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_s (we use many models from this library)
- installing surprise: https://github.com/NicolasHug/Surprise#installat
- 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
  </ul>

```
The first line of each file [combine
d_data_1.txt, combined_data_2.txt, c
ombined_data_3.txt, combined_data_4.
txt] contains the movie id followed
by a colon. Each subsequent line in
the file corresponds to a rating fro
m a customer and its date in the fol
lowing format:

CustomerID,Rating,Date

MovieIDs range from 1 to 17770 seque
ntially.
CustomerIDs range from 1 to 2649429,
with gaps. There are 480189 users.
Ratings are on a five star (integra
l) 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 pr
edict the rating would be given by him/h
er to the movie.
The given problem is a Recommendation pr
oblem
It can also seen as a Regression problem
```

2.2.2 Performance metric

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

2.2.3 Machine Learning Objective and Constraints

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

In [1]:
```
# this is just to know how much time will it ta
ke to run this entire ipython notebook
from datetime import datetime
# globalstart = datetime.now()
import pandas as pd
import numpy as np
import matplotlib
matplotlib.use('nbagg')

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

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

from sklearn.decomposition import TruncatedSVD
from sklearn.metrics.pairwise import cosine_sim
ilarity
import random
```

3. Exploratory Data Analysis

3.1 Preprocessing

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

In [0]:

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

    row = list()
    files=['data_folder/combined_data_1.txt','d
ata_folder/combined_data_2.txt',
           'data_folder/combined_data_3.txt',
'data_folder/combined_data_4.txt']
    for file in files:
        print("Reading ratings from {}...".form
at(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.spli
t(',')]
                    row.insert(0, movie_id)
                    data.write(','.join(row))
                    data.write('\n')
        print("Done.\n")
    data.close()
print('Time taken :', datetime.now() - start)
```

```
Reading ratings from data_folder/combined
_data_1.txt...
Done.

Reading ratings from data_folder/combined
_data_2.txt...
Done.

Reading ratings from data_folder/combined
_data_3.txt...
Done.

Reading ratings from data_folder/combined

_data_4.txt...
Done.

Time taken : 0:05:03.705966
```

In [2]:

```
print("creating the dataframe from data.csv fil
e..")
df = pd.read_csv('data.csv', sep=',',
                       names=['movie', 'user',
'rating','date'])
df.date = pd.to_datetime(df.date)
```

```
print('Done.\n')

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

```
creating the dataframe from data.csv fil
e..
Done.

Sorting the dataframe by date..
Done..
```

In [3]:
```
df.head()
```

Out[3]:

	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 [4]:
```
df.describe()['rating']
```

Out[4]:
```
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 [5]:

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

```
No of Nan values in our dataframe :  0
```

3.1.3 Removing Duplicates

In [6]:

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

```
There are 0 duplicate rating entries in t
he data..
```

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

In [7]:

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

```
Total data
-----------------------------------------
---------

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

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

In [8]:
```
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("tra
in.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("tes
t.csv", index=False)
```

In [2]:
```
train_df = pd.read_csv("train.csv", parse_dates
=['date'])
test_df = pd.read_csv("test.csv")
```

In [65]:
```
print(train_df.shape)
train_df.head()
```

```
(80384405, 4)
```

Out[65]:

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

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

In [4]:
```
# movies = train_df.movie.value_counts()
# users = train_df.user.value_counts()
print("Training data ")
print("-"*50)
print("\nTotal no of ratings :",train_df.shape[
0])
print("Total No of Users   :", len(np.unique(tr
ain_df.user)))
print("Total No of movies  :", len(np.unique(tr
ain_df.movie)))
```

```
Training data
-----------------------------------------


---------


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

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

In [5]:
```
print("Test data ")
print("-"*50)
print("\nTotal no of ratings :",test_df.shape[0
])
print("Total No of Users   :", len(np.unique(te
st_df.user)))
print("Total No of movies  :", len(np.unique(te
st_df.movie)))
```

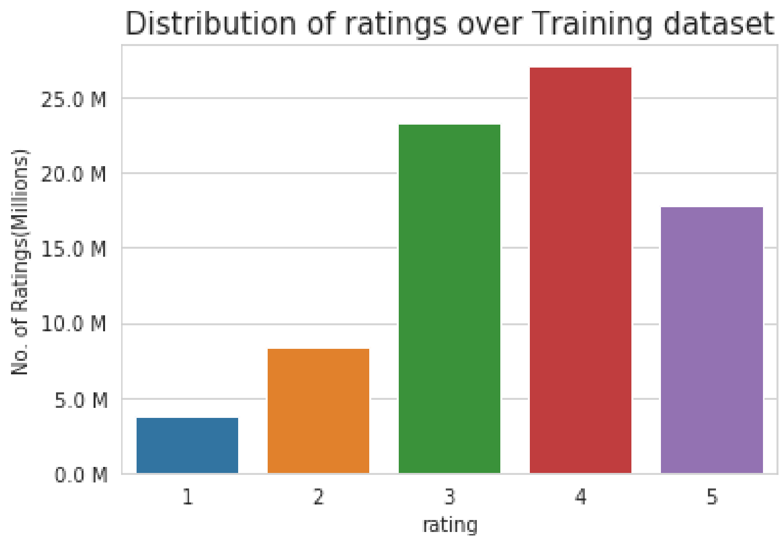
```
Test data
-----------------------------------------
---------

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

3.3 Exploratory Data Analysis on Train data

In [6]:
```
# 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 [14]:
```
%matplotlib inline
fig, ax = plt.subplots()
plt.title('Distribution of ratings over Trainin
g dataset', fontsize=15)
sns.countplot(train_df.rating)
ax.set_yticklabels([human(item, 'M') for item i
n ax.get_yticks()])
ax.set_ylabel('No. of Ratings(Millions)')

plt.show()
```



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

In [7]:
```
# It is used to skip the warning ''SettingWithC
```

```
opyWarning''..
pd.options.mode.chained_assignment = None  # de
fault='warn'

train_df['day_of_week'] = train_df.date.dt.week
day_name

train_df.tail()
```

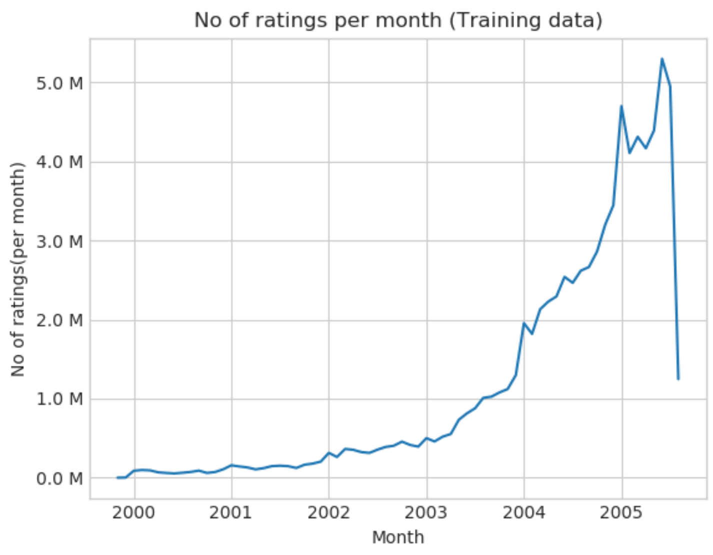
Out[7]:

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

3.3.2 Number of Ratings per a month

In [0]:

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



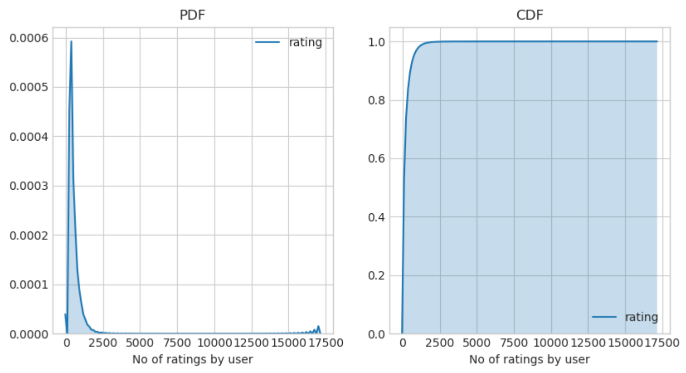
3.3.3 Analysis on the Ratings given by user

In [8]:
```
no_of_rated_movies_per_user = train_df.groupby(
by='user')['rating'].count().sort_values(ascend
ing=False)

no_of_rated_movies_per_user.head()
```

Out[8]:
```
user
305344     17112
2439493    15896
387418     15402
1639792     9767
1461435     9447
Name: rating, dtype: int64
```

In [0]:
```
fig = plt.figure(figsize=plt.figaspect(.5))
```

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

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

plt.show()
```



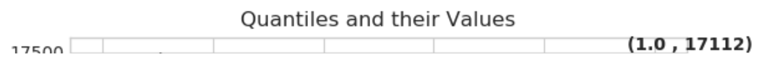
In [9]:
```
no_of_rated_movies_per_user.describe()
```

