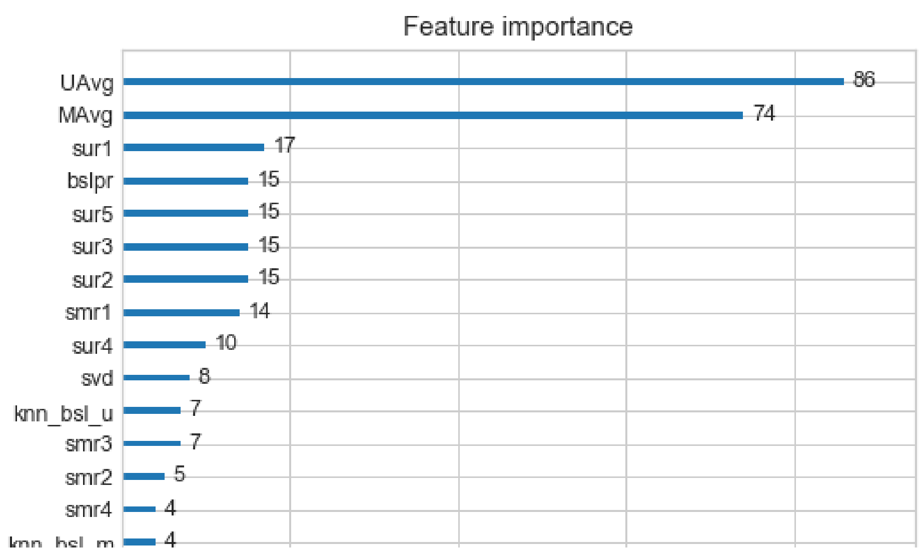
```
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.075242167168039
MAPE :  34.590256198860686
------------------------------
Max-depth= 3
Estimators= 100
[15:33:32] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0769599573828592
MAPE :  34.431788329400995
------------------------------
Max-depth= 4
Estimators= 100
[15:34:48] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0752562377090324
MAPE :  34.593572731673525
------------------------------
Max-depth= 5
Estimators= 100
[15:36:10] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.077182757447147
MAPE :  34.41750848790213
------------------------------
Max-depth= 6
Estimators= 100
[15:37:49] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0878153849422967
MAPE :  33.82799972046007
------------------------------
Max-depth= 7
Estimators= 100
[15:40:16] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.1130752179547787
MAPE :  33.031530547257994
------------------------------
Max-depth= 2
Estimators= 200
[15:42:36] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0752680377794044
MAPE :  34.600010653713454
------------------------------
Max-depth= 3
Estimators= 200
[15:44:30] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.076522074457759
MAPE :  34.47139586149072
------------------------------
Max-depth= 4
Estimators= 200
[15:46:51] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0752684168297386
MAPE :  34.59918056733446
------------------------------
Max-depth= 5
Estimators= 200
[15:49:28] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0768661719949078
MAPE :  34.44715886306988
------------------------------
```

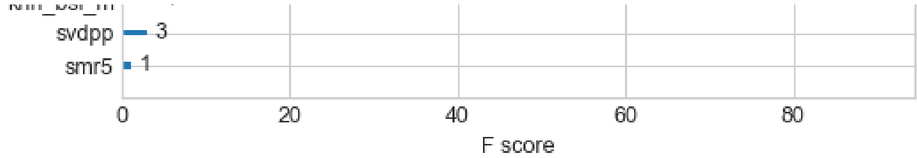
```
Max-depth= 6
Estimators= 200
[15:52:52] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.0879544076256689
MAPE :  33.82365133844834
------------------------------
Max-depth= 7
Estimators= 200
[15:57:48] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.112606463815137
MAPE :  33.050734413510604
------------------------------
Best RMSE score 1.075242167168039
Best paramater are
Max-depth= 2
Estimators= 100
```

In [120]:

```python
xgb_final = xgb.XGBRegressor(n_jobs=10, random_state=15,max_depth=2,n_estimators=100,learning_rate=0.5)
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..
[16:07:32] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Done. Time taken : 0:01:06.230940

Done

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

TEST DATA
------------------------------
RMSE :  1.0741508440105254
MAPE :  34.86368755778593
```





