1. Libraries used

In [1]:

```
# this is just to know how much time will it take to run this entire ipython notebook
# date and time:
from datetime import datetime

# data structures:
from surprise import Reader, Dataset
import pandas as pd
import numpy as np
from scipy import sparse
from scipy.sparse import csr_matrix

# visuals:
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')

# Misc:
import os
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.model_selection import GridSearchCV as sklearnGridSearchCV
#from surprise.model_selection.search import GridSearchCV as surpriseGridSearchCV
import random

# models:
from surprise import BaselineOnly,KNNBaseline,SVD,SVDpp
import xgboost as xgb

```

```
C:\Users\Byron\Applications\PythonMaster\lib\site-packages\ipykernel_launche
r.py:14: UserWarning:
This call to matplotlib.use() has no effect because the backend has already
been chosen; matplotlib.use() must be called *before* pylab, matplotlib.pypl
ot,
or matplotlib.backends is imported for the first time.

The backend was *originally* set to 'module://ipykernel.pylab.backend_inlin
e' by the following code:
  File "C:\Users\Byron\Applications\PythonMaster\lib\runpy.py", line 193, in
_run_module_as_main
    "__main__", mod_spec)
  File "C:\Users\Byron\Applications\PythonMaster\lib\runpy.py", line 85, in
_run_code
    exec(code, run_globals)
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\ipykernel
_launcher.py", line 16, in <module>
    app.launch_new_instance()
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\traitlets
\config\application.py", line 658, in launch_instance
    app.start()
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\ipykernel
\kernelapp.py", line 497, in start
    self.io_loop.start()
```

```
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\tornado\p
latform\asyncio.py", line 132, in start
    self.asyncio_loop.run_forever()
  File "C:\Users\Byron\Applications\PythonMaster\lib\asyncio\base_events.p
y", line 523, in run_forever
    self._run_once()
  File "C:\Users\Byron\Applications\PythonMaster\lib\asyncio\base_events.p
y", line 1758, in _run_once
    handle._run()
  File "C:\Users\Byron\Applications\PythonMaster\lib\asyncio\events.py", lin
e 88, in _run
    self._context.run(self._callback, *self._args)
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\tornado\p
latform\asyncio.py", line 122, in _handle_events
    handler_func(fileobj, events)
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\tornado\s
tack_context.py", line 300, in null_wrapper
    return fn(*args, **kwargs)
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\zmq\event
loop\zmqstream.py", line 450, in _handle_events
    self._handle_recv()
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\zmq\event
loop\zmqstream.py", line 480, in _handle_recv
    self._run_callback(callback, msg)
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\zmq\event
loop\zmqstream.py", line 432, in _run_callback
    callback(*args, **kwargs)
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\tornado\s
tack_context.py", line 300, in null_wrapper
    return fn(*args, **kwargs)
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\ipykernel
\kernelbase.py", line 283, in dispatcher
    return self.dispatch_shell(stream, msg)
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\ipykernel
\kernelbase.py", line 233, in dispatch_shell
    handler(stream, idents, msg)
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\ipykernel
\kernelbase.py", line 399, in execute_request
    user_expressions, allow_stdin)
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\ipykernel
\ipkernel.py", line 208, in do_execute
    res = shell.run_cell(code, store_history=store_history, silent=silent)
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\ipykernel
\zmqshell.py", line 537, in run_cell
    return super(ZMQInteractiveShell, self).run_cell(*args, **kwargs)
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\IPython\c
ore\interactiveshell.py", line 2666, in run_cell
    self.events.trigger('post_run_cell', result)
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\IPython\c
ore\events.py", line 88, in trigger
    func(*args, **kwargs)
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\ipykernel
\pylab\backend_inline.py", line 164, in configure_once
    activate_matplotlib(backend)
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\IPython\c
ore\pylabtools.py", line 311, in activate_matplotlib
    matplotlib.pyplot.switch_backend(backend)
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\matplotli
b\pyplot.py", line 231, in switch_backend
    matplotlib.use(newbackend, warn=False, force=True)
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\matplotli
```

```
b\__init__.py", line 1422, in use
    reload(sys.modules['matplotlib.backends'])
  File "C:\Users\Byron\Applications\PythonMaster\lib\importlib\__init__.py",
line 169, in reload
    _bootstrap._exec(spec, module)
  File "C:\Users\Byron\Applications\PythonMaster\lib\site-packages\matplotli
b\backends\__init__.py", line 16, in <module>
    line for line in traceback.format_stack()
```

