1. Business Problem

1.1 Problem Description

Netflix is all about connecting people to the movies they love. To help customers find those movies, they developed world-class movie recommendation system: CinematchSM. Its job is to predict whether someone will enjoy a movie based on how much they liked or disliked other movies. Netflix use those predictions to make personal movie recommendations based on each customer's unique tastes. And while **Cinematch** is doing pretty well, it can always be made better.

Now there are a lot of interesting alternative approaches to how Cinematch works that netflix haven't tried. Some are described in the literature, some aren't. We're curious whether any of these can beat Cinematch by making better predictions. Because, frankly, if there is a much better approach it could make a big difference to our customers and our business.

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

1.2 Problem Statement

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

1.3 Sources

- https://www.netflixprize.com/rules.html
- https://www.kaggle.com/netflix-inc/netflix-prize-data
- Netflix blog: https://medium.com/netflix-techblog/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429 (very nice blog)
- surprise library: http://surpriselib.com/ (we use many models from this library)
- surprise library doc: http://surprise.readthedocs.io/en/stable/getting_started.html (we use many models from this library)
- installing surprise: https://github.com/NicolasHug/Surprise#installation
- Research paper: http://courses.ischool.berkeley.edu/i290-dm/s11/SECURE/a1-koren.pdf (most of our work was inspired by this paper)
- SVD Decomposition : https://www.youtube.com/watch?v=P5mlg91as1c

1.4 Real world/Business Objectives and constraints

Objectives:

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

Constraints:

1. Some form of interpretability.

2. Machine Learning Problem

2.1 Data

2.1.1 Data Overview

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

Data files :

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

```
The first line of each file [combined_data_1.txt, combined_data_2.txt, combined_data_3.txt, combined_data_4.txt] contains the movie id foll
owed by a colon. Each subsequent line in the file corresponds to a rating from a customer and its date in the following format:

CustomerID,Rating,Date

MovieIDs range from 1 to 17770 sequentially.
CustomerIDs range from 1 to 2649429, with gaps. There are 480189 users.
Ratings are on a five star (integral) scale from 1 to 5.
Dates have the format YYYY-MM-DD.
```

2.1.2 Example Data point

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

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

2.2 Mapping the real world problem to a Machine Learning Problem

2.2.1 Type of Machine Learning Problem

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

2.2.2 Performance metric

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

2.2.3 Machine Learning Objective and Constraints

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

In [1]:

```
# this is just to know how much time will it take to run this entire ipython notebook
from datetime import datetime
# globalstart = datetime.now()
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({'figure.max_open_warning': 0})

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

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

In [0]:

```
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
```

In [3]:

```
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
WARNING:tensorflow:
The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-sunset.md
  * https://github.com/tensorflow/addons
  * https://github.com/tensorflow/io (for I/O related ops)
If you depend on functionality not listed there, please file an issue.
```

In [0]:

```
downloaded = drive.CreateFile({'id':'1ILIaNy-Pi-O0cnhvfoR05HgzTqFPRKAu'}) # replace the id with id of file you want to access
downloaded.GetContentFile('data.csv')
downloaded = drive.CreateFile({'id':'1NIm9r0SOpiGOy9LwE9Kk6p5YyS7V11vy'}) # replace the id with id of file you want to access
downloaded.GetContentFile('combined_data_1.txt')
```

In [0]:

```
downloaded = drive.CreateFile({'id':'1QR0XHvcCu3tnNWtAkXde2Gij2ZrdVvE4'}) # replace the id with id of file you want to access
downloaded.GetContentFile('combined_data_2.txt')
downloaded = drive.CreateFile({'id':'1tFK6vE_0n5wteDsmiqkZnV1h7p2-eE_y'}) # replace the id with id of file you want to access
downloaded.GetContentFile('combined_data_3.txt')
downloaded = drive.CreateFile({'id':'1Pr8R79AKuNhN48Q3wByo-EZcmEmlNhL-'}) # replace the id with id of file you want to access
downloaded.GetContentFile('combined_data_4.txt')
```

3. Exploratory Data Analysis

3.1 Preprocessing

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

In [2]:

```
start = datetime.now()
if not os.path.isfile('data.csv'):
    # Create a file 'data.csv' before reading it
    # Read all the files in netflix and store them in one big file('data.csv')
    # We 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)
```

```
Time taken : 0:00:00.000752
```

In [3]:

```
df = pd.read_csv('data.csv', names= ['movie', 'user', 'rating', 'date'])
df['date']= pd.to_datetime(df['date'])
```

```
print('Sorting dataframe by Date:...')
df.sort_values('date', inplace= True)
print('Done sorting.')
```

```
Sorting dataframe by Date:...
Done sorting.
```

In [4]:

```
df.head()
```

Out[4]:

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

In [5]:

```
df.shape
```

Out[5]:

```
(100480507, 4)
```

In [6]:

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

Out[6]:

```
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 [7]:

```
# https://chartio.com/resources/tutorials/how-to-check-if-any-value-is-nan-in-a-pandas-dataframe/
df.isnull().any()
```

Out[7]:

```
movie     False
user      False
rating    False
date      False
dtype: bool
```

3.1.3 Removing Duplicates

In [8]:

```
df.duplicated().any()
```

Out[8]:

```
False
```

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

In [9]:

```
print('The total number of ratings: ', df['rating'].count())
print('Total number of users: ', len(df['user'].unique()))
print('Total number of movies: ', len(df['movie'].unique()))
```

The total number of ratings: 100480507
Total number of users: 480189
Total number of movies: 17770

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

In [0]:

```
downloaded = drive.CreateFile({'id':'1vr1-A8GJ24CP0us553LrQ4pkwZdTwIzn'}) # replace the id with id of file you want to access
downloaded.GetContentFile('train.csv')
downloaded = drive.CreateFile({'id':'1IaVRHSLBiO4OLSWZp-oILE4SFFN45ubC'}) # replace the id with id of file you want to access
downloaded.GetContentFile('test.csv')
```

In [10]:

```
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, Test Data (#Ratings, #Users, and #Movies)

In [16]:

```
print('Train data:-')
print('The total number of ratings: ', train_df['rating'].count())
print('Total number of users: ', len(train_df['user'].unique()))
print('Total number of movies: ', len(train_df['movie'].unique()))
print('*'*50)
print('Test data:-')
print('The total number of ratings: ', test_df['rating'].count())
print('Total number of users: ', len(test_df['user'].unique()))
print('Total number of movies: ', len(test_df['movie'].unique()))
```

```
Train data:-
The total number of ratings:  80384405
Total number of users:  405041
Total number of movies:  17424
**************************************************
Test data:-
The total number of ratings:  20096102
Total number of users:  349312
Total number of movies:  17757
```

3.3 Exploratory Data Analysis on Train data

In [11]:

```
# 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 [12]:

```
sns.countplot(data= train_df, x='rating')
plt.title('Train data Ratings')
plt.ylabel('Ratings in Millions')
sns.set_style('whitegrid')
plt.legend()
plt.show()
```

No handles with labels found to put in legend.

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

In [13]:

```
# It is used to skip the warning "SettingWithCopyWarning"..
pd.options.mode.chained_assignment = None  # default='warn'

# return the name of the weekday series command
train_df['weekday'] = train_df['date'].dt.weekday_name
train_df.head()
```

Out[13]:

	movie	user	rating	date	weekday
0	10341	510180	4	1999-11-11	Thursday
1	1798	510180	5	1999-11-11	Thursday
2	10774	510180	3	1999-11-11	Thursday
3	8651	510180	2	1999-11-11	Thursday
4	14660	510180	2	1999-11-11	Thursday

3.3.2 Number of Ratings per a month

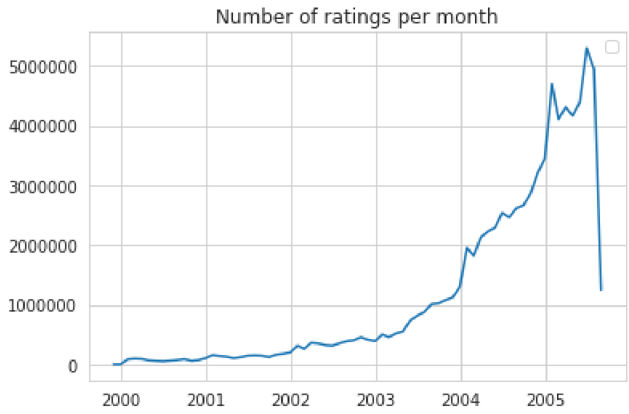
In [103]:

```
# https://www.geeksforgeeks.org/python-pandas-dataframe-resample/
rpm = train_df.resample(rule= 'm', on='date')['rating'].count()
plt.plot(rpm)
plt.title('Number of ratings per month')
plt.legend()
plt.show()
```

/usr/local/lib/python3.6/dist-packages/pandas/plotting/_matplotlib/converter.py:103: FutureWarning: Using an implicitly registered datetime converter f or a matplotlib plotting method. The converter was registered by pandas on import. Future versions of pandas will require you to explicitly register ma tplotlib converters.