4.4.8 XgBoost with Surprise Baseline + Surprise KNNbaseline + MF Techniques

In [121]:

```
# 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']

train_rmse=[]
test_rmse=[]
best_rmsc=100.1
estimator= [5, 10, 50, 100, 200]
max_depth=[2, 3, 4, 5, 6, 7]
for est in estimator:
    for dep in max_depth:
        print("Max-depth=",dep)
        print("Estimators=",est)
        xgb_knn_bsl = xgb.XGBRegressor(silent=False, n_jobs=13, random_state=15, n_estimators=est,m
ax_depth=dep)
        rmse_test,rmse_train=run_xgboost_hyp(xgb_knn_bsl, x_train, y_train, x_test, y_test)
        train_rmse.append(rmse_train)
        test_rmse.append(rmse_test)
        if rmse_test<best_rmsc:
            best_rmsc=rmse_test
            best_depth=dep
            best_est=est
print("Best RMSE score",best_rmsc)
print("Best paramater are")
print("Max-depth=",best_depth)
print("Estimators=",best_est)
```

```
Max-depth= 2
Estimators= 5
[16:39:09] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  2.1150294770851157
MAPE :  50.06152468944951
------------------------------
Max-depth= 3
Estimators= 5
[16:39:14] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  2.1152882823571666
MAPE :  50.068480694227326
------------------------------
Max-depth= 4
Estimators= 5
[16:39:19] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  2.1153628153937576
MAPE :  50.07048389116056
------------------------------
Max-depth= 5
Estimators= 5
[16:39:22] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  2.115248039004232
MAPE :  50.06739907825235
------------------------------
```

```
Max-depth= 6
Estimators= 5
[16:39:28] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  2.1153492638645193
MAPE :  50.07011967353634
------------------------------
Max-depth= 7
Estimators= 5
[16:39:34] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  2.1153297579285533
MAPE :  50.06959542089541
------------------------------
Max-depth= 2
Estimators= 10
[16:39:41] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.521475233309654
MAPE :  37.69053440212342
------------------------------
Max-depth= 3
Estimators= 10
[16:39:47] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.5234402761498025
MAPE :  37.728318866098284
------------------------------
Max-depth= 4
Estimators= 10
[16:39:54] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.524166732347037
MAPE :  37.742253360469114
------------------------------
Max-depth= 5
Estimators= 10
[16:40:02] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.523491246805091
MAPE :  37.72936628598442
------------------------------
Max-depth= 6
Estimators= 10
[16:40:14] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.5238033265339779
MAPE :  37.7357581670872
------------------------------
Max-depth= 7
Estimators= 10
[16:40:24] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.5238382068360006
MAPE :  37.735917053957884
------------------------------
Max-depth= 2
Estimators= 50
[16:40:34] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.0755728208337845
MAPE :  34.97665716640581
------------------------------
Max-depth= 3
Estimators= 50
[16:40:52] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