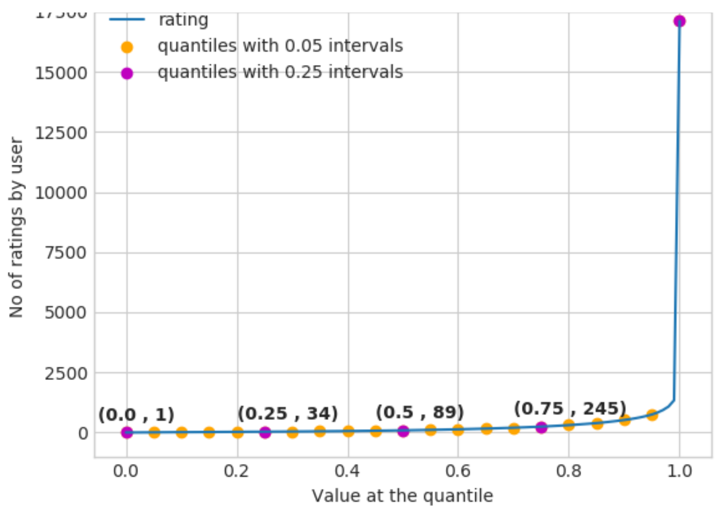
Out[9]:
```
count    405041.000000
mean        198.459921
std         290.793238
min           1.000000
25%          34.000000
50%          89.000000
75%         245.000000
max       17112.000000
Name: rating, dtype: float64
```

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

In [10]:
```
quantiles = no_of_rated_movies_per_user.quantil
e(np.arange(0,1.01,0.01), interpolation='highe
r')
```

In [0]:
```
plt.title("Quantiles and their Values")
quantiles.plot()
# quantiles with 0.05 difference
plt.scatter(x=quantiles.index[::5], y=quantiles
.values[::5], c='orange', label="quantiles with
0.05 intervals")
# quantiles with 0.25 difference
plt.scatter(x=quantiles.index[::25], y=quantile
s.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 perce
ntile 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 [11]:
```
quantiles[::5]
```

Out[11]:
```
0.00        1
0.05        7
0.10       15
0.15       21
0.20       27
0.25       34
0.30       41
0.35       50
0.40       60
0.45       73
0.50       89
0.55      109
0.60      133
0.65      163
0.70      199
0.75      245
0.80      307
0.85      392
0.90      520
0.95      749
1.00    17112
Name: rating, dtype: int64
```

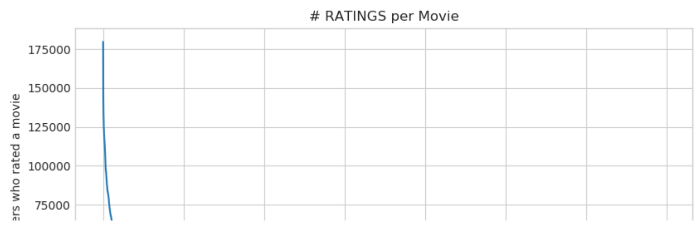
how many ratings at the last 5% of all ratings??

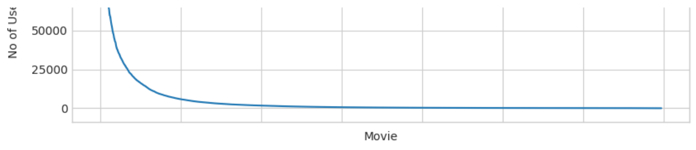
In [12]:
```
print('\n No of ratings at last 5 percentile :
{}\n'.format(sum(no_of_rated_movies_per_user>=
749)) )
```

```
 No of ratings at last 5 percentile : 203
05
```

3.3.4 Analysis of ratings of a movie given by a user

In [0]:
```
no_of_ratings_per_movie = train_df.groupby(by=
'movie')['rating'].count().sort_values(ascendin
g=False)

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

plt.show()
```



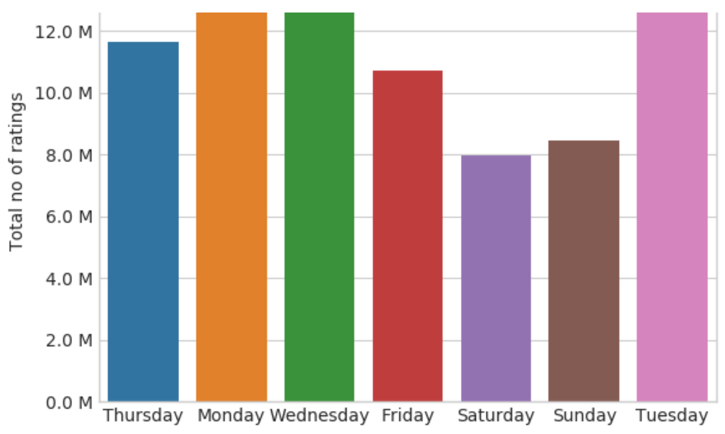


- **It is very skewed.. just like nunmber of ratings given per user.**

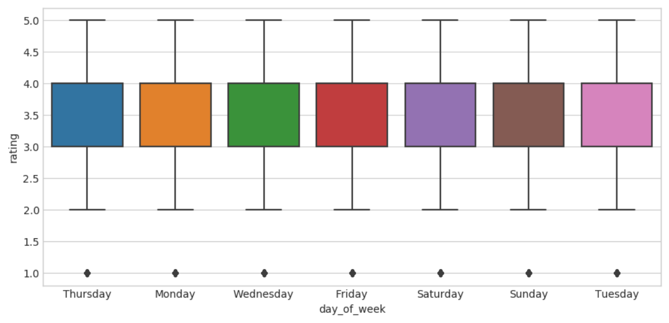
```
- There are some movies (which are very
 popular) which are rated by huge number
of users.

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

3.3.5 Number of ratings on each day of the week

In [0]:
```
fig, ax = plt.subplots()
sns.countplot(x='day_of_week', data=train_df, a
x=ax)
plt.title('No of ratings on each day...')
plt.ylabel('Total no of ratings')
plt.xlabel('')
ax.set_yticklabels([human(item, 'M') for item i
n ax.get_yticks()])
plt.show()
```



In [0]:
```
start = datetime.now()
fig = plt.figure(figsize=plt.figaspect(.45))
sns.boxplot(y='rating', x='day_of_week', data=t
rain_df)
plt.show()
print(datetime.now() - start)
```



```
0:01:10.003761
```

In [13]:
```
avg_week_df = train_df.groupby(by=['day_of_wee
k'])['rating'].mean()
print(" AVerage ratings")
print("-"*30)
print(avg_week_df)
print("\n")
```

```
 AVerage ratings
```

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

3.3.6 Creating sparse matrix from data frame

3.3.6.1 Creating sparse matrix from train data frame

In [14]:

```
start = datetime.now()
if os.path.isfile('train_sparse_matrix.npz'):
    print("It is present in your pwd, getting i
t from disk....")
    # just get it from the disk instead of comp
uting it
    train_sparse_matrix = sparse.load_npz('trai
n_sparse_matrix.npz')
    print("DONE..")
else:
    print("We are creating sparse_matrix from t
he dataframe..")
    # create sparse_matrix and store it for aft
er usage.
    # csr_matrix(data_values, (row_index, col_i
ndex), shape_of_matrix)
    # It should be in such a way that, MATRIX[r
ow, col] = data
    train_sparse_matrix = sparse.csr_matrix((tr
ain_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 usag
e..')
    # save it into disk
    sparse.save_npz("train_sparse_matrix.npz",
train_sparse_matrix)
    print('Done..\n')

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

```
We are creating sparse_matrix from the da
taframe..
Done. It's shape is : (user, movie) :  (2
649430, 17771)
Saving it into disk for furthur usage..
Done..

0:01:00.363404
```

The Sparsity of Train Sparse Matrix

In [15]:

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

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

```
Sparsity Of Train matrix : 99.82927092591
95 %
```

3.3.6.2 Creating sparse matrix from test data frame

In [16]:

```
start = datetime.now()
if os.path.isfile('test_sparse_matrix.npz'):
    print("It is present in your pwd, getting i
t from disk....")
    # just get it from the disk instead of comp
uting it
```

```
    test_sparse_matrix = sparse.load_npz('test_
sparse_matrix.npz')
    print("DONE..")
else:
    print("We are creating sparse_matrix from t
he dataframe..")
    # create sparse_matrix and store it for aft
er usage.
    # csr_matrix(data_values, (row_index, col_i
ndex), shape_of_matrix)
    # It should be in such a way that, MATRIX[r
ow, col] = data
    test_sparse_matrix = sparse.csr_matrix((tes
t_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 usag
e..')
    # save it into disk
    sparse.save_npz("test_sparse_matrix.npz", t
est_sparse_matrix)
    print('Done..\n')

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

```
We are creating sparse_matrix from the da
taframe..
Done. It's shape is : (user, movie) :  (2
649430, 17771)
Saving it into disk for furthur usage..
Done..

0:00:16.291838
```

The Sparsity of Test data Matrix

In [17]:
```
us,mv = test_sparse_matrix.shape
elem = test_sparse_matrix.count_nonzero()

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

```
Sparsity Of Test matrix : 99.957317729886
94 %
```

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

In [12]:
```
# get the user averages in dictionary (key: use
r_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 use
r 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 mat
rix
    u,m = sparse_matrix.shape
    # creae a dictonary of users and their aver
age ratigns..
    average_ratings = { i : sum_of_ratings[i]/n
o_of_ratings[i]
                                 for i in range
(u if of_users else m)
                                    if no_of_ra
tings[i] !=0}