To register the converters:
 >>> from pandas.plotting import register_matplotlib_converters

```
>>> register_matplotlib_converters()
  warnings.warn(msg, FutureWarning)
No handles with labels found to put in legend.
```



3.3.3 Analysis on the Ratings given by user

In [104]:

```
# highest number of ratings given by any user

ratings_by_user= train_df.groupby('user').count().sort_values(by='rating', ascending= False)
ratings_by_user['rating'].head()
```

Out[104]:

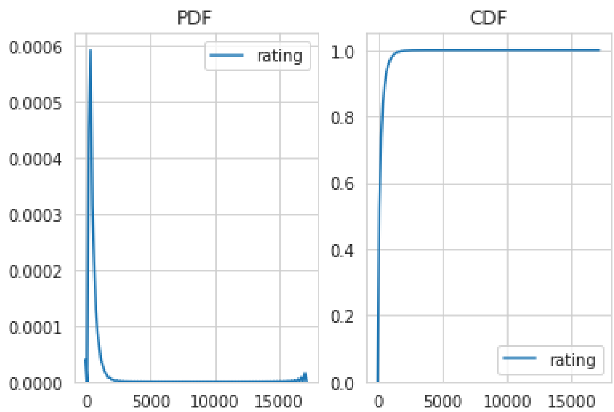
```
user
305344     17112
2439493    15896
387418     15402
1639792     9767
1461435     9447
Name: rating, dtype: int64
```

In [105]:

```
plt.subplot(1, 2, 1)
sns.kdeplot(ratings_by_user['rating'])
plt.title('PDF')
plt.subplot(1,2, 2)
sns.kdeplot(ratings_by_user['rating'], cumulative= True)
plt.title('CDF')
```

Out[105]:

```
Text(0.5, 1.0, 'CDF')
```



In [106]:

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

Out[106]:

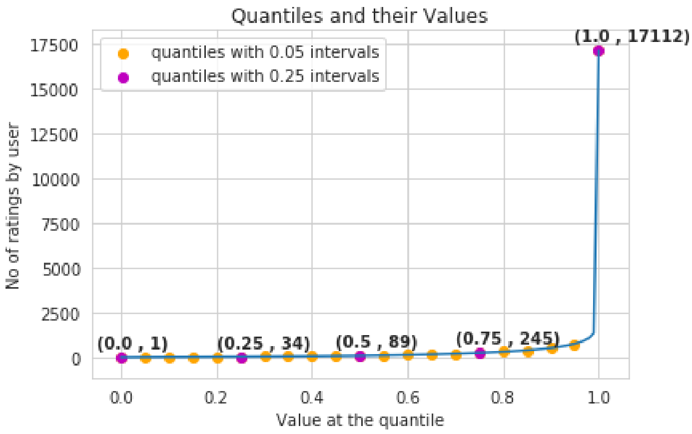
```
count    405041.000000
mean        198.459921
```

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

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

In [107]:

```python
quantiles= ratings_by_user['rating'].quantile(np.arange(0,1.01,0.01), interpolation='higher')

plt.title("Quantiles and their Values")
plt.plot(quantiles)

# quantiles with 0.05 difference => 100/5 = 20 values
plt.scatter(x= quantiles.index[::5], y= quantiles.values[::5], c= 'orange', label= "quantiles with 0.05 intervals")

# quantiles with 0.25 difference => 100/25 = 4 values
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 [108]:

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

Out[108]:

```
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
```

1.00   17112
Name: rating, dtype: int64

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

In [109]:

```
(ratings_by_user['rating']>749).sum()
```

Out[109]:

20242

3.3.4 Analysis of ratings of a movie given by a user

In [110]:

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

Out[110]:

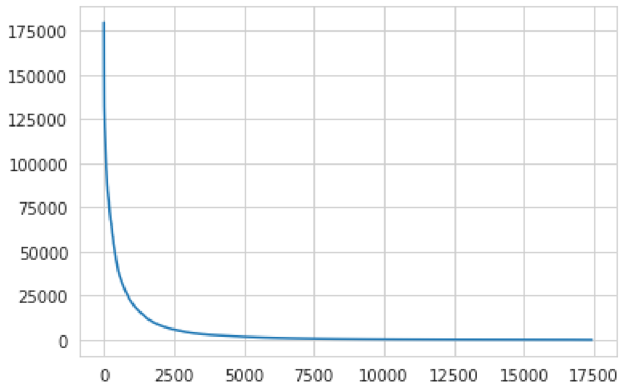
movie
5317   179684
15124   176811
1905   160062
6287   155787
14313   153899
Name: rating, dtype: int64

In [111]:

```
plt.plot(ratings_for_movie.values) # if we dont provide '.values' then the output is different.
```

Out[111]:

[<matplotlib.lines.Line2D at 0x7f6402466be0>]



- **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 [112]:

```
weekday_ratings= train_df.groupby('weekday')['rating'].count()
weekday_ratings
```

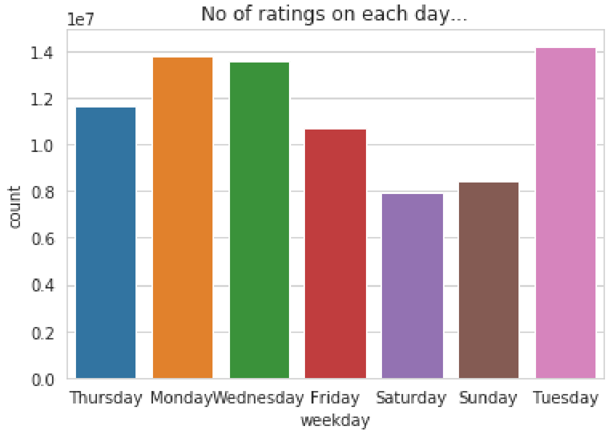
Out[112]:

weekday
Friday   10703108
Monday   13818790
Saturday   7963290

```
Sunday       8445834
Thursday     11634550
Tuesday      14221359
Wednesday    13597474
Name: rating, dtype: int64
```

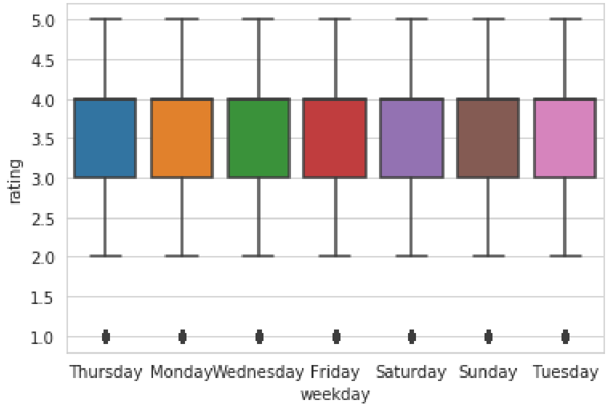
In [113]:

```python
sns.countplot(x='weekday', data= train_df)
plt.title('No of ratings on each day...')
plt.show()
```



In [114]:

```python
start= datetime.now()
sns.boxplot(x='weekday', y= 'rating', data= train_df)
plt.show()
print(datetime.now()-start)
```




```
0:00:26.644181
```

In [115]:

```python
weekday_ratings_mean= train_df.groupby('weekday')['rating'].mean()
weekday_ratings_mean
```

Out[115]:

```
weekday
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 [0]:

```
downloaded = drive.CreateFile({'id':'1a_zcFL3hNOIjrOO2RLtML9FZtd2xR8g1'}) # replace the id with id of file you want to access
downloaded.GetContentFile('train_sparse_matrix.npz')
downloaded = drive.CreateFile({'id':'1YKQ4Y9a2a_Why48eacLPDkiiDH0hu4zX'}) # replace the id with id of file you want to access
downloaded.GetContentFile('test_sparse_matrix.npz')
```