```
reg:squarederror.
RMSE :  1.0755443897903791
MAPE :  34.98036778885168
------------------------------
Max-depth= 4
Estimators= 50
[16:41:15] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0755499389847234
MAPE :  34.97580036623238
------------------------------
Max-depth= 5
Estimators= 50
[16:41:28] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0755861283239119
MAPE :  34.96947678220646
------------------------------
Max-depth= 6
Estimators= 50
[16:42:12] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.075550615728022
MAPE :  34.97072894078254
------------------------------
Max-depth= 7
Estimators= 50
[16:42:52] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0756145152388188
MAPE :  34.9669154752758
------------------------------
Max-depth= 2
Estimators= 100
[16:43:46] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0754669841505962
MAPE :  35.02122605934518
------------------------------
Max-depth= 3
Estimators= 100
[16:44:18] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0753047860953797
MAPE :  35.07058962951319
------------------------------
Max-depth= 4
Estimators= 100
[16:45:00] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.075252473462124
MAPE :  35.088326401198756
------------------------------
Max-depth= 5
Estimators= 100
[16:45:16] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.0755450330014913
MAPE :  35.00226507848423
------------------------------
Max-depth= 6
Estimators= 100
[16:46:28] WARNING: C:/Jenkins/workspace/xgboost-
win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
reg:squarederror.
RMSE :  1.075477520622328
MAPE :  35.00818520872434
------------------------------
Max-depth= 7
Estimators= 100
```

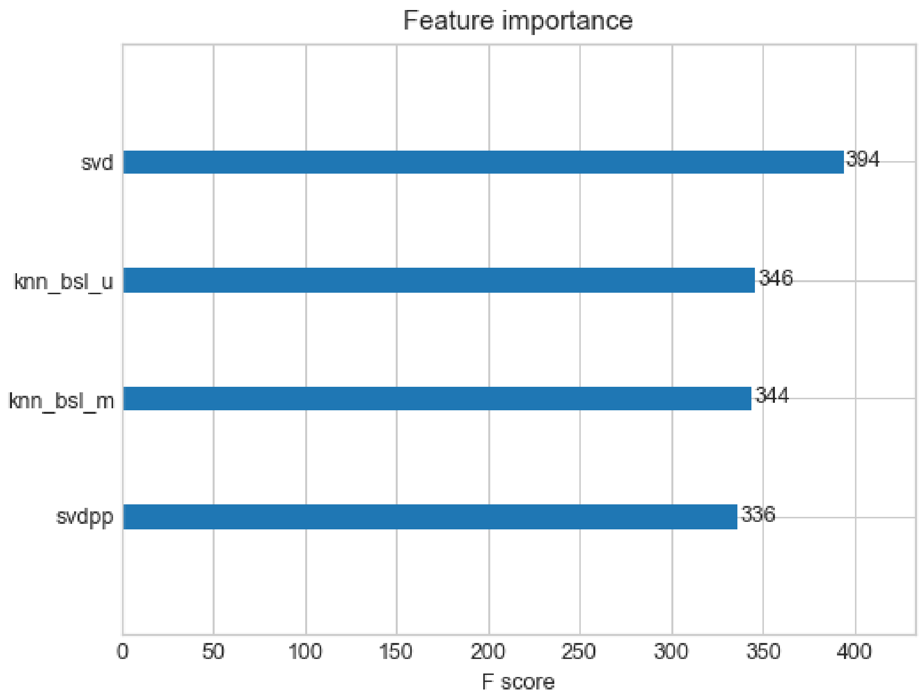
```
Estimators= 100
[16:48:08] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.0757255593502222
MAPE :  34.972636566860494
------------------------------
Max-depth= 2
Estimators= 200
[16:50:11] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.0754833662287098
MAPE :  35.009167429165245
------------------------------
Max-depth= 3
Estimators= 200
[16:51:24] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.0754678605906915
MAPE :  35.01501425915812
------------------------------
Max-depth= 4
Estimators= 200
[16:52:56] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.075554795275701
MAPE :  34.9956304806786
------------------------------
Max-depth= 5
Estimators= 200
[16:54:25] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.0755698752062204
MAPE :  34.9937935881552
------------------------------
Max-depth= 6
Estimators= 200
[16:56:27] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.0756043015593508
MAPE :  34.97133283231675
------------------------------
Max-depth= 7
Estimators= 200
[16:59:50] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
RMSE :  1.0769253793884428
MAPE :  34.78949107182634
------------------------------
Best RMSE score 1.075252473462124
Best paramater are
Max-depth= 4
Estimators= 100
```

In [123]:

```
xgb_all_models = xgb.XGBRegressor(n_jobs=10, random_state=15,max_depth=4,n_estimators=100,learning_rate=0.5)
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..
[17:03:48] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of
```

```
reg:squarederror.
Done. Time taken : 0:01:05.869269

Done

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

TEST DATA
------------------------------
RMSE :  1.0754530846209494
MAPE :  35.03146527956777
```



4.5 Comparision between all models

Before hyperparameter tunning

In [53]:

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

Out[53]:

```
svd                1.0726046873826458
knn_bsl_u          1.0726493739667242
knn_bsl_m           1.072758832653683
svdpp              1.0728491944183447
bsl_algo           1.0730330260516174
xgb_knn_bsl_mu     1.0753229281412784
xgb_all_models      1.075480663561971
first_algo         1.0761851474385373
xgb_bsl            1.0763419061709816
xgb_final          1.0763580984894978
xgb_knn_bsl        1.0763602465199797
Name: rmse, dtype: object
```

After hyperparameter tunning

In [127]:

```
# Saving our TEST_RESULTS into a dataframe so that you don't have to run it again
pd.DataFrame(models_evaluation_test).to_csv('ssmall_sample_results2.csv')
models = pd.read_csv('ssmall_sample_results2.csv', index_col=0)
models.loc['rmse'].sort_values()
```

Out[127]:

```
svd              1.0726046873826458
knn_bsl_u        1.0726493739667242
knn_bsl_m         1.072758832653683
svdpp            1.0728491944183447
bsl_algo         1.0730330260516174
first_algo       1.0741508440105254
xgb_bsl          1.0741508440105254
xgb_knn_bsl      1.0741508440105254
xgb_final        1.0741508440105254
xgb_all_models   1.0754530846209494
Name: rmse, dtype: object
```

In [ ]:

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

5. Assignment

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

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

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

In [187]:

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

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

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

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

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

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

     level = parseInt(openLevel);


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

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

 });


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


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

};

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

// Rebuild to TOC every minute
setInterval(function(){createTOC();},60000);
```