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

3.3.7.1 finding global average of all movie ratings

In [19]:
```
train_averages = dict()
# get the global average of ratings in our trai
n set.
train_global_average = train_sparse_matrix.sum
()/train_sparse_matrix.count_nonzero()
train_averages['global'] = train_global_average
train_averages
```

Out[19]:
```
{'global': 3.582890686321557}
```

3.3.7.2 finding average rating per user

In [20]:
```
train_averages['user'] = get_average_ratings(tr
ain_sparse_matrix, of_users=True)
print('\nAverage rating of user 10 :',train_ave
rages['user'][10])
```

```
Average rating of user 10 : 3.37810945273
63185
```

3.3.7.3 finding average rating per movie

In [21]:

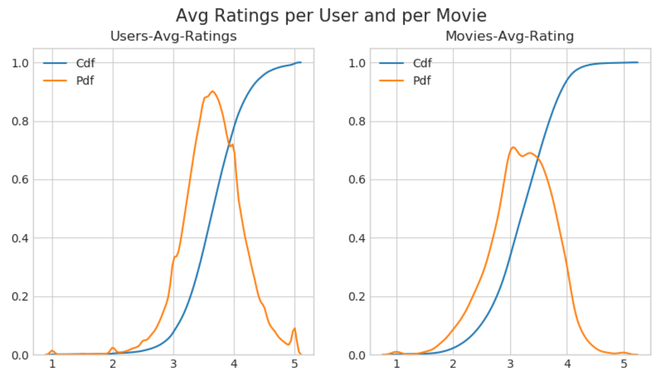
```
train_averages['movie'] =  get_average_ratings(
train_sparse_matrix, of_users=False)
print('\n AVerage rating of movie 15 :',train_a
verages['movie'][15])
```

```
 AVerage rating of movie 15 : 3.303846153
8461537
```

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

In [0]:

```
start = datetime.now()
# draw pdfs for average rating per user and ave
rage
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2
, figsize=plt.figaspect(.5))
fig.suptitle('Avg Ratings per User and per Movi
e', 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), lab
el='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 th
e dictionary..
movie_averages = [rat for rat in train_averages
['movie'].values()]
sns.distplot(movie_averages, ax=ax2, hist=False
,
             kde_kws=dict(cumulative=True), lab
el='Cdf')
sns.distplot(movie_averages, ax=ax2, hist=False
, label='Pdf')

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



```
0:00:35.003443
```

3.3.8 Cold Start problem

3.3.8.1 Cold Start problem with Users

In [0]:
```
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 :', user
s_train)
print("\nNo of Users that didn't appear in trai
n 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 d
ata: 75148(15.65 %)
```

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

3.3.8.2 Cold Start problem with Movies

In [0]:

```
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_movi
es)
print('\nNumber of Users in Train data :', movi
es_train)
print("\nNo of Movies that didn't appear in tra
in data: {}({} %) \n ".format(new_movies,

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

```
Total number of Movies  : 17770

Number of Users in Train data : 17424

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

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

3.4 Computing Similarity matrices

3.4.1 Computing User-User Similarity matrix

1. Calculating User User Similarity_Matrix is **not very easy**(*unless you have huge Computing Power and lots of time*) because of number of. usersbeing lare.

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

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

In [0]:

```
from sklearn.metrics.pairwise import cosine_sim
ilarity



def compute_user_similarity(sparse_matrix, comp
ute_for_few=False, top = 100, verbose=False, ve
rb_for_n_rows = 20,
                            draw_time_taken=Tru
e):
    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 h
ave to
    time_taken = list() #  time taken for findi
ng similar users for an user..

    # we create rows, cols, and data lists.., w
```

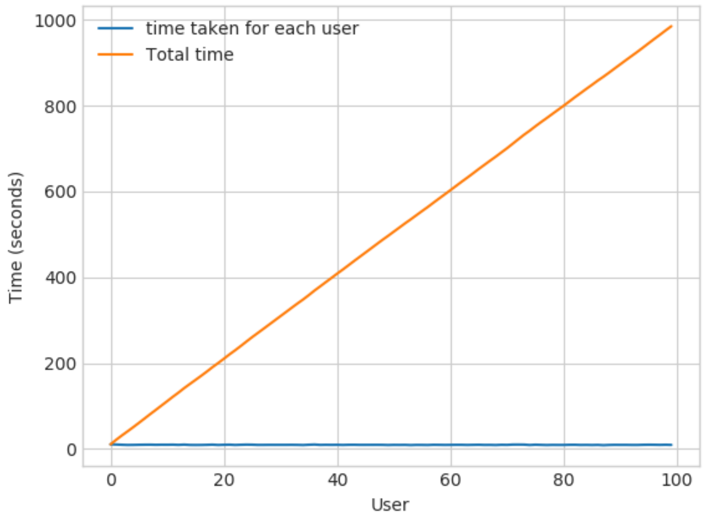
```
hich can be used to create sparse matrices
    rows, cols, data = list(), list(), list()
    if verbose: print("Computing top",top,"simi
larities for each user..")

    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.g
etrow(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().timest
amp() - prev.timestamp())
        if verbose:
            if temp%verb_for_n_rows == 0:
                print("computing done for {} us
ers [  time elapsed : {}  ]"
                      .format(temp, datetime.no
w()-start))
```

```
    # lets create sparse matrix out of these an
d return it
    if verbose: print('Creating Sparse matrix f
rom the computed similarities')
    #return rows, cols, data

    if draw_time_taken:
        plt.plot(time_taken, label = 'time take
n 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_tak
en
```

In [0]:

```
start = datetime.now()
u_u_sim_sparse, _ = compute_user_similarity(tra
in_sparse_matrix, compute_for_few=True, top = 1
00,

verbose=True)
print("-"*100)
print("Time taken :",datetime.now()-start)
```

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



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

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

- We have **405,041 users** in out training set and computing similarities between them..(**17K dimensional vector..**) is time consuming..

- From above plot, It took roughly **8.88 sec** for computing simliar users for **one user**
- We have **405,041 users** with us in training set.
-
    - Even if we run on 4 cores parallelly (a typical system now a days), It will still take almost **10 and 1/2** days.

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

In [0]:

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

start = datetime.now()

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

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

```
0:29:07.069783
```

Here,

- (netflix_svd.**singular_values_**)

- (netflix_svd.**components_**)

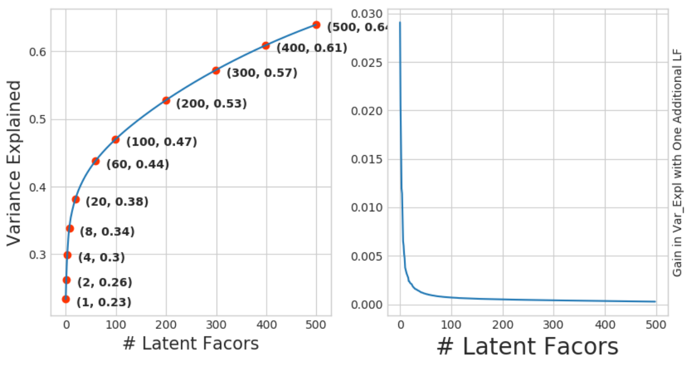
- is not returned. instead **Projection_of_X** onto the new vectorspace is returned.

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

In [0]:
```
expl_var = np.cumsum(netflix_svd.explained_vari
ance_ratio_)
```

In [0]:
```
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2
, figsize=plt.figaspect(.5))

ax1.set_ylabel("Variance Explained", fontsize=1
5)
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_va
r[[i-1 for i in ind]], c='#ff3300')
for i in ind:
    ax1.annotate(s ="({}, {})".format(i,  np.ro
und(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 Addit
ional LF", fontsize=10)
ax2.yaxis.set_label_position("right")
ax2.set_xlabel("# Latent Facors", fontsize=20)

plt.show()
```



In [0]:
```
for i in ind:
    print("({}, {})".format(i, np.round(expl_va
r[i-1], 2)))
```

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


(400, 0.61)
```

```
(500, 0.64)
```

> I think 500 dimensions is good enough

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

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

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

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

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

In [0]:
```
# Let's project our Original U_M matrix into in
to 500 Dimensional space...
```

```
start = datetime.now()
trunc_matrix = train_sparse_matrix.dot(netflix_
svd.components_.T)
print(datetime.now()- start)
```

```
0:00:45.670265
```