In [14]:

```
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.
    # It should be in such a way that, MATRIX[row, col] = data

    # csr_matrix(data_values, (row_index, col_index), shape_of_matrix)
    train_sparse_matrix = sparse.csr_matrix((train_df.rating.values, (train_df.user.values,
                                               train_df.movie.values)),)

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

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

It is present in your pwd, getting it from disk....
DONE..
0:00:03.900961

The Sparsity of Train Sparse Matrix

In [19]:

```
print(train_sparse_matrix.shape)
# no of ((rows x columns) - non zero elements) / (rows x columns)
print('Percentage of Sparsity: ', ((train_sparse_matrix.shape[0] * train_sparse_matrix.shape[1]) -
      (train_sparse_matrix.count_nonzero())) / (train_sparse_matrix.shape[0] * train_sparse_matrix.shape[1]),'%')
```

(2649430, 17771)
Percentage of Sparsity: 0.998292709259195 %

3.3.6.2 Creating sparse matrix from test data frame

In [15]:

```
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')
```

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

DONE..

The Sparsity of Test Sparse Matrix

In [21]:

```
print(test_sparse_matrix.shape)
# no of ((rows x columns) - non zero elements) / (rows x columns)
print('Percentage of Sparsity: ', ((test_sparse_matrix.shape[0] * test_sparse_matrix.shape[1]) -
      (test_sparse_matrix.count_nonzero())) / (test_sparse_matrix.shape[0] * test_sparse_matrix.shape[1]),'%')
```

(2649430, 17771)
Percentage of Sparsity: 0.9995731772988694 %

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

In [16]:

```
# 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 [17]:

```
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[17]:

{'global': 3.582890686321557}

3.3.7.2 finding average rating per user

In [18]:

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

Average rating of user 10 : 3.3781094527363185

3.3.7.3 finding average rating per movie

In [19]:

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

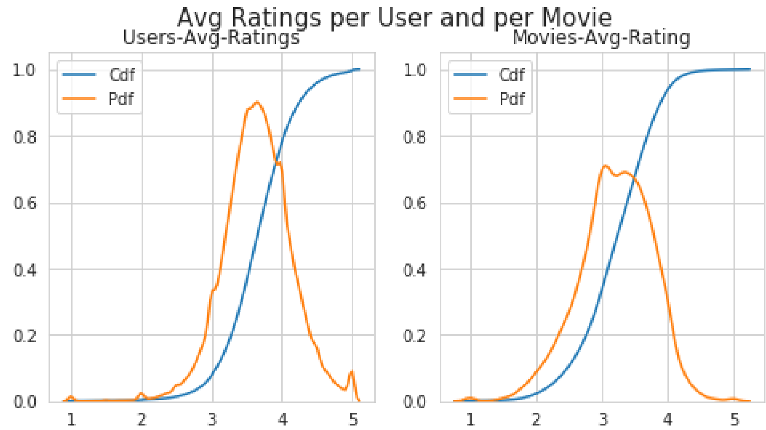
```
print('\n AVerage rating of movie 15 :',train_averages['movie'][15])
```

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 [120]:

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

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

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

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



0:00:34.826430

3.3.8 Cold Start problem

3.3.8.1 Cold Start problem with Users

In [121]:

```
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 %)

In [122]:

```
"""
```

```
# This almost took a lot of time of nearly 2.5 hrs for unique movies of 17770 !!!

start = datetime.now()
count=0
for i in test_df['movie'].unique():
    if i not in train_df['movie'].unique():
        count+=1
print(count)
print(datetime.now()-start)
"""
```

Out[122]:

"\n# This almost took a lot of time of nearly 2.5 hrs for unique movies of 17770 !!!\n\nstart = datetime.now()\ncount=0\nfor i in test_df['movie'].unique():\n   if i not in train_df['movie'].unique():\n   count+=1\nprint(count)\nprint(datetime.now()-start)\n"

In [123]:

```
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 **new users** (***75148***) who didn't appear in train data.

3.3.8.2 Cold Start problem with Movies

> 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 [20]:

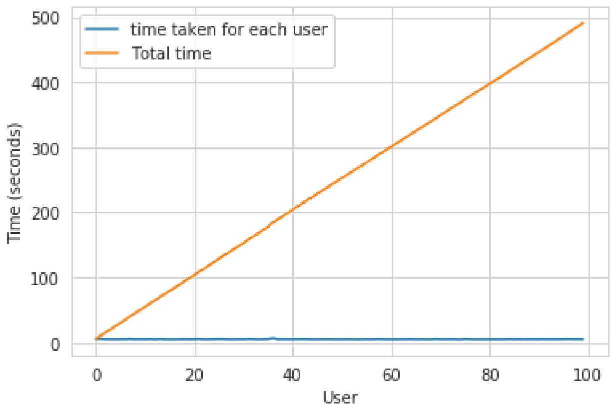
```
def compute_user_similarity(sparse_matrix, compute_for_few=False, top = 100, verbose=False, verb_for_n_rows = 20,
                            draw_time_taken=True):
    no_of_users, _ = sparse_matrix.shape
    # get the indices of  non zero rows(users) from our sparse matrix
    row_ind, col_ind = 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 [125]:

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

Computing top 100 similarities for each user..
computing done for 20 users [time elapsed : 0:01:38.657804]
computing done for 40 users [time elapsed : 0:03:19.138978]
computing done for 60 users [time elapsed : 0:04:56.127406]
computing done for 80 users [time elapsed : 0:06:32.820812]
computing done for 100 users [time elapsed : 0:08:10.999217]
Creating Sparse matrix from the computed similarities



Time taken : 0:08:21.230505

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.
- $405041 \times 8.88 = 3596764.08 \sec = 59946.068 \min = 999.101133333$ hours $= 41.629213889$ days. . .
    - 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 [126]:

```
start= datetime.now()
user_tsvd = TruncatedSVD(n_components= 500, random_state= 5)
tsvd_user = user_tsvd.fit_transform(train_sparse_matrix[:450000])
print(datetime.now()-start)
```

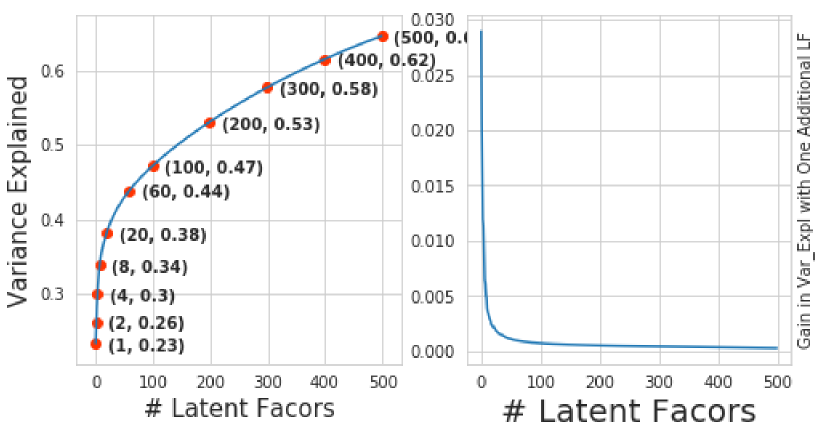
0:03:30.967248

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 [127]:

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

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

ax1.set_ylabel("Variance Explained", fontsize=15)
ax1.set_xlabel("# Latent 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 [128]:

```
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, 0.58)
(400, 0.62)
(500, 0.65)

> 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 [129]:

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

0:00:50.892854

In [130]:

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

Out[130]:

(numpy.ndarray, (2649430, 500))

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

In [21]:

```
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 [22]:

```
trunc_sparse_matrix.shape
```

Out[22]:

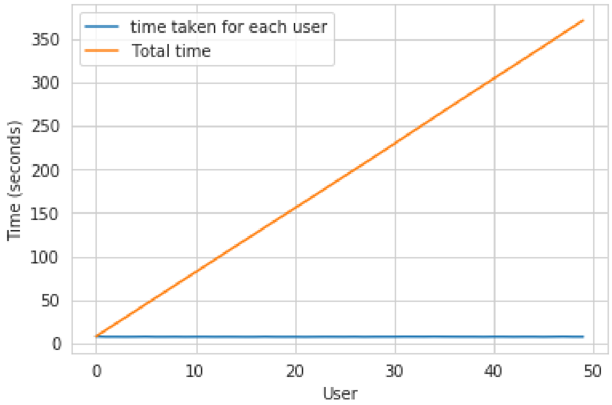
(2649430, 500)