2. Data acquisition

In [2]:

```
start = datetime.now()
if not os.path.isfile(os.getcwd() + r'\\data\\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 f
    data = open('data/data.csv', mode='w')

    row = list()
    files=['data/combined_data_1.txt','data/combined_data_2.txt',
           'data/combined_data_3.txt', 'data/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 appear
                    movie_id = line.replace(':', '')
                else:
                    row = [x for x in line.split(',')]
                    row.insert(0, movie_id)
                    data.write(','.join(row))
                    data.write('\n')
        print("Done.\n")
    data.close()
print('Time taken :', datetime.now() - start)

```

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

In [3]:

```
print("creating the dataframe from data.csv file..")
df = pd.read_csv('data/data.csv', sep=',',
                       names=['movie', 'user','rating','date'])
df.date = pd.to_datetime(df.date)
print('Done.\n')

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

```

```
creating the dataframe from data.csv file..
Done.

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

In [4]:

```
df.head()

```

Out[4]:

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

In [5]:

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

```

Out[5]:

```
count    1.004805e+08
mean     3.604290e+00
std      1.085219e+00
min      1.000000e+00
25%      3.000000e+00
50%      4.000000e+00
75%      4.000000e+00
max      5.000000e+00
Name: rating, dtype: float64
```

2.1. Checking for NULL values

In [6]:

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

```

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

2.2. Checking for duplicates

In [7]:

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

```

```
There are 0 duplicate rating entries in the data..
```

2.3. Basic statistics

In [8]:

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

```

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

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

2.4. Spliting data into Train and Test(80:20)

In [9]:

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

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

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

```

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

In [10]:

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

```

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

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

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

In [11]:

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

```

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

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

3. Creating sparse matrix from data frame

3.1. Creating sparse matrix from train data frame

In [12]:

```
start = datetime.now()
if os.path.isfile('train_sparse_matrix.npz'):
    print("It is present in your pwd, getting it from disk....")
    # just get it from the disk instead of computing it
    train_sparse_matrix = sparse.load_npz('train_sparse_matrix.npz')
    print("DONE..")
else:
    print("We are creating sparse_matrix from the dataframe..")
    # create sparse_matrix and store it for after usage.
    # csr_matrix((data_values, (row_index, col_index)), shape=(row,column))
    # It should be in such a way that, MATRIX[row, col] = data
    train_sparse_matrix = sparse.csr_matrix((train_df['rating'].values, (train_df['user

    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)

```

```
We are creating sparse_matrix from the dataframe..
Done. It's shape is : (user, movie) :  (2649430, 17771)
Saving it into disk for furthur usage..
Done..

0:00:54.158307
```

3.1.1. The Sparsity of Train Sparse Matrix

In [13]:

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

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

```

```
Sparsity Of Train matrix : 99.8292709259195 %
```

3.2. Creating sparse matrix from test data frame

In [14]:

```
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']

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

print(datetime.now() - start)

```

```
We are creating sparse_matrix from the dataframe..
Done. It's shape is : (user, movie) :  (2649430, 17771)
Saving it into disk for furthur usage..
Done..

0:00:14.595046
```

3.2.1. The Sparsity of Test Sparse Matrix

In [15]:

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

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

```

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

4. 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 (sum per row),0 - Movie axes (sum per co

    # ".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 ratings..
    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

```

4.1. 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}
```

4.2. 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
```

4.3. 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])

```

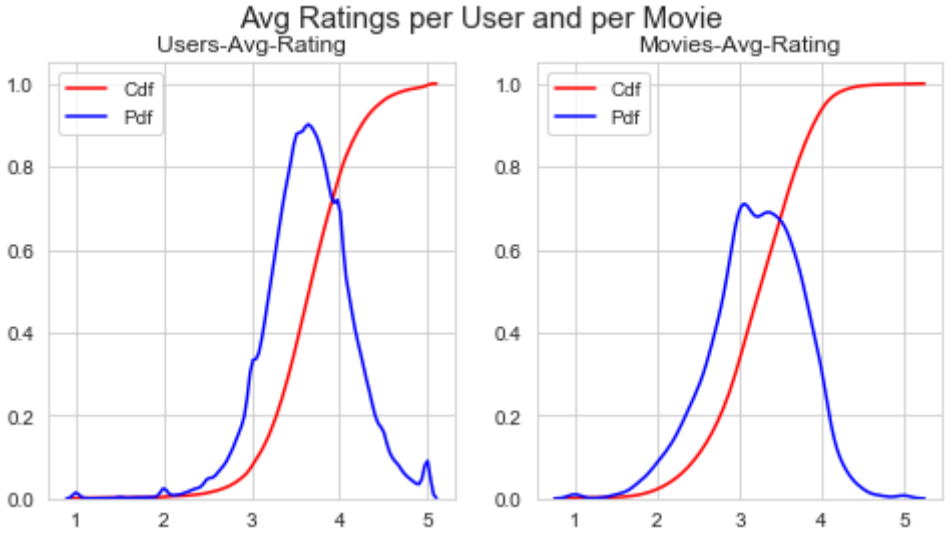
```
 AVerage rating of movie 15 : 3.3038461538461537
```