In [0]:
```
type(trunc_matrix), trunc_matrix.shape
```

Out[0]:
```
(numpy.ndarray, (2649430, 500))
```

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

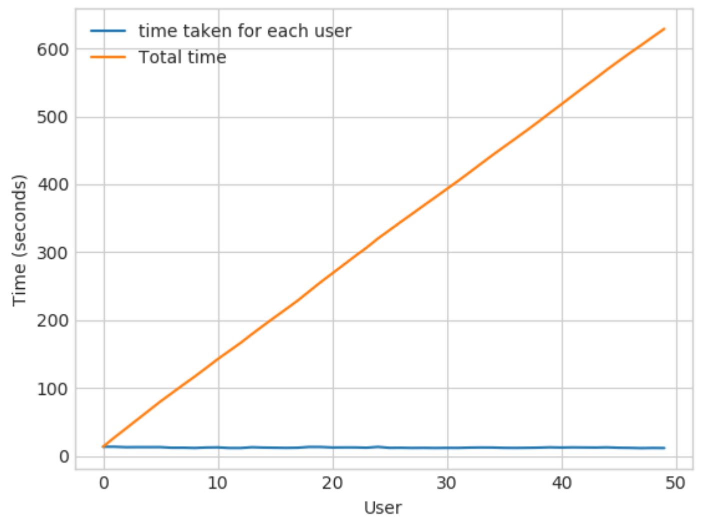
In [0]:
```
if not os.path.isfile('trunc_sparse_matrix.npz'
):
    # create that sparse sparse matrix
    trunc_sparse_matrix = sparse.csr_matrix(tru
nc_matrix)
    # Save this truncated sparse matrix for lat
er usage..
    sparse.save_npz('trunc_sparse_matrix', trun
c_sparse_matrix)
else:
    trunc_sparse_matrix = sparse.load_npz('trun
c_sparse_matrix.npz')
```

In [0]:
```
trunc_sparse_matrix.shape
```

Out[0]:
```
(2649430, 500)
```

In [0]:
```
start = datetime.now()
trunc_u_u_sim_matrix, _ = compute_user_similari
ty(trunc_sparse_matrix, compute_for_few=True, t
op=50, verbose=True,
verb_for_n_rows=10)
print("-"*50)
print("time:",datetime.now()-start)
```

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



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

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

- from above plot, It took almost **12.18** for computing simlilar users for **one user**
- We have **405041 users** with us in training set.
-
    - 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 dif
ficult.
```

--------------------------------*(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 comput
ed 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 t
hat we can just access it(similar users)
without recomputing it again.
    -
- ***If It is already Computed***:
    - Just get it directly from our data
structure, which has that information.
    - In production time, We might have
 to recompute similarities, if it is com
puted a long time ago. Because user pref
erences changes over time. If we could m
aintain some kind of Timer, which when e
xpires, we have to update it ( recompute
it ).
    -
- ***Which datastructure to use:***
    - It is purely implementation depend
ant.
    - One simple method is to maintain a
**Dictionary Of Dictionaries**.
        -
        - **key    :** _userid_
        - __value__: _Again a dictionary
_
            - __key__  : _Similar User_
            - __value__: _Similarity Val
ue_
```

3.4.2 Computing Movie-Movie Similarity matrix

In [0]:

```
start = datetime.now()
if not os.path.isfile('m_m_sim_sparse.npz'):
    print("It seems you don't have that file. C
omputing 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 u
sing it. For future purposes.
    print("Saving it to disk without the need o
f re-computing it again.. ")
    sparse.save_npz("m_m_sim_sparse.npz", m_m_s
im_sparse)
    print("Done..")
else:
    print("It is there, We will get it.")
    m_m_sim_sparse = sparse.load_npz("m_m_sim_s
parse.npz")
    print("Done ...")

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

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

```
It seems you don't have that file. Comput
ing movie_movie similarity...

Done..
Saving it to disk without the need of re-
computing it again..
Done..
It's a  (17771, 17771)  dimensional matri
x
0:10:02.736054
```

In [0]:
```
m_m_sim_sparse.shape
```
Out[0]:
```
(17771, 17771)
```

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

In [0]:
```
movie_ids = np.unique(m_m_sim_sparse.nonzero()[
1])
```

In [0]:
```
start = datetime.now()
similar_movies = dict()
for movie in movie_ids:
    # get the top similar movies and store them
in the dictionary
    sim_movies = m_m_sim_sparse[movie].toarray
().ravel().argsort()[::-1][1:]
    similar_movies[movie] = sim_movies[:100]
print(datetime.now() - start)

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

```
0:00:33.411700
```

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

3.4.3 Finding most similar movies using similarity matrix

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

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

In [0]:

```
# First Let's load the movie details into soe d
ataframe..
# movie details are in 'netflix/movie_titles.cs
```

```
v'

movie_titles = pd.read_csv("data_folder/movie_t
itles.csv", sep=',', header = None,
                           names=['movie_id',
'year_of_release', 'title'], verbose=True,
                      index_col = 'movie_id', e
ncoding = "ISO-8859-1")

movie_titles.head()
```

```
Tokenization took: 4.50 ms
Type conversion took: 165.72 ms
Parser memory cleanup took: 0.01 ms
```

Out[0]:

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

Similar Movies for 'Vampire Journals'

In [0]:

```
mv_id = 67

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

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

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

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

It has 270 Ratings from users.

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

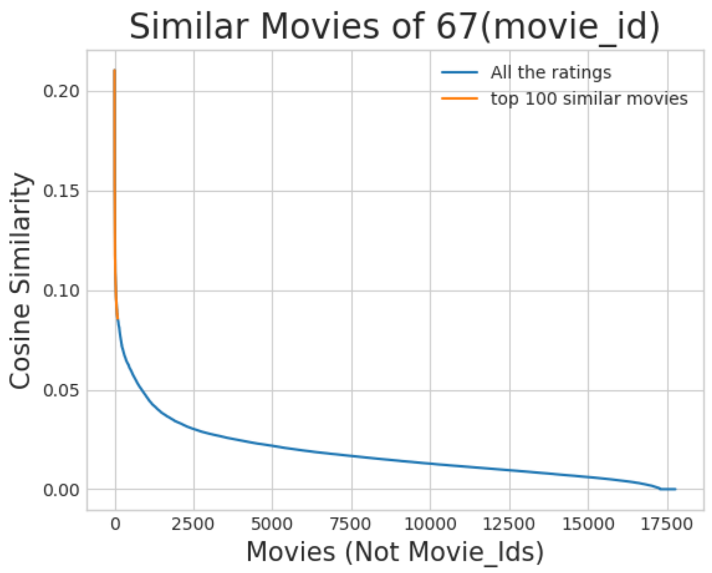
In [0]:

```
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 [0]:

```
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)".form
at(mv_id), fontsize=20)
plt.xlabel("Movies (Not Movie_Ids)", fontsize=1
5)
```

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



Top 10 similar movies

In [0]:
```
movie_titles.loc[sim_indices[:10]]
```

Out[0]:

	year_of_release	title
movie_id		
323	1999.0	Modern Vampires
4044	1998.0	Subspecies 4: Bloodstorm
1688	1993.0	To Sleep With a Vampire
13962	2001.0	Dracula: The Dark Prince
12053	1993.0	Dracula Rising

	year_of_release	title
movie_id		
16279	2002.0	Vampires: Los Muertos
4667	1996.0	Vampirella
1900	1997.0	Club Vampire
13873	2001.0	The Breed
15867	2003.0	Dracula II: Ascension

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

4. Machine Learning Models

In [4]:

```
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 sp
arse_matrix...
    row_ind, col_ind, ratings = sparse.find(spa
rse_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".f
ormat(len(ratings)))

    # It just to make sure to get same sample e
verytime we run this program..
    # and pick without replacement....
    np.random.seed(15)
    sample_users = np.random.choice(users, no_u
sers, replace=False)
    sample_movies = np.random.choice(movies, no
_movies, replace=False)
    # get the boolean mask or these sampled_ite
ms in originl row/col_inds..
    mask = np.logical_and( np.isin(row_ind, sam
ple_users),
                      np.isin(col_ind, sample_m
ovies) )

    sample_sparse_matrix = sparse.csr_matrix((r
atings[mask], (row_ind[mask], col_ind[mask])),
                                             sh
ape=(max(sample_users)+1, max(sample_movies)+1
))

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

    print('Saving it into disk for furthur usag
e..')
    # 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 [7]:
```
start = datetime.now()
path = "sample_train_sparse_matrix.npz" #sampl
e/small/
if os.path.isfile(path):
    print("It is present in your pwd, getting i
t from disk....")
    # just get it from the disk instead of comp
uting it
    sample_train_sparse_matrix = sparse.load_np
z(path)
    print("DONE..")
else:
    # get 10k users and 1k movies from availabl
e data
    sample_train_sparse_matrix = get_sample_spa
rse_matrix(train_sparse_matrix, no_users=25000,
no_movies=3000,
                                              pa
th = path)

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

```
It is present in your pwd, getting it fro
m disk....
DONE..
0:00:00.036747
```

4.1.2 Build sample test data from the test data

In [8]:
```
start = datetime.now()

path = "sample_test_sparse_matrix.npz" #sample/
small/
if os.path.isfile(path):
    print("It is present in your pwd, getting i
t from disk....")
    # just get it from the disk instead of comp
uting it
    sample_test_sparse_matrix = sparse.load_npz
(path)
    print("DONE..")
else:
    # get 5k users and 500 movies from availabl
e data
    sample_test_sparse_matrix = get_sample_spar
se_matrix(test_sparse_matrix, no_users=5000, no
_movies=500,

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