In [23]:

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

```
                                                verb_for_n_rows=10)
print("-"*50)
print("time:",datetime.now()-start)
```

```
Computing top 50 similarities for each user..
computing done for 10 users [  time elapsed : 0:01:14.290835  ]
computing done for 20 users [  time elapsed : 0:02:27.938801  ]
computing done for 30 users [  time elapsed : 0:03:41.737247  ]
computing done for 40 users [  time elapsed : 0:04:56.654651  ]
computing done for 50 users [  time elapsed : 0:06:11.022909  ]
Creating Sparse matrix from the computed similarities
```




```
--------------------------------------------------
time: 0:06:39.122774
```

: 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 × 12.18 ==== 4933399.38sec ==== 82223.323 min ==== 1370.388716667 hours ==== 57.099529861 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 [0]:

```
downloaded = drive.CreateFile({'id':'1TGnEzVnzqqGBxcjpEpVUfa7haqnXrexa'}) # replace the id with id of file you want to access
downloaded.GetContentFile('m_m_sim_sparse.npz')
```

In [24]:

```
start = datetime.now()
if not os.path.isfile('mm_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_new.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
0:08:52.634497
```

- 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 [26]:

```
movie_ids = np.unique(m_m_sim_sparse[:5000].nonzero()[1])
```

In [27]:

```
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[1]
```

```
0:00:29.980393
```

Out[27]:

```
array([  694,  5302,  1084, 13586,  1173,  4181,  8800, 10656, 15648,
       10257, 15100, 10495, 16892,  8408,  7302, 11914, 12125,  3029,
       14182,  8664,  4057, 15192,  2403,  6939,  1248,  2850,  2754,
       15284,  8842,  6178, 15543,  3229,  6009,  8103,   845, 12749,
       16096,  6413,  1883,   221,  2720,  8233, 15159,  6491, 10139,
       17291,  4100,  6781, 11861, 14378,  5534, 17447,   630,  8121,
        1941, 17700,  8573,   588, 15123, 10056,  3279,   486, 17266,
       17063,  8824,   541, 12356,  3377,  2447, 12354, 17613, 15872,
        3081, 10482, 17670,  2794, 12697, 13891, 14184,  9722,  2912,
       12990,  1651, 13430, 10760, 13878,  2459, 14072, 16507,   481,
       15198, 12252,  8203, 15979, 14597, 11727, 14002,  7398, 12928,
        3932])
```

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....

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

In [28]:

```
# 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: 25.31 ms
Type conversion took: 9.82 ms
Parser memory cleanup took: 0.01 ms

Out[28]:

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

Similar Movies for 'Grind'

In [29]:

```
mv_id = 75

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

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

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

Movie -----> Grind

It has 199 Ratings from users.

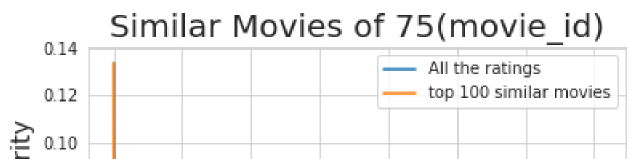
We have 17284 movies which are similar to this and we will get only top most..

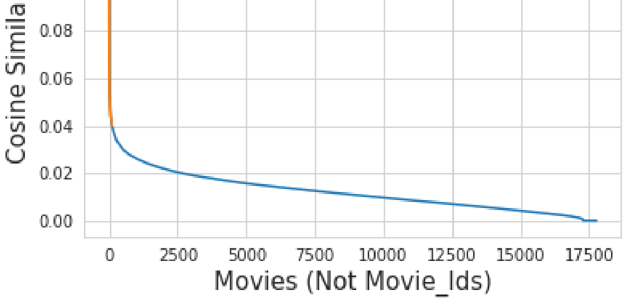
In [30]:

```
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 [31]:

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





In [32]:

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

Out[32]:

	year_of_release	title
movie_id		
15102	2002.0	World Traveler
7275	1999.0	Waking the Dead
6798	2005.0	The Mummy an' the Armadillo
13596	1998.0	Monument Ave.
9867	2002.0	Fairly Oddparents: Scary Godparents
6796	2000.0	Jesus' Son
17183	2005.0	King of the Corner
5715	2004.0	Phil of the Future: Gadgets and Gizmos
14901	1994.0	Guinevere
15906	1997.0	Keys to Tulsa

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

4. Machine Learning Models

In [34]:

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

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

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

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

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

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

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

    return sample_sparse_matrix
```

4.1 Sampling Data

4.1.1 Build sample train data from the train data

In [35]:

```
start = datetime.now()
path = "sample_train_sparse_matrix_new.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 25k users and 5k movies from available data
    sample_train_sparse_matrix= get_sample_sparse_matrix(train_sparse_matrix, no_users=20000, no_movies=2000, path= path)

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

```
It is present in your pwd, getting it from disk....
DONE..
0:00:00.152976
```

4.1.2 Build sample test data from the test data

In [36]:

```
start = datetime.now()

path = "sample_test_sparse_matrix_new.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 10k users and 2.5k movies from available data
    sample_test_sparse_matrix = get_sample_sparse_matrix(test_sparse_matrix, no_users=10000, no_movies=1000, path = path)
print(datetime.now() - start)
```

```
It is present in your pwd, getting it from disk....
DONE..
0:00:00.116590
```

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

In [37]:

```
sample_train_averages = dict()
```

4.2.1 Finding Global Average of all movie ratings

In [38]:

```
# 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[38]:

{'global': 3.5915725873941238}

4.2.2 Finding Average rating per User

In [39]:

```
sample_train_averages['user'] = get_average_ratings(sample_train_sparse_matrix, of_users=True)
print('\nAverage rating of user 917517 :',sample_train_averages['user'][917517])
```

Average rating of user 917517 : 3.235294117647059

4.2.3 Finding Average rating per Movie

In [40]:

```
sample_train_averages['movie'] =  get_average_ratings(sample_train_sparse_matrix, of_users=False)
print('\n AVerage rating of movie 16094 :',sample_train_averages['movie'][16094])
```

AVerage rating of movie 16094 : 2.9477611940298507

4.3 Featurizing data

In [41]:

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

No of ratings in Our Sampled train matrix is : 848398

No of ratings in Our Sampled test matrix is : 73804

4.3.1 Featurizing data for regression problem

4.3.1.1 Featurizing train data

In [42]:

```
# 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 [43]:

```
# 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
```

In [27]:

```
sample_train_averages = dict()

# get the global average of ratings in our train set.
global_average = sample_train_sparse_matrix.sum()/sample_train_sparse_matrix.count_nonzero()
sample_train_averages['global'] = global_average
print('sample train averages: ', sample_train_averages)

sample_train_averages['user'] = get_average_ratings(sample_train_sparse_matrix, of_users=True)
print('\nAverage rating of user 917517 :',sample_train_averages['user'][917517])

sample_train_averages['movie'] =  get_average_ratings(sample_train_sparse_matrix, of_users=False)
print('\n AVerage rating of movie 16094 :',sample_train_averages['movie'][16094])

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

```
sample train averages:  {'global': 3.570537458659482}

Average rating of user 917517 : 3.090909090909091

 AVerage rating of movie 16094 : 2.882978723404255

 No of ratings in Our Sampled train matrix is : 469878


 No of ratings in Our Sampled test  matrix is : 36017
```

In [0]:

```
downloaded = drive.CreateFile({'id':'151IDv7Jm1bIV9fQvEchRJxrocar-Azis'}) # replace the id with id of file you want to access
downloaded.GetContentFile('reg_train.csv')
downloaded = drive.CreateFile({'id':'19tc7aowmstZh5npl3DL8dlI-V1zO0Bom'}) # replace the id with id of file you want to access
downloaded.GetContentFile('reg_test.csv')
```

In [44]:

```
############################################################
# It took me almost 10 hours to prepare this train dataset.#
############################################################

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


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

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

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


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

File already exists you don't have to prepare again...
0:00:00.000733

In [51]:

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

Out[51]:

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

In [52]:

```
reg_train_new.shape
```

Out[52]:

(856986, 16)

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

4.3.1.2 Featurizing test data

In [47]:

```
# 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 [49]:

```
# Used the file provided by another aaic student.

start = datetime.now()

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

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

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

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



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

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

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

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

It is already created...

In [53]:

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

Out[53]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg
0	1129620	2	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581
1	779046	71	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581
2	808635	71	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581
3	898730	71	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581

In [54]:

```
reg_test_new.shape
```

Out[54]:

(72192, 16)

4.3.2 Transforming data for Surprise models

In [39]:

```
!pip3 install surprise
```

```
Collecting surprise
  Downloading https://files.pythonhosted.org/packages/61/de/e5cba8682201fcf9c3719a6fdda95693468ed061945493dea2dd37c5618b/surprise-0.1-py2.py3-none-any.whl
Collecting scikit-surprise
  Downloading https://files.pythonhosted.org/packages/f5/da/b5700d96495fb4f092be497f02492768a3d96a3f4fa2ae7dea46d4081cfa/scikit-surprise-1.1.0.tar.gz (6.4MB)
     |████████████████████████████████| 6.5MB 77kB/s
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (from scikit-surprise->surprise) (0.14.0)
Requirement already satisfied: numpy>=1.11.2 in /usr/local/lib/python3.6/dist-packages (from scikit-surprise->surprise) (1.17.4)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.6/dist-packages (from scikit-surprise->surprise) (1.3.3)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages (from scikit-surprise->surprise) (1.12.0)
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (setup.py) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.0-cp36-cp36m-linux_x86_64.whl size=1678252 sha256=ac508824e9b0ac0576a8d7e0f8cc00fca220e63098babc522f1ed1a1af6082ca
  Stored in directory: /root/.cache/pip/wheels/cc/fa/8c/16c93fccce688ae1bde7d979ff102f7bee980d9cfeb8641bcf
Successfully built scikit-surprise
Installing collected packages: scikit-surprise, surprise
Successfully installed scikit-surprise-1.1.0 surprise-0.1
```

```
Successfully installed scikit-surprise-1.1.0 surprise-0.1
```

In [55]:

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

- 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 [56]:

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

# create the traindata from the dataframe...
train_data = Dataset.load_from_df(reg_train_new[['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

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

In [57]:

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

Out[57]:

```
[(1129620, 2, 3), (779046, 71, 5), (808635, 71, 5)]
```

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 [58]:

```python
# dict variables to hold the outputs of all models.

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

> **Utility functions for running regression models**

In [59]:

```python
# 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):
```

```
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 [60]:

```
# 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 ratings".
    '''
    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 [61]:

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

```
y_train = reg_train_new['rating']

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

In [62]:

```
from sklearn.model_selection import RandomizedSearchCV
import xgboost as xgb
```

In [63]:

```
# Hyper param tuning.
parameters = {'learning_rate':   [0.01, 0.1], # 2 components
              'n_estimators':    [50, 100, 150], # 3 components
              'max_depth':       [2,3,4], # 3 components
              'min_child_weight':[3, 5], # 2 components
              'sub_sample':      [0.6, 0.8], # 2 components
              'colsample_bytree':[0.6, 0.8], # 2 components
              'gamma':           [0, 0.1], # 2 components
              'reg_alpha':       [0.005, 0.01], # 2 components
              'reg_lambda':      [0.005, 0.01]} # 2 components

# total of 768 combinations used for randomized search
rscv = RandomizedSearchCV(estimator = xgb.XGBRegressor(nthread=4, n_jobs=-1), param_distributions= parameters, cv= 2,
                    n_jobs= -1, return_train_score=True, scoring = 'r2')
rscv.fit(x_train, y_train)
```

```
/home/passionateguy_bharat/.local/lib/python3.5/site-packages/xgboost/core.py:587: FutureWarning: Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/home/passionateguy_bharat/.local/lib/python3.5/site-packages/xgboost/core.py:588: FutureWarning: Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
```

```
[10:39:37] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

Out[63]:

```
RandomizedSearchCV(cv=2, error_score='raise-deprecating',
          estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                         colsample_bylevel=1,
                         colsample_bynode=1,
                         colsample_bytree=1, gamma=0,
                         importance_type='gain',
                         learning_rate=0.1, max_delta_step=0,
                         max_depth=3, min_child_weight=1,
                         missing=None, n_estimators=100,
                         n_jobs=-1, nthread=4,
                         objective='reg:linear',
                         random_stat...
          iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'colsample_bytree': [0.6, 0.8],
                        'gamma': [0, 0.1],
                        'learning_rate': [0.01, 0.1],
                        'max_depth': [2, 3, 4],
                        'min_child_weight': [3, 5],
                        'n_estimators': [50, 100, 150],
                        'reg_alpha': [0.005, 0.01],
                        'reg_lambda': [0.005, 0.01],
                        'sub_sample': [0.6, 0.8]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score=True, scoring='r2', verbose=0)
```

In [64]:

```
print('Best parameters: \n', rscv.best_estimator_)
print()
print('neg_mean_absolute_error:', rscv.score(x_test, y_test))
```

```
Best parameters:
 XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=0.6, gamma=0,
       importance_type='gain', learning_rate=0.1, max_delta_step=0,
       max_depth=2, min_child_weight=3, missing=None, n_estimators=100,
       n_jobs=-1, nthread=4, objective='reg:linear', random_state=0,
```

```
            reg_alpha=0.01, reg_lambda=0.005, scale_pos_weight=1, seed=None,
            silent=None, sub_sample=0.6, subsample=1, verbosity=1)
```

neg_mean_absolute_error: 0.010537179555950904

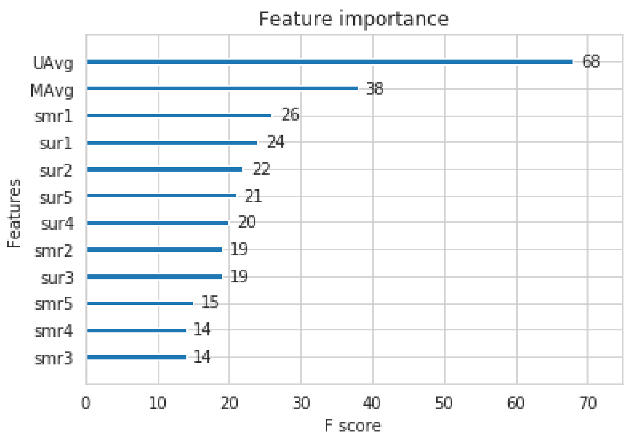
In [65]:

```python
# initialize Our first XGBoost model...
first_xgb = xgb.XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
            colsample_bynode=1, colsample_bytree=0.6, gamma=0,
            importance_type='gain', learning_rate=0.1, max_delta_step=0,
            max_depth=2, min_child_weight=3, missing=None, n_estimators=100,
            n_jobs=-1, nthread=4, objective='reg:linear', random_state=0,
            reg_alpha=0.01, reg_lambda=0.005, scale_pos_weight=1, seed=None,
            silent=None, sub_sample=0.6, subsample=1, verbosity=1)

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..
```

```
/home/passionateguy_bharat/.local/lib/python3.5/site-packages/xgboost/core.py:587: FutureWarning: Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/home/passionateguy_bharat/.local/lib/python3.5/site-packages/xgboost/core.py:588: FutureWarning: Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
```

```
[10:39:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Done. Time taken : 0:00:08.946443

Done

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

TEST DATA
------------------------------
RMSE :  1.0887173978827296
MAPE :  35.103864167700245
```



4.4.2 Suprise BaselineModel

In [66]:

```python
from surprise import BaselineOnly
```

Predicted_rating : (baseline prediction)

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

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

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

- $\mu\mu$: Average of all ratings in training data.
- bb_u : User bias
- bb_i : Item bias (movie biases)

Optimization function (Least Squares Problem)

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

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

In [50]:

```
# options are to specify.., how to compute those user and item biases
params = {'method': 'sgd',
          'learning_rate': .001}

bsl_algo = BaselineOnly(bsl_options= params)

# 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 : 0:00:07.148251

Evaluating the model with train data..
time taken : 0:00:08.183330
---------------
Train Data
---------------
RMSE : 0.9220478981418425

MAPE : 28.6415868708249

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.606168
---------------
Test Data
---------------
RMSE : 1.0863663098706433

MAPE : 34.9272700831115

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:00:15.939966
```