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

In [20]:

```
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-Rating')
# 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',color='r')
sns.distplot(user_averages, ax=ax1, hist=False,label='Pdf',color='b')

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',color='r')
sns.distplot(movie_averages, ax=ax2, hist=False, label='Pdf',color='b')

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

```

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




```
0:00:55.539501
```

5.1. Cold Start problem

5.1.1. Cold Start problem with Users

In [21]:

```
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_
```

```
Total number of Users  : 480189

Number of Users in Train data : 405041

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

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

5.1.2. Cold Start problem with Movies

In [22]:

```
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_movie
                                                                           np.round((new_
```

```
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

6.1. Machine Learning Models

In [23]:

```
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_movi

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

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

```

6.2. Sampling Data

6.2.1. Build sample train data from the train data

In [24]:

```
start = datetime.now()
path = "sample_train_sparse_matrix.npz"
if os.path.isfile(path):
    print("It is present in your pwd, getting it from disk....")
    # just get it from the disk instead of computing it
    sample_train_sparse_matrix = sparse.load_npz(path)
    print("DONE..")
else:
    # get 25k users and 3k movies from available data
    sample_train_sparse_matrix = get_sample_sparse_matrix(train_sparse_matrix, no_user

print(datetime.now() - start)

```

```
Original Matrix : (users, movies) -- (405041 17424)
Original Matrix : Ratings -- 80384405

Sampled Matrix : (users, movies) -- (25000 3000)
Sampled Matrix : Ratings -- 856986
Saving it into disk for furthur usage..
Done..

0:00:36.306218
```

6.2.2. Build sample test data from the test data

In [25]:

```
start = datetime.now()

path = "sample_test_sparse_matrix.npz"
if os.path.isfile(path):
    print("It is present in your pwd, getting it from disk....")
    # just get it from the disk instead of computing it
    sample_test_sparse_matrix = sparse.load_npz(path)
    print("DONE..")
else:
    # get 10k users and 1k movies from available data
    sample_test_sparse_matrix = get_sample_sparse_matrix(test_sparse_matrix, no_users=
print(datetime.now() - start)

```

```
Original Matrix : (users, movies) -- (349312 17757)
Original Matrix : Ratings -- 20096102

Sampled Matrix : (users, movies) -- (10000 2000)
Sampled Matrix : Ratings -- 68854
Saving it into disk for furthur usage..
Done..

0:00:08.608969
```

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

In [26]:

```
sample_train_averages = dict()

```

7.2. Finding Global Average of all movie ratings

In [27]:

```
# get the global average of ratings in our train set.
global_average = sample_train_sparse_matrix.sum()/sample_train_sparse_matrix.count_non
sample_train_averages['global'] = global_average
sample_train_averages

```

Out[27]:

```
{'global': 3.5875813607223455}
```

7.3. Finding Average rating per User

In [28]:

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

```

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

7.4. Finding Average rating per Movie

In [29]:

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

```

```
 AVerage rating of movie 15153 : 2.752
```

8.1. Featurizing data

In [30]:

```
print('\n No of ratings in Our Sampled train matrix is : {}\n'.format(sample_train_spa
print('\n No of ratings in Our Sampled test  matrix is : {}\n'.format(sample_test_spar
```

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

No of ratings in Our Sampled test  matrix is : 68854
```

8.2. Featurizing data for regression problem

8.2.1 Featurizing train data

In [31]:

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

In [32]:

```
############################################################
# It took me almost 10 hours to prepare this train dataset.#
############################################################
start = datetime.now()
if os.path.isfile('data/reg_train.csv'):
    print("File already exists you don't have to prepare again..." )
else:
    print('preparing {} tuples for the dataset..\n'.format(len(sample_train_ratings)))
    with open('data/reg_train.csv', mode='w') as reg_data_file:
        count = 0
        for (user, movie, rating)  in zip(sample_train_users, sample_train_movies, sam
            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_trai
            top_sim_users = user_sim.argsort()[::-1][1:] # we are ignoring 'The User'
            # get the ratings of most similar users for this movie
            top_ratings = sample_train_sparse_matrix[top_sim_users, movie].toarray().r
            # 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 -
        #     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, sampl
            top_sim_movies = movie_sim.argsort()[::-1][1:] # we are ignoring 'The User
            # get the ratings of most similar movie rated by this user..
            top_ratings = sample_train_sparse_matrix[user, top_sim_movies].toarray().r
            # 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
        #     print(top_sim_movies_ratings, end=" : -- ")

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

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

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

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

print(datetime.now() - start)

```

```
File already exists you don't have to prepare again...
0:00:00.000998
```

Reading from the file to make a Train_dataframe

In [33]:

```
reg_train_df = pd.read_csv('data/reg_train.csv', names = ['user', 'movie', 'GAvg', 'su
                                                          'smr1', 'smr2', 'smr3', 'smr4',
reg_train_df.sample(n=5)

```

Out[33]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5
377	29446	30	3.583034	5	3.0	3.0	5.0	3.0	5.0	5.0	5.0	5.0	5.0
87	1776165	12	3.583034	3	3.0	5.0	2.0	3.0	4.0	4.0	4.0	4.0	5.0
184	1270980	17	3.583034	3	2.0	3.0	2.0	3.0	4.0	4.0	3.0	4.0	4.0
173	1120332	17	3.583034	2	1.0	3.0	1.0	2.0	4.0	3.0	4.0	4.0	4.0
113	5980	17	3.583034	1	2.0	3.0	3.0	1.0	5.0	4.0	5.0	2.0	3.0

8.2.2 Featurizing test data

In [34]:

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

```

In [35]:

```
sample_train_averages['global']

```

Out[35]:

```
3.5875813607223455
```

In [36]:

```
start = datetime.now()

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

    print('preparing {} tuples for the dataset..\n'.format(len(sample_test_ratings)))
    with open('data/reg_test.csv', mode='w') as reg_data_file:
        count = 0
        for (user, movie, rating)  in zip(sample_test_users, sample_test_movies, sampl
            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_
                top_sim_users = user_sim.argsort()[::-1][1:] # we are ignoring 'The Us
                # get the ratings of most similar users for this movie
                top_ratings = sample_train_sparse_matrix[top_sim_users, movie].toarray
                # 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]]*(
                # 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
                ########## Cold STart Problem ##########
                top_sim_users_ratings.extend([sample_train_averages['global']]*(5 - le
                #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, s
                top_sim_movies = movie_sim.argsort()[::-1][1:] # we are ignoring 'The
                # get the ratings of most similar movie rated by this user..
                top_ratings = sample_train_sparse_matrix[user, top_sim_movies].toarray
                # 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
                #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
                #print(top_sim_movies_ratings)
            except :
                raise

            #-----------------prepare the row to be stores in a file-----------------#
            row = list()
```

```
            # add usser and movie name first
            row.append(user)
            row.append(movie)
            row.append(sample_train_averages['global']) # first feature
            #print(row)
            # next 5 features are similar_users "movie" ratings
            row.extend(top_sim_users_ratings)
            #print(row)
            # next 5 features are "user" ratings for similar_movies
            row.extend(top_sim_movies_ratings)
            #print(row)
            # Avg_user rating
            try:
                row.append(sample_train_averages['user'][user])
            except KeyError:
                row.append(sample_train_averages['global'])
            except:
                raise
            #print(row)
            # Avg_movie rating
            try:
                row.append(sample_train_averages['movie'][movie])
            except KeyError:
                row.append(sample_train_averages['global'])
            except:
                raise
            #print(row)
            # finalley, The actual Rating of this user-movie pair...
            row.append(rating)
            #print(row)
            count = count + 1

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