```
It is present in your pwd, getting it fro
m disk....
DONE..
0:00:00.029070
```

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

In [9]:
```
sample_train_averages = dict()
```

4.2.1 Finding Global Average of all movie ratings

In [10]:
```
# get the global average of ratings in our trai
n set.
global_average = sample_train_sparse_matrix.sum
()/sample_train_sparse_matrix.count_nonzero()
sample_train_averages['global'] = global_averag
e
sample_train_averages
```

Out[10]:
```
{'global': 3.581679377504138}
```

4.2.2 Finding Average rating per User

In [13]:
```
sample_train_averages['user'] = get_average_rat
ings(sample_train_sparse_matrix, of_users=True)
print('\nAverage rating of user 1515220 :',samp
le_train_averages['user'][1515220])
```

```
Average rating of user 1515220 : 3.965517
2413793105
```

4.2.3 Finding Average rating per Movie

In [14]:
```
sample_train_averages['movie'] =  get_average_r
atings(sample_train_sparse_matrix, of_users=Fal
se)
```

```
print('\n AVerage rating of movie 15153 :',samp
le_train_averages['movie'][15153])
```

```
 AVerage rating of movie 15153 : 2.645833
3333333335
```

4.3 Featurizing data

In [15]:
```
print('\n No of ratings in Our Sampled train ma
trix is : {}\n'.format(sample_train_sparse_matr
ix.count_nonzero()))
print('\n No of ratings in Our Sampled test  ma
trix is : {}\n'.format(sample_test_sparse_matri
x.count_nonzero()))
```

```
 No of ratings in Our Sampled train matri
x is : 129286


 No of ratings in Our Sampled test  matri
x is : 7333
```

4.3.1 Featurizing data for regression problem

4.3.1.1 Featurizing train data

In [16]:
```
# get users, movies and ratings from our sample
s train sparse matrix
```

```
sample_train_users, sample_train_movies, sample
_train_ratings = sparse.find(sample_train_spars
e_matrix)
```

In [0]:

```
################################################
#############
# It took me almost 10 hours to prepare this tr
ain dataset.#
################################################
#############
start = datetime.now()
if os.path.isfile('sample/small/reg_train.csv'
):
    print("File already exists you don't have t
o prepare again..." )
else:
    print('preparing {} tuples for the datase
t..\n'.format(len(sample_train_ratings)))
    with open('sample/small/reg_train.csv', mod
e='w') as reg_data_file:
        count = 0
        for (user, movie, rating)  in zip(sampl
e_train_users, sample_train_movies, sample_trai
n_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 it
s similar users.
            # get the ratings of most similar u
sers for this movie
            top_ratings = sample_train_sparse_m
atrix[top_sim_users, movie].toarray().ravel()
            # we will make it's length "5" by a
dding movie averages to .
            top_sim_users_ratings = list(top_ra
tings[top_ratings != 0][:5])
            top_sim_users_ratings.extend([sampl
e_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(sampl
e_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 m
ovie rated by this user..
            top_ratings = sample_train_sparse_m
atrix[user, top_sim_movies].toarray().ravel()
            # we will make it's length "5" by a
dding user averages to.
            top_sim_movies_ratings = list(top_r
atings[top_ratings != 0][:5])
```

```
            top_sim_movies_ratings.extend([samp
le_train_averages['user'][user]]*(5-len(top_sim
_movies_ratings)))
        #     print(top_sim_movies_ratings, end
=" : -- ")

            #-----------------prepare the row t
o be stores in a file-----------------#
            row = list()
            row.append(user)
            row.append(movie)
            # Now add the other features to thi
s data...
            row.append(sample_train_averages['g
lobal']) # first feature
            # next 5 features are similar_users
"movie" ratings
            row.extend(top_sim_users_ratings)
            # next 5 features are "user" rating
s for similar_movies
            row.extend(top_sim_movies_ratings)
            # Avg_user rating
            row.append(sample_train_averages['u
ser'][user])
            # Avg_movie rating
            row.append(sample_train_averages['m
ovie'][movie])

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

            # add rows to the file opened..
            reg_data_file.write(','.join(map(st
r, row)))
```

```
                reg_data_file.write('\n')
                if (count)%10000 == 0:
                    # print(','.join(map(str, ro
w)))
                    print("Done for {} rows----- {}
".format(count, datetime.now() - start))


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

```
preparing 129286 tuples for the dataset..

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

Reading from the file to make a Train_dataframe

In [19]:
```
#sample/small/
reg_train = pd.read_csv('reg_train.csv', names
= ['user', 'movie', 'GAvg', 'sur1', 'sur2', 'su
r3', 'sur4', 'sur5','smr1', 'smr2', 'smr3', 'sm
r4', 'smr5', 'UAvg', 'MAvg', 'rating'], header=
None)
```

```
print(reg_train.shape)
reg_train.head()
```

```
(129286, 16)
```

Out[19]:

	user	movie	GAvg	sur1	sur2	sur3	sur4
0	53406	33	3.581679	4.0	5.0	5.0	4.0
1	99540	33	3.581679	5.0	5.0	5.0	4.0
2	99865	33	3.581679	5.0	5.0	4.0	5.0
3	101620	33	3.581679	2.0	3.0	5.0	5.0
4	112974	33	3.581679	5.0	5.0	5.0	5.0

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

4.3.1.2 Featurizing test data

In [20]:
```
# get users, movies and ratings from the Sample
d Test
sample_test_users, sample_test_movies, sample_t
est_ratings = sparse.find(sample_test_sparse_ma
trix)
```

In [21]:
```
sample_train_averages['global']
```

Out[21]:
```
3.581679377504138
```

In [0]:
```
start = datetime.now()

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

    print('preparing {} tuples for the datase
t..\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(sampl
e_test_users, sample_test_movies, sample_test_r
atings):
            st = datetime.now()

        #--------------------- Ratings of "movi
e" by similar users of "user" ------------------
----
            #print(user, movie)
            try:
                # compute the similar Users of
 the "user"
                user_sim = cosine_similarity(sa
mple_train_sparse_matrix[user], sample_train_sp
arse_matrix).ravel()
                top_sim_users = user_sim.argsor
t()[::-1][1:] # we are ignoring 'The User' from
its similar users.
                # get the ratings of most simil
ar users for this movie
                top_ratings = sample_train_spar
se_matrix[top_sim_users, movie].toarray().ravel
()
                # we will make it's length "5"
 by adding movie averages to .
                top_sim_users_ratings = list(to
p_ratings[top_ratings != 0][:5])
                top_sim_users_ratings.extend([s
ample_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([s
ample_train_averages['global']]*(5 - len(top_si
m_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 t
o be stores in a file------------------#
            row = list()
            # add usser and movie name first
            row.append(user)
            row.append(movie)
            row.append(sample_train_averages['g
lobal']) # 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" rating
s for similar_movies
            row.extend(top_sim_movies_ratings)
            #print(row)
            # Avg_user rating
            try:
                row.append(sample_train_average
s['user'][user])
            except KeyError:
                row.append(sample_train_average
s['global'])
            except:
                raise
            #print(row)
            # Avg_movie rating
            try:
                row.append(sample_train_average
s['movie'][movie])
            except KeyError:
                row.append(sample_train_average
s['global'])
            except:
                raise
            #print(row)
            # finalley, The actual Rating of th
is user-movie pair...
            row.append(rating)
            #print(row)
            count = count + 1

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

```
preparing 7333 tuples for the dataset..

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

Reading from the file to make a test dataframe

In [23]:

```
#sample/small/
reg_test_df = pd.read_csv('reg_test.csv', names
= ['user', 'movie', 'GAvg', 'sur1', 'sur2', 'su
r3', 'sur4', 'sur5',
'smr1', 'smr2', 'smr3', 'smr4', 'smr5',

'UAvg', 'MAvg', 'rating'], header=None)
print(reg_test_df.shape)
reg_test_df.head(4)
```

```
(7333, 16)
```

Out[23]:

	user	movie	GAvg	sur1	sur2	
0	808635	71	3.581679	3.581679	3.581679	3
1	941866	71	3.581679	3.581679	3.581679	3
2	1737912	71	3.581679	3.581679	3.581679	3
3	1849204	71	3.581679	3.581679	3.581679	3

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

4.3.2 Transforming data for Surprise models

In [26]:
```python
from surprise import Reader, Dataset
```

4.3.2.1 Transforming train data

- We can't give raw data (movie, user, rating) to train the model in Surprise library.
- They have a saperate format for TRAIN and TEST data, which will be useful for training the models like SVD, KNNBaseLineOnly....etc..,in Surprise.
- We can form the trainset from a file, or from a Pandas DataFrame. http://surprise.readthedocs.io/en/stable/getting_s
dom-dataframe-py

In [27]:
```python
# It is to specify how to read the dataframe.
# for our dataframe, we don't have to specify a
nything extra..
reader = Reader(rating_scale=(1,5))

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

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

4.3.2.2 Transforming test data

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

In [28]:
```
testset = list(zip(reg_test_df.user.values, reg
_test_df.movie.values, reg_test_df.rating.value
s))
testset[:3]
```

Out[28]:
```
[(808635, 71, 5), (941866, 71, 4), (17379
12, 71, 3)]
```

4.4 Applying Machine Learning models

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

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

In [29]:
```
models_evaluation_train = dict()
models_evaluation_test = dict()

models_evaluation_train, models_evaluation_test
```