4.4.3 XGBoost with initial 13 features + Surprise Baseline predictor

Updating Train Data

In [53]:

```
# output of 'run_surprise' function
models_evaluation_train['bsl_algo']
```

Out[53]:

```
{'mape': 28.6415868708249,
 'predictions': array([3.68139346, 3.72015018, 4.51053701, ..., 3.91562856, 3.93909859,
        3.91926974]),
 'rmse': 0.9220478981418425}
```

In [52]:

```
# add our baseline_predicted value as our 14th new feature (predictions of baseline) => (13 + 1 =14 features in total)
reg_train_new['bslpr'] = models_evaluation_train['bsl_algo']['predictions']
reg_train_new.head(2)
```

Out[52]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating	bslpr
0	174683	10	3.587581	5.0	5.0	3.0	4.0	4.0	3.0	5.0	4.0	3.0	2.0	3.882353	3.611111	5	3.681393
1	233949	10	3.587581	4.0	4.0	5.0	1.0	3.0	2.0	3.0	2.0	3.0	3.0	2.692308	3.611111	3	3.720150

Updating Test Data

In [54]:

```
models_evaluation_test['bsl_algo']
```

Out[54]:

```
{'mape': 34.9272700831115,
 'predictions': array([3.58758136, 3.58758136, 3.58758136, ..., 3.58758136, 3.58758136,
        3.58758136]),
 'rmse': 1.0863663098706433}
```

In [55]:

```
reg_test_new['bslpr'] = models_evaluation_test['bsl_algo']['predictions']
reg_test_new.head(2)
```

Out[55]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg
0	1129620	2	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581
1	779046	71	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581

In [0]:

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

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

In [0]:

```
# Hyper param tuning.
parameters = {'learning_rate':   [0.01, 0.1], # 2 components
              'n_estimators':    [50, 100, 150], # 3 components
              'max_depth':       [2,3,4], # 3 components
              'min_child_weight':[3, 5], # 2 components
              'sub_sample':      [0.6, 0.8], # 2 components
              'colsample_bytree':[0.6, 0.8], # 2 components
              'gamma':           [0, 0.1], # 2 components
              'reg_alpha':       [0.005, 0.01], # 2 components
              'reg_lambda':      [0.005, 0.01]} # 2 components

# total of 768 combinations used for randomized search
rscv = RandomizedSearchCV(estimator = xgb.XGBRegressor(nthread=4, n_jobs=-1), param_distributions= parameters, n_jobs= -1,
                    return_train_score=True, scoring = 'r2')
rscv.fit(x_train, y_train)
```

In [0]:

```
print('Best parameters: \n', rscv.best_estimator_)
print()
print('neg_mean_absolute_error:', rscv.score(x_test, y_test))
```

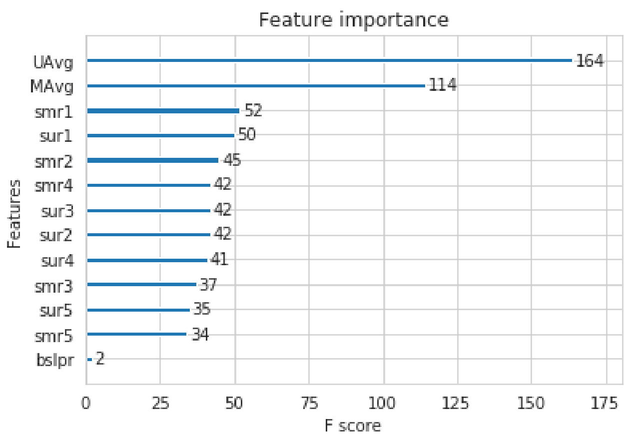
In [57]:

```
# initialize Our first XGBoost model...
xgb_bsl = xgb.XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
          colsample_bynode=1, colsample_bytree=0.8, gamma=0,
          importance_type='gain', learning_rate=0.1, max_delta_step=0,
          max_depth=3, min_child_weight=3, missing=None, n_estimators=100,
          n_jobs=-1, nthread=4, objective='reg:linear', random_state=0,
          reg_alpha=0.005, reg_lambda=0.01, scale_pos_weight=1, seed=None,
          silent=None, sub_sample=0.6, subsample=1, verbosity=1)

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..
```

```
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning: Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning: Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
```

```
[07:51:08] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Done. Time taken : 0:00:17.755979

Done

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

TEST DATA
------------------------------
RMSE :  1.092464637038003
MAPE :  34.86015569350967
```



4.4.4 Surprise KNNBaseline predictor

In [0]:

```
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***)

$$\frac{\sum\limits_{v \in N_i^k(u)} \text{sim}(u, v) \cdot (r_{vi} - b_{vi})}{\sum}$$

$$\hat{r}_{ui} = b_{ui} + \quad \sum_{v \in N_i^k(u)} \text{sim}(u, v)$$

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

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

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

4.4.4.1 Surprise KNNBaseline with movie movie similarities

In [59]:

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

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

sim_params = {'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_params = {'method': 'sgd'}

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

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 : 0:00:15.477464

Evaluating the model with train data..
time taken : 0:02:02.597895
---------------
Train Data
---------------
RMSE : 0.5038994796517224

MAPE : 14.168515366483724

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.725691
---------------
Test Data
---------------
RMSE : 1.0871254774375978

MAPE : 34.9334477168073

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:02:18.802668
```

4.4.4.2 Surprise KNNBaseline with user user similarities

In [60]:

```
# we specify , how to compute similarities and what to consider with sim_options to our algorithm
sim_params = {'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_params = {'method': 'sgd'}

knn_bsl_u = KNNBaseline(k=40, sim_options = sim_params, bsl_options = bsl_params)
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 : 0:07:13.685311

Evaluating the model with train data..
time taken : 0:28:29.860786
---------------
Train Data
---------------
RMSE : 0.4536279292470732

MAPE : 12.840252350475915

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:01.330704
---------------
Test Data
---------------
RMSE : 1.0868961034865674

MAPE : 34.93095406703468

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:35:44.878760
```

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 [63]:

```
models_evaluation_train['knn_bsl_u']
```

Out[63]:

```
{'mape': 12.840252350475915,
 'predictions': array([4.98449486, 3.18129643, 4.9012647 , ..., 3.        , 5.        ,
        4.        ]),
 'rmse': 0.4536279292470732}
```

In [64]:

```
models_evaluation_train['knn_bsl_m']
```

Out[64]:

```
{'mape': 14.168515366483724,
 'predictions': array([4.88478232, 3.29593441, 4.9676822 , ..., 3.        , 5.        ,
        4.        ]),
 'rmse': 0.5038994796517224}
```

'rmse': 0.5038994796517224}

In [66]:

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

reg_train_new.head(2)
```

Out[66]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating	bslpr	knn_bsl_u	knn_bsl_
0	174683	10	3.587581	5.0	5.0	3.0	4.0	4.0	3.0	5.0	4.0	3.0	2.0	3.882353	3.611111	5	3.681393	4.984495	4.8847
1	233949	10	3.587581	4.0	4.0	5.0	1.0	3.0	2.0	3.0	2.0	3.0	3.0	2.692308	3.611111	3	3.720150	3.181296	3.2959

Preparing Test data

In [67]:

```
models_evaluation_test['knn_bsl_u']
```

Out[67]:

```
{'mape': 34.93095406703468,
 'predictions': array([3.58758136, 3.58758136, 3.58758136, ..., 3.58758136, 3.58758136,
        3.58758136]),
 'rmse': 1.0868961034865674}
```

In [68]:

```
models_evaluation_test['knn_bsl_m']
```

Out[68]:

```
{'mape': 34.9334477168073,
 'predictions': array([3.58758136, 3.58758136, 3.58758136, ..., 3.58758136, 3.58758136,
        3.58758136]),
 'rmse': 1.0871254774375978}
```

In [69]:

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

reg_test_new.head(2)
```

Out[69]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg
0	1129620	2	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581
1	779046	71	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581

In [0]:

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

# prepare the train data....
x_test = reg_test_new.drop(['user','movie','rating'], axis=1)
y_test = reg_test_new['rating']
```

In [71]:

```
# Hyper param tuning.
parameters = {'learning_rate':   [0.01, 0.1], # 2 components
            'n_estimators':   [50, 100, 150], # 3 components
            'max_depth':       [2,3,4], # 3 components
            'min_child_weight':[3, 5], # 2 components
            'sub_sample':      [0.6, 0.8], # 2 components
```