```

```
preparing 68854 tuples for the dataset..

Done for 1000 rows----- 0:06:47.875655
Done for 2000 rows----- 0:13:35.976798
Done for 3000 rows----- 0:20:25.191625
Done for 4000 rows----- 0:27:14.779766
Done for 5000 rows----- 0:34:05.131268
Done for 6000 rows----- 0:40:52.836060
Done for 7000 rows----- 0:47:40.373562
Done for 8000 rows----- 0:54:27.574833
Done for 9000 rows----- 1:01:15.271022
Done for 10000 rows----- 1:08:03.229344
Done for 11000 rows----- 1:14:51.049218
Done for 12000 rows----- 1:21:40.473169
Done for 13000 rows----- 1:28:33.291700
Done for 14000 rows----- 1:35:22.588255
Done for 15000 rows----- 1:42:09.931676
Done for 16000 rows----- 1:48:57.884408
```

```
Done for 17000 rows----- 1:55:45.539446
Done for 18000 rows----- 2:02:34.363992
Done for 19000 rows----- 2:09:24.736460
Done for 20000 rows----- 2:16:12.960504
Done for 21000 rows----- 2:23:00.697405
Done for 22000 rows----- 2:29:50.503464
Done for 23000 rows----- 2:36:39.261033
Done for 24000 rows----- 2:43:27.963430
Done for 25000 rows----- 2:50:17.065249
Done for 26000 rows----- 2:57:04.508412
Done for 27000 rows----- 3:03:51.466688
Done for 28000 rows----- 3:10:39.442825
Done for 29000 rows----- 3:17:26.611302
Done for 30000 rows----- 3:24:15.010375
Done for 31000 rows----- 3:31:06.841712
Done for 32000 rows----- 3:37:55.688509
Done for 33000 rows----- 3:44:42.781330
Done for 34000 rows----- 3:51:29.639095
Done for 35000 rows----- 3:58:16.384106
Done for 36000 rows----- 4:05:05.350731
Done for 37000 rows----- 4:11:54.142754
Done for 38000 rows----- 4:18:43.504108
Done for 39000 rows----- 4:25:30.825862
Done for 40000 rows----- 4:32:17.744725
Done for 41000 rows----- 4:39:10.272154
Done for 42000 rows----- 4:45:57.376706
Done for 43000 rows----- 4:52:44.114672
Done for 44000 rows----- 4:59:32.148890
Done for 45000 rows----- 5:06:19.040508
Done for 46000 rows----- 5:13:05.699699
Done for 47000 rows----- 5:19:54.265569
Done for 48000 rows----- 5:26:42.878092
Done for 49000 rows----- 5:33:35.010656
Done for 50000 rows----- 5:40:22.401986
Done for 51000 rows----- 5:47:10.073079
Done for 52000 rows----- 5:53:57.110354
Done for 53000 rows----- 6:00:45.180706
Done for 54000 rows----- 6:07:32.036575
Done for 55000 rows----- 6:14:19.223312
Done for 56000 rows----- 6:21:07.146089
Done for 57000 rows----- 6:27:55.889090
Done for 58000 rows----- 6:34:43.181598
Done for 59000 rows----- 6:41:29.827752
Done for 60000 rows----- 6:48:16.248595
Done for 61000 rows----- 6:55:03.436354
Done for 62000 rows----- 7:01:49.963872
Done for 63000 rows----- 7:08:37.488259
Done for 64000 rows----- 7:15:24.833689
Done for 65000 rows----- 7:22:13.005775
Done for 66000 rows----- 7:29:00.747964
Done for 67000 rows----- 7:35:49.463682
Done for 68000 rows----- 7:42:37.553466
 7:48:25.357726
```

__Reading from the file to make a test dataframe __

In [37]:

```
reg_test_df = pd.read_csv('data/reg_test.csv', names = ['user', 'movie', 'GAvg', 'sur1
                                                        'smr1', 'smr2', 'smr3', 'smr4'
reg_test_df.sample(n=5)

```

Out[37]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1
44634	1174054	12435	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581
65285	284819	16755	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581
40511	1127812	11149	3.587581	1.000000	3.000000	3.000000	5.000000	3.000000	3.587581
39677	1853215	10947	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581
37916	1030812	10759	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581

9.1. Transforming data for Surprise models

9.1.1. Transforming train data

In [38]:

```
# 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_df[['user', 'movie', 'rating']], reader)

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

```

9.1.2. Transforming test data

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

In [39]:

```
testset = list(zip(reg_test_df['user'].values, reg_test_df['movie'].values, reg_test_d

```

10.1. Applying Machine