Out[29]:
```
({}, {})
```

Utility functions for running regression models

In [30]:
```
# to get rmse and mape given actual and predicted ratings..
def get_error_metrics(y_true, y_pred):
    rmse = np.sqrt(np.mean([ (y_true[i] - y_pred[i])**2 for i in range(len(y_pred)) ]))
    mape = np.mean(np.abs( (y_true - y_pred)/y_true )) * 100
    return rmse, mape

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

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


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

    # from the trained model, get the predictio
ns....
    print('Evaluating the model with TRAIN dat
a...')
    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 dictio
nary..
    train_results = {'rmse': rmse_train,
                    'mape' : mape_train,
                    'predictions' : y_train_pre
d}

    #######################################
    # 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 dictionar
y.
    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 [ ]:
```
def get_ratings(predictions):
    actual = np.array([pred.r_ui for pred in pr
edictions])
    pred = np.array([pred.est for pred in predi
ctions])

    return actual, pred
```

In [ ]:
```
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
```

In [31]:
```
# it is just to makesure that all of our algori
thms should produce same results
# everytime they run...

my_seed = 15
```

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

def run_surprise(algo, trainset, testset, verbo
se=True):
    '''
        return train_dict, test_dict

        It returns two dictionaries, one for tr
ain and the other is for test
        Each of them have 3 key-value pairs, wh
ich specify ''rmse'', ''mape'', and ''predicted
ratings''.
    '''
    start = datetime.now()
    # dictionaries that stores metrics for trai
n 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(dat
etime.now()-st))

    # ---------------- Evaluating train data---
-----------------#
    st = datetime.now()
    print('Evaluating the model with train dat
a..')
    # get the train predictions (list of predic
tion class inside Surprise)
    train_preds = algo.test(trainset.build_test
set())
```

```
    # get predicted ratings from the train pred
ictions..
    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_p
reds)
    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 dict
ionary..')
    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 c
lasses) of test data
    test_preds = algo.test(testset)
    # get the predicted ratings from the list o
f predictions
    test_actual_ratings, test_pred_ratings = ge
```

```
t_ratings(test_preds)
    # get error metrics from the predicted and
 actual ratings
    test_rmse, test_mape = get_errors(test_pred
s)
    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 algorit
hm :', datetime.now() - start)

    # return two dictionaries train and test
    return train, test
```

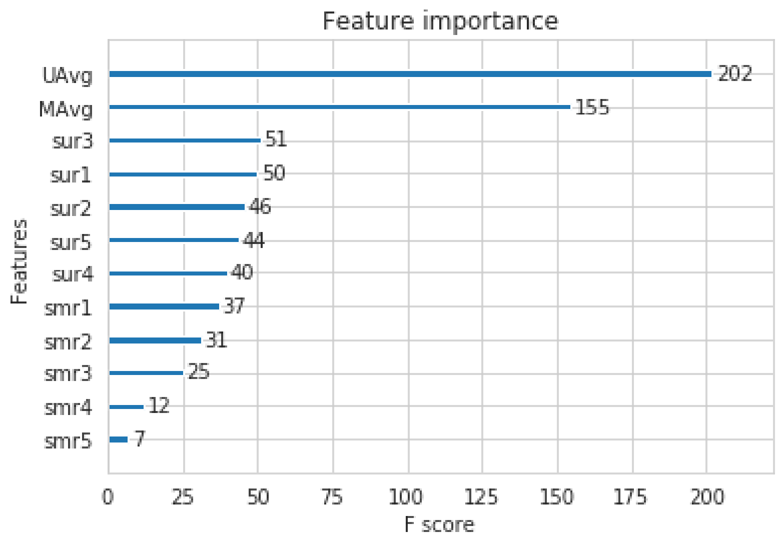
4.4.1 XGBoost with initial 13 features

In [34]:
```
%matplotlib inline
import xgboost as xgb
```

In [35]:
```
# prepare Train data
x_train = reg_train.drop(['user','movie','ratin
g'], axis=1)
y_train = reg_train['rating']

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

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

# store the results in models_evaluations dicti
onaries
models_evaluation_train['first_algo'] = train_r
esults
models_evaluation_test['first_algo'] = test_res
ults

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

```
Training the model..
[15:02:45] WARNING: /workspace/src/object
ive/regression_obj.cu:152: reg:linear is
now deprecated in favor of reg:squarederr
or.
Done. Time taken : 0:00:02.316872

Done
```

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

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



4.4.2 Suprise BaselineModel

In [36]:
```python
from surprise import BaselineOnly
```

Predicted_rating : (baseline prediction)

```
- http://surprise.readthedocs.io/en/sta
ble/basic_algorithms.html#surprise.predi
ction_algorithms.baseline_only.BaselineO
nly
```

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

Optimization function (Least Squares Problem)

```
- http://surprise.readthedocs.io/en/stab
le/prediction_algorithms.html#baselines-
estimates-configuration
```

In [38]:

```
# options are to specify.., how to compute thos
e user and item biases
bsl_options = {'method': 'sgd',
               'learning_rate': 0.001
               }
bsl_algo = BaselineOnly(bsl_options=bsl_options
)
# run this algorithm.., It will return the trai
n and test results..
bsl_train_results, bsl_test_results = run_surpr
ise(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_r
esults
```

```
Training the model...
Estimating biases using sgd...
Done. time taken : 0:00:00.514993

Evaluating the model with train data..
time taken : 0:00:01.192722
---------------
Train Data
---------------
RMSE : 0.9347153928678286

MAPE : 29.389572652358183

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.071277
---------------
Test Data
---------------
RMSE : 1.0730330260516174

MAPE : 35.04995544572911

storing the test results in test dictiona
ry...

-----------------------------------------
----
Total time taken to run this algorithm :
0:00:01.779784
```

4.4.3 XGBoost with initial 13 features + Surprise Baseline predictor

Updating Train Data

In [39]:
```
# add our baseline_predicted value as our feature..
reg_train['bslpr'] = models_evaluation_train['bsl_algo']['predictions']
reg_train.head(2)
```

Out[39]:

	user	movie	GAvg	sur1	sur2	sur3	sur4
0	53406	33	3.581679	4.0	5.0	5.0	4.0
1	99540	33	3.581679	5.0	5.0	5.0	4.0

Updating Test Data

In [40]:
```
# add that baseline predicted ratings with Surprise to the test data as well
reg_test_df['bslpr']  = models_evaluation_test['bsl_algo']['predictions']

reg_test_df.head(2)
```

Out[40]:

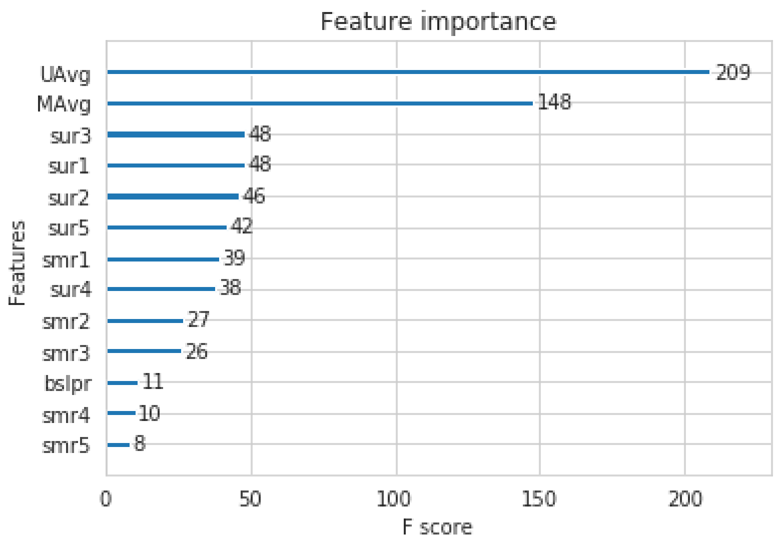
	user	movie	GAvg	sur1	sur2	
0	808635	71	3.581679	3.581679	3.581679	3.!
1	941866	71	3.581679	3.581679	3.581679	3.!

In [41]:
```
# prepare train data
```

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

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

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

# store the results in models_evaluations dicti
onaries
models_evaluation_train['xgb_bsl'] = train_resu
lts
models_evaluation_test['xgb_bsl'] = test_result
s

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

```
Training the model..
[15:07:40] WARNING: /workspace/src/object
ive/regression_obj.cu:152: reg:linear is
now deprecated in favor of reg:squarederr
or.
Done. Time taken : 0:00:02.916295

Done

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

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



4.4.4 Surprise KNNBaseline predictor

In [42]:
```python
from surprise import KNNBaseline
```

- KNN BASELINE
  - http://surprise.readthedocs.io/en/stable/knn

- PEARSON_BASELINE SIMILARITY
  - http://surprise.readthedocs.io/en/stable/sim

- 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***)

- - *Baseline prediction* of (user,movie) rating
- - 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):