```
            'colsample_bytree':[0.6, 0.8], # 2 components
            'gamma':         [0, 0.1], # 2 components
            'reg_alpha':      [0.005, 0.01], # 2 components
            'reg_lambda':      [0.005, 0.01]} # 2 components

# total of 768 combinations used for randomized search
rscv = RandomizedSearchCV(estimator = xgb.XGBRegressor(nthread=4, n_jobs=-1), param_distributions= parameters, n_jobs= -1,
                    return_train_score=True, scoring = 'r2')
rscv.fit(x_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default value of cv will change from 3 to 5 in version 0.22. Specify it explicitly to silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning: Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning: Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
```

```
[08:40:09] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

Out[71]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
          estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                          colsample_bylevel=1,
                          colsample_bynode=1,
                          colsample_bytree=1, gamma=0,
                          importance_type='gain',
                          learning_rate=0.1, max_delta_step=0,
                          max_depth=3, min_child_weight=1,
                          missing=None, n_estimators=100,
                          n_jobs=-1, nthread=4,
                          objective='reg:linear',
                          random...
          iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'colsample_bytree': [0.6, 0.8],
                         'gamma': [0, 0.1],
                         'learning_rate': [0.01, 0.1],
                         'max_depth': [2, 3, 4],
                         'min_child_weight': [3, 5],
                         'n_estimators': [50, 100, 150],
                         'reg_alpha': [0.005, 0.01],
                         'reg_lambda': [0.005, 0.01],
                         'sub_sample': [0.6, 0.8]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score=True, scoring='r2', verbose=0)
```

In [72]:

```
print('Best parameters: \n', rscv.best_estimator_)
print()
print('neg_mean_absolute_error:', rscv.score(x_test, y_test))
```

```
Best parameters:
 XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=0.6, gamma=0,
       importance_type='gain', learning_rate=0.1, max_delta_step=0,
       max_depth=2, min_child_weight=3, missing=None, n_estimators=100,
       n_jobs=-1, nthread=4, objective='reg:linear', random_state=0,
       reg_alpha=0.01, reg_lambda=0.005, scale_pos_weight=1, seed=None,
       silent=None, sub_sample=0.6, subsample=1, verbosity=1)

neg_mean_absolute_error: 0.011646380887316399
```

In [73]:

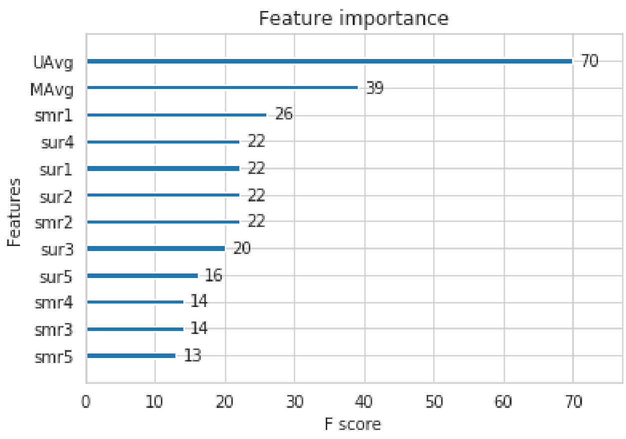
```
# declare the model
xgb_knn_bsl = xgb.XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=0.6, gamma=0,
       importance_type='gain', learning_rate=0.1, max_delta_step=0,
       max_depth=2, min_child_weight=3, missing=None, n_estimators=100,
       n_jobs=-1, nthread=4, objective='reg:linear', random_state=0,
       reg_alpha=0.01, reg_lambda=0.005, scale_pos_weight=1, seed=None,
       silent=None, sub_sample=0.6, subsample=1, verbosity=1)
train_results, test_results = run_xgboost(xgb_knn_bsl, x_train, y_train, x_test, y_test)

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

```
models_evaluation_test['xgb_knn_bsl'] = test_results


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

```
Training the model..
```

```
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning: Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning: Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
```

```
[08:41:12] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Done. Time taken : 0:00:13.263243

Done

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

TEST DATA
------------------------------
RMSE :  1.0881069932328495
MAPE :  35.35244502169828
```



4.4.6 Matrix Factorization Techniques

4.4.6.1 SVD Matrix Factorization User Movie intractions

In [0]:

```
from surprise import import 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)**
  - $\sum_{r_{ui} \in R_{train}} \left(r_{ui} - \hat{r}_{ui} \right)^2 + \lambda\left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2\right)$

In [75]:

```
# 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)
```

```python
# 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 : 0:00:52.615455

Evaluating the model with train data..
time taken : 0:00:09.203319
---------------
Train Data
---------------
RMSE : 0.6746731413267192

MAPE : 20.05479554670084

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:01.144430
---------------
Test Data
---------------
RMSE : 1.0864183392580073

MAPE : 34.86051459384974

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:01:02.965255
```

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

In [0]:

```python
from surprise import SVDpp
```

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

- ## Predicted Rating :
  - $\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j\right)$
  - I_u --- the set of all items rated by user u
- y_j --- Our new set of item factors that capture implicit ratings.

- ## Optimization problem with user item interactions and regularization (to avoid overfitting)
  - $\sum_{r_{ui} \in R_{train}} \left(r_{ui} - \hat{r}_{ui}\right)^2 + \lambda\left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2 + ||y_j||^2\right)$

In [77]:

```
# 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 : 0:35:58.485374

Evaluating the model with train data..
time taken : 0:01:32.066327
---------------
Train Data
---------------
RMSE : 0.6641918784333875

MAPE : 19.24213231265533

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:01.223381
---------------
Test Data
---------------
RMSE : 1.0868790316621306

MAPE : 34.82076787455494

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:37:31.777362
```

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

Preparing Train data

In [78]:

```
models_evaluation_train['svd']
```

Out[78]:

```
{'mape': 20.05479554670084,
 'predictions': array([4.07334811, 3.64907292, 4.80044776, ..., 3.80136179, 4.1158486 ,
        4.28123386]),
 'rmse': 0.6746731413267192}
```

In [79]:

```
models_evaluation_train['svdpp']
```

Out[79]:

```
{'mape': 19.24213231265533,
```

'predictions': array([3.88411495, 3.61847613, 4.6160544 , ..., 3.56489355, 4.33083611,
 4.13181654]),
'rmse': 0.6641918784333875}

In [80]:

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

reg_train_new.head(2)
```

Out[80]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating	bslpr	knn_bsl_u	knn_bsl_
0	174683	10	3.587581	5.0	5.0	3.0	4.0	4.0	3.0	5.0	4.0	3.0	2.0	3.882353	3.611111	5	3.681393	4.984495	4.8847
1	233949	10	3.587581	4.0	4.0	5.0	1.0	3.0	2.0	3.0	2.0	3.0	3.0	2.692308	3.611111	3	3.720150	3.181296	3.2959

Preparing Test data

In [81]:

```
models_evaluation_test['svd']
```

Out[81]:

{'mape': 34.86051459384974,
'predictions': array([3.58758136, 3.58758136, 3.58758136, ..., 3.58758136, 3.58758136,
 3.58758136]),
'rmse': 1.0864183392580073}

In [82]:

```
models_evaluation_test['svdpp']
```

Out[82]:

{'mape': 34.82076787455494,
'predictions': array([3.58758136, 3.58758136, 3.58758136, ..., 3.58758136, 3.58758136,
 3.58758136]),
'rmse': 1.0868790316621306}

In [83]:

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

reg_test_new.head(2)
```

Out[83]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg
0	1129620	2	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581
1	779046	71	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581

In [0]:

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

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

In [85]:

```
# Hyper param tuning.
parameters = {'learning_rate':   [0.01, 0.1], # 2 components
              'n_estimators':    [50, 100, 150], # 3 components
```

```
            'n_estimators':   [50, 100, 150], # 3 components
            'max_depth':      [2,3,4], # 3 components
            'min_child_weight':[3, 5], # 2 components
            'sub_sample':     [0.6, 0.8], # 2 components
            'colsample_bytree':[0.6, 0.8], # 2 components
            'gamma':          [0, 0.1], # 2 components
            'reg_alpha':      [0.005, 0.01], # 2 components
            'reg_lambda':     [0.005, 0.01]} # 2 components

# total of 768 combinations used for randomized search
rscv = RandomizedSearchCV(estimator = xgb.XGBRegressor(nthread=4, n_jobs=-1), param_distributions= parameters, n_jobs= -1,
                    return_train_score=True, scoring = 'r2')
rscv.fit(x_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default value of cv will change from 3 to 5 in versio
n 0.22. Specify it explicitly to silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning: Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning: Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
```

```
[09:26:00] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

Out[85]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
          estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                        colsample_bylevel=1,
                        colsample_bynode=1,
                        colsample_bytree=1, gamma=0,
                        importance_type='gain',
                        learning_rate=0.1, max_delta_step=0,
                        max_depth=3, min_child_weight=1,
                        missing=None, n_estimators=100,
                        n_jobs=-1, nthread=4,
                        objective='reg:linear',
                        random...
          iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'colsample_bytree': [0.6, 0.8],
                        'gamma': [0, 0.1],
                        'learning_rate': [0.01, 0.1],
                        'max_depth': [2, 3, 4],
                        'min_child_weight': [3, 5],
                        'n_estimators': [50, 100, 150],
                        'reg_alpha': [0.005, 0.01],
                        'reg_lambda': [0.005, 0.01],
                        'sub_sample': [0.6, 0.8]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score=True, scoring='r2', verbose=0)
```

In [86]:

```
print('Best parameters: \n', rscv.best_estimator_)
print()
print('neg_mean_absolute_error:', rscv.score(x_test, y_test))
```

```
Best parameters:
 XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=0.8, gamma=0,
       importance_type='gain', learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=3, missing=None, n_estimators=100,
       n_jobs=-1, nthread=4, objective='reg:linear', random_state=0,
       reg_alpha=0.005, reg_lambda=0.01, scale_pos_weight=1, seed=None,
       silent=None, sub_sample=0.6, subsample=1, verbosity=1)

neg_mean_absolute_error: 0.0021508264973544
```

In [87]:

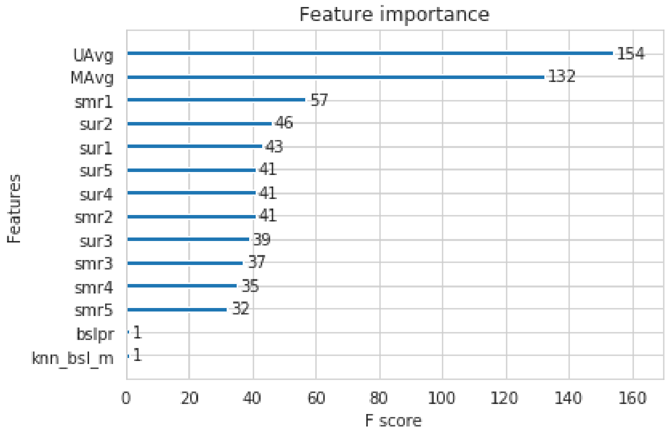
```
xgb_final = xgb.XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=0.8, gamma=0,
       importance_type='gain', learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=3, missing=None, n_estimators=100,
       n_jobs=-1, nthread=4, objective='reg:linear', random_state=0,
       reg_alpha=0.005, reg_lambda=0.01, scale_pos_weight=1, seed=None,
       silent=None, sub_sample=0.6, subsample=1, verbosity=1)
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..
```

```
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning: Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning: Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
```

```
[09:26:55] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Done. Time taken : 0:00:22.721396

Done

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

TEST DATA
------------------------------
RMSE :  1.0933214635262558
MAPE :  34.76531616343575
```



4.4.8 XgBoost with Surprise Baseline + Surprise KNNbaseline + MF Techniques

In [0]:

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

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

In [90]:

```
# Hyper param tuning.
parameters = {'learning_rate':   [0.01, 0.1], # 2 components
            'n_estimators':    [50, 100, 150], # 3 components
            'max_depth':       [2,3,4], # 3 components
            'min_child_weight':[3, 5], # 2 components
            'sub_sample':      [0.6, 0.8], # 2 components
            'colsample_bytree':[0.6, 0.8], # 2 components
            'gamma':           [0, 0.1], # 2 components
            'reg_alpha':       [0.005, 0.01], # 2 components
            'reg_lambda':      [0.005, 0.01]} # 2 components

# total of 768 combinations used for randomized search
rscv = RandomizedSearchCV(estimator = xgb.XGBRegressor(nthread=4, n_jobs=-1), param_distributions= parameters, n_jobs= -1,
                return_train_score=True, scoring = 'r2')
rscv.fit(x_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default value of cv will change from 3 to 5 in versio
n 0.22. Specify it explicitly to silence this warning.
```

```
  warnings.warn(CV_WARNING, FutureWarning)
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning: Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning: Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
```

```
[09:31:43] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

Out[90]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
          estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                    colsample_bylevel=1,
                    colsample_bynode=1,
                    colsample_bytree=1, gamma=0,
                    importance_type='gain',
                    learning_rate=0.1, max_delta_step=0,
                    max_depth=3, min_child_weight=1,
                    missing=None, n_estimators=100,
                    n_jobs=-1, nthread=4,
                    objective='reg:linear',
                    random...
          iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'colsample_bytree': [0.6, 0.8],
                    'gamma': [0, 0.1],
                    'learning_rate': [0.01, 0.1],
                    'max_depth': [2, 3, 4],
                    'min_child_weight': [3, 5],
                    'n_estimators': [50, 100, 150],
                    'reg_alpha': [0.005, 0.01],
                    'reg_lambda': [0.005, 0.01],
                    'sub_sample': [0.6, 0.8]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score=True, scoring='r2', verbose=0)
```

In [91]:

```python
print('Best parameters: \n', rscv.best_estimator_)
print()
print('neg_mean_absolute_error:', rscv.score(x_test, y_test))
```

```
Best parameters:
 XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=0.8, gamma=0.1,
       importance_type='gain', learning_rate=0.1, max_delta_step=0,
       max_depth=2, min_child_weight=5, missing=None, n_estimators=50,
       n_jobs=-1, nthread=4, objective='reg:linear', random_state=0,
       reg_alpha=0.01, reg_lambda=0.01, scale_pos_weight=1, seed=None,
       silent=None, sub_sample=0.8, subsample=1, verbosity=1)

neg_mean_absolute_error: -0.003171467865579425
```

In [92]:

```python
xgb_all_models = xgb.XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=0.8, gamma=0.1,
       importance_type='gain', learning_rate=0.1, max_delta_step=0,
       max_depth=2, min_child_weight=5, missing=None, n_estimators=50,
       n_jobs=-1, nthread=4, objective='reg:linear', random_state=0,
       reg_alpha=0.01, reg_lambda=0.01, scale_pos_weight=1, seed=None,
       silent=None, sub_sample=0.8, subsample=1, verbosity=1)
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..
```

```
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning: Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning: Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
```

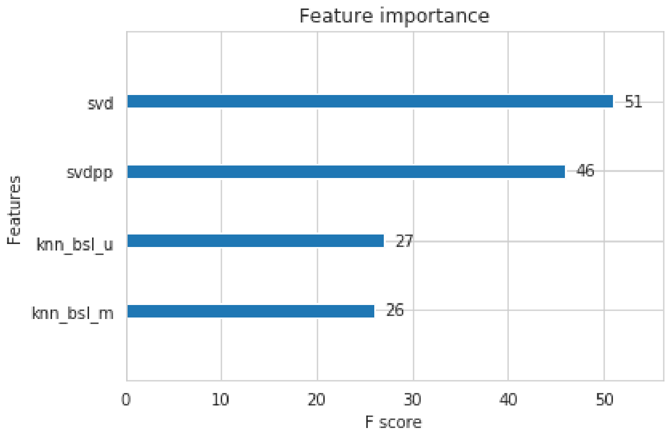
[09:32:47] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
Done. Time taken : 0:00:05.749484

Done

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

TEST DATA

RMSE : 1.0962333464904743
MAPE : 35.422529611306096



4.5 Comparision between all models

In [94]:

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

Out[94]:

```
bsl_algo          1.0863663098706433
svd               1.0864183392580073
svdpp             1.0868790316621306
knn_bsl_u         1.0868961034865674
knn_bsl_m         1.0871254774375978
xgb_knn_bsl       1.0881069932328495
first_algo        1.0887173978827296
xgb_bsl           1.092464637038003
xgb_final         1.0933214635262558
xgb_all_models    1.0962333464904743
Name: rmse, dtype: object
```