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

4.4.4.1 Surprise KNNBaseline with user user similarities

In [75]:
```
# we specify , how to compute similarities and
 what to consider with sim_options to our algor
ithm
sim_options = {'user_based' : True,
               'name': 'pearson_baseline',
               'shrinkage': 100,
               'min_support': 2
              }
# we keep other parameters like regularization
 parameter and learning_rate as default values.
bsl_options = {'method': 'sgd'}

knn_bsl_u = KNNBaseline(k=40, sim_options = sim
_options, bsl_options = bsl_options)
knn_bsl_u_train_results, knn_bsl_u_test_results
= run_surprise(knn_bsl_u, trainset, testset, ve
rbose=True)

# Just store these error metrics in our models_
evaluation datastructure
models_evaluation_train['knn_bsl_u'] = knn_bsl_
u_train_results
models_evaluation_test['knn_bsl_u'] = knn_bsl_u
_test_results
```

```
Training the model...
Estimating biases using sgd...
Computing the pearson_baseline similarity
```

```
matrix...
Done computing similarity matrix.
Done. time taken : 0:00:35.326122

Evaluating the model with train data..
time taken : 0:01:54.011719
---------------
Train Data
---------------
RMSE : 0.33642097416508826

MAPE : 9.145093375416348

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.076766
---------------
Test Data
---------------
RMSE : 1.0726493739667242

MAPE : 35.02094499698424

storing the test results in test dictiona
ry...

-----------------------------------------
----
Total time taken to run this algorithm :
0:02:29.415862
```

4.4.4.2 Surprise KNNBaseline with movie movie similarities

In [76]:
```
# we specify , how to compute similarities and
 what to consider with sim_options to our algor
ithm

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

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


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

knn_bsl_m_train_results, knn_bsl_m_test_results
= run_surprise(knn_bsl_m, trainset, testset, ve
rbose=True)

# Just store these error metrics in our models_
evaluation datastructure
models_evaluation_train['knn_bsl_m'] = knn_bsl_
m_train_results
models_evaluation_test['knn_bsl_m'] = knn_bsl_m
_test_results
```

```
Training the model...
Estimating biases using sgd...
Computing the pearson_baseline similarity
matrix...
Done computing similarity matrix.
```

```
Done. time taken : 0:00:00.779623

Evaluating the model with train data..
time taken : 0:00:10.610246
---------------
Train Data
---------------
RMSE : 0.32584796251610554

MAPE : 8.447062581998374

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.073214
---------------
Test Data
---------------
RMSE : 1.072758832653683

MAPE : 35.02269653015042


storing the test results in test dictiona
ry...

-----------------------------------------
----
Total time taken to run this algorithm :
0:00:11.463743
```

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 [77]:
```
# add the predicted values from both knns to th
is dataframe
reg_train['knn_bsl_u'] = models_evaluation_trai
n['knn_bsl_u']['predictions']
reg_train['knn_bsl_m'] = models_evaluation_trai
n['knn_bsl_m']['predictions']

reg_train.head(2)
```

Out[77]:

	user	movie	GAvg	sur1	sur2	sur3	sur4
0	53406	33	3.581679	4.0	5.0	5.0	4.0
1	99540	33	3.581679	5.0	5.0	5.0	4.0

Preparing Test data

In [78]:

```
reg_test_df['knn_bsl_u'] = models_evaluation_te
st['knn_bsl_u']['predictions']
reg_test_df['knn_bsl_m'] = models_evaluation_te
st['knn_bsl_m']['predictions']

reg_test_df.head(2)
```

Out[78]:

	user	movie	GAvg	sur1	sur2	
0	808635	71	3.581679	3.581679	3.581679	3.5
1	941866	71	3.581679	3.581679	3.581679	3.5

In [84]:

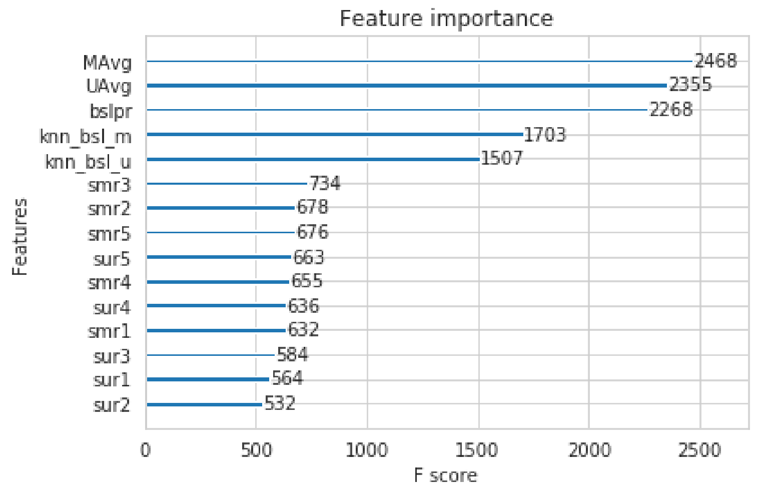
```
from sklearn.model_selection import RandomizedS
earchCV
from scipy import stats
from scipy.stats import randint as sp_randint
```

In [85]:

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

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

params = {'learning_rate' :stats.uniform(0.01,
0.2),
            'n_estimators':[5, 10, 50, 100, 200
, 500, 1000],
            'max_depth':[2, 3, 4, 5, 6, 7, 8, 9
, 10]
            'reg_alpha':sp_randint(0,200),
             'reg_lambda':stats.uniform(0,200)}



# Declare  XGBoost model...
xgbreg = xgb.XGBRegressor(silent=True, n_jobs=-
1, random_state=15)
start =datetime.now()
print('Tuning parameters: \n')
xgb_best = RandomizedSearchCV(xgbreg, param_dis
tributions= params,refit=False, scoring = "neg_
mean_squared_error",n_jobs=-1,
                              cv = 3)
xgb_best.fit(x_train, y_train)
best_para = xgb_best.best_params_

xgb_knn_bsl = xgbreg.set_params(**best_para)
print('Time taken to tune:{}\n'.format(datetime
.now()-start))

train_results, test_results = run_xgboost(xgb_k
nn_bsl, x_train, y_train, x_test, y_test)

# store the results in models_evaluations dicti
onaries
models_evaluation_train['xgb_knn_bsl'] = train_
results
models_evaluation_test['xgb_knn_bsl'] = test_re
sults



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

```
Tuning parameters:
```

```
Time taken to tune:0:03:54.073465

Training the model..
Done. Time taken : 0:00:18.214044

Done

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

TEST DATA
------------------------------
RMSE :  1.0882423800960463
MAPE :  33.80986432892837
```



4.4.6 Matrix Factorization Techniques

4.4.6.1 SVD Matrix Factorization User Movie intractions

In [86]:
```
from surprise import SVD
```

http://surprise.readthedocs.io/en/stable/matrix_factor

- Predicted Rating :

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

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

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

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

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

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

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

In [87]:
```
# initiallize the model
svd = SVD(n_factors=100, biased=True, random_st
ate=15, verbose=True)
svd_train_results, svd_test_results = run_surpr
ise(svd, trainset, testset, verbose=True)
```

```
# Just store these error metrics in our models_
evaluation datastructure
models_evaluation_train['svd'] = svd_train_resu
lts
models_evaluation_test['svd'] = svd_test_result
s
```

```
Training the model...
Processing epoch 0
Processing epoch 1
Processing epoch 2
Processing epoch 3
Processing epoch 4
Processing epoch 5
Processing epoch 6
Processing epoch 7
Processing epoch 8
Processing epoch 9
Processing epoch 10
Processing epoch 11
Processing epoch 12
Processing epoch 13
Processing epoch 14
Processing epoch 15
Processing epoch 16
Processing epoch 17
Processing epoch 18
Processing epoch 19
Done. time taken : 0:00:07.722170

Evaluating the model with train data..
time taken : 0:00:01.479347
---------------
Train Data
---------------
```

```
RMSE : 0.6574721240954099

MAPE : 19.704901088660474

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.070269
---------------
Test Data
---------------
RMSE : 1.0726046873826458

MAPE : 35.01953535988152

storing the test results in test dictiona
ry...

-----------------------------------------
----
Total time taken to run this algorithm :
0:00:09.272654
```

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

In [88]:
```python
from surprise import SVDpp
```

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

- Predicted Rating :

```
- $ \large \hat{r}_{ui} = \mu + b_u + b_
i + q_i^T\left(p_u +
|I_u|^{-\frac{1}{2}} \sum_{j \in I_u}y_j
\right) $
```

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

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

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

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

In [89]:

```
# initiallize the model
svdpp = SVDpp(n_factors=50, random_state=15, ve
rbose=True)
svdpp_train_results, svdpp_test_results = run_s
urprise(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_re
sults
```

```
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 : 0:02:11.309397

Evaluating the model with train data..
time taken : 0:00:07.507721
---------------
Train Data
---------------
RMSE : 0.6032438403305899



MAPE : 17.49285063490268
```

```
adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.073415
---------------
Test Data
---------------
RMSE : 1.0728491944183447

MAPE : 35.03817913919887

storing the test results in test dictiona
ry...

-----------------------------------------
----
Total time taken to run this algorithm :
0:02:18.892001
```

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

Preparing Train data

In [90]:

```
# add the predicted values from both knns to th
is dataframe
reg_train['svd'] = models_evaluation_train['sv
d']['predictions']
reg_train['svdpp'] = models_evaluation_train['s
vdpp']['predictions']
```

```
print(reg_train.shape)
reg_train.head(2)
```

```
(129286, 21)
```

Out[90]:

	user	movie	GAvg	sur1	sur2	sur3	sur4
0	53406	33	3.581679	4.0	5.0	5.0	4.0
1	99540	33	3.581679	5.0	5.0	5.0	4.0

2 rows × 21 columns

Preparing Test data

In [91]:

```
reg_test_df['svd'] = models_evaluation_test['sv
d']['predictions']
reg_test_df['svdpp'] = models_evaluation_test[
'svdpp']['predictions']
print(reg_test_df.shape)
reg_test_df.head(2)
```

```
(7333, 21)
```

Out[91]:

	user	movie	GAvg	sur1	sur2	
0	808635	71	3.581679	3.581679	3.581679	3.5
1	941866	71	3.581679	3.581679	3.581679	3.5

2 rows × 21 columns

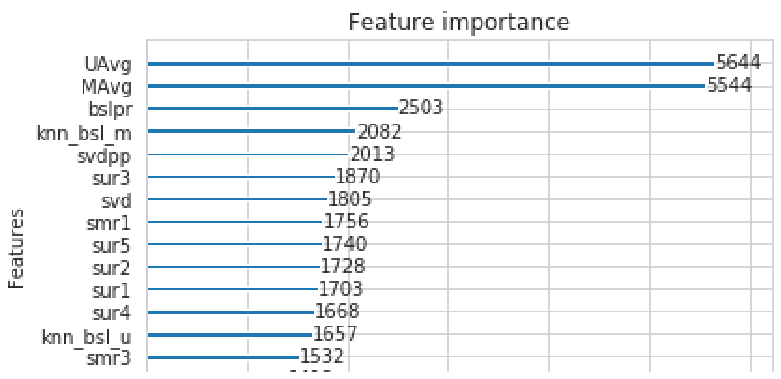
In [92]:

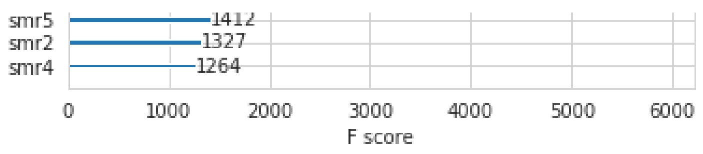
```
# prepare x_train and y_train
x_train = reg_train.drop(['user', 'movie', 'rat
ing',], axis=1)
```

```
y_train = reg_train['rating']

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

params = {'learning_rate' :stats.uniform(0.01,0.2),
            'n_estimators':[5, 10, 50, 100, 200, 500, 1000],
            'max_depth':[2, 3, 4, 5, 6, 7, 8, 9, 10]
            'reg_alpha':sp_randint(0,200),
            'reg_lambda':stats.uniform(0,200)}


# Declare  XGBoost model...
xgbreg = xgb.XGBRegressor(silent=True, n_jobs=-1, random_state=15)
start =datetime.now()
print('Tuning parameters: \n')
xgb_best = RandomizedSearchCV(xgbreg, param_distributions= params,refit=False, scoring = "neg_mean_squared_error",n_jobs=-1,
                              cv = 3)
xgb_best.fit(x_train, y_train)
best_para = xgb_best.best_params_

xgb_final = xgbreg.set_params(**best_para)
print('Time taken to tune:{}\n'.format(datetime.now()-start))

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

```
# store the results in models_evaluations dicti
onaries
models_evaluation_train['xgb_final'] = train_re
sults
models_evaluation_test['xgb_final'] = test_resu
lts


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

```
Tuning parameters:

Time taken to tune:0:09:51.792877

Training the model..
Done. Time taken : 0:00:56.157929

Done


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

TEST DATA
------------------------------
RMSE :  1.08177621710006
MAPE :  34.12278618258321
```



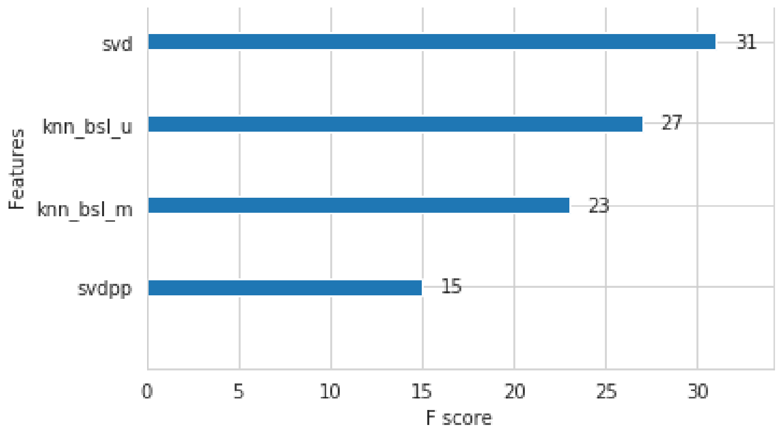


4.4.8 XgBoost with Surprise Baseline + Surprise KNNbaseline + MF Techniques

In [94]:

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

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



params = {'learning_rate' :stats.uniform(0.01,
0.2),
            'n_estimators':[5, 10, 50, 100, 200
, 500, 1000],
            'max_depth':[2, 3, 4, 5, 6, 7, 8, 9
, 10]
            'reg_alpha':sp_randint(0,200),
            'reg_lambda':stats.uniform(0,200)}

# Declare  XGBoost model...
xgbreg = xgb.XGBRegressor(silent=True, n_jobs=-
1, random_state=15)
start =datetime.now()
print('Tuning parameters: \n')
xgb_best = RandomizedSearchCV(xgbreg, param_dis
tributions= params,refit=False, scoring = "neg_
mean_squared_error",n_jobs=-1,
                              cv = 3)
xgb_best.fit(x_train, y_train)
best_para = xgb_best.best_params_

xgb_all_models = xgbreg.set_params(**best_para)
train_results, test_results = run_xgboost(xgb_a
ll_models, x_train, y_train, x_test, y_test)

# store the results in models_evaluations dicti
onaries
models_evaluation_train['xgb_all_models'] = tra
in_results
models_evaluation_test['xgb_all_models'] = test
_results

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

```
Tuning parameters:

Training the model..
Done. Time taken : 0:00:01.176195

Done

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

TEST DATA
------------------------------
RMSE :  1.0751967713900248
MAPE :  35.10661419925215
```



4.5 Comparision between all models

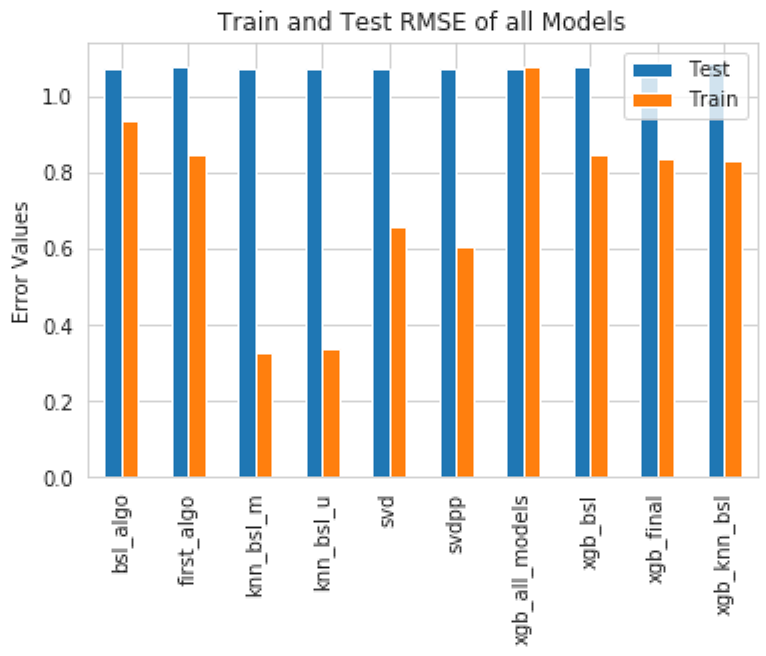
In [98]:
```
# Saving our TEST_RESULTS into a dataframe so t
hat you don't have to run it again
pd.DataFrame(models_evaluation_test).to_csv('af
ter_tuned_small_sample_results.csv')
models = pd.read_csv('after_tuned_small_sample_
results.csv', index_col=0)
models.loc['rmse'].sort_values()
```

Out[98]:
```
svd                 1.0726046873826458
knn_bsl_u           1.0726493739667242
knn_bsl_m            1.072758832653683
svdpp               1.0728491944183447
bsl_algo            1.0730330260516174
xgb_all_models      1.0751967713900248

first_algo           1.076373581778953
xgb_bsl             1.0765603714651855
xgb_final             1.08177621710006
xgb_knn_bsl         1.0882423800960463
```

```
Name: rmse, dtype: object
```

In [97]:

```
train_performance = pd.DataFrame(models_evaluation_train)
test_performance = pd.DataFrame(models_evaluation_test)
performance_dataframe = pd.DataFrame({'Train':train_performance.loc["rmse"],'Test':test_performance.loc["rmse"]})
performance_dataframe.plot(kind = "bar",grid = True)
plt.title("Train and Test RMSE of all Models")
plt.ylabel("Error Values")
plt.show()
```



> The best model is svd with the least RMSE value is 1.0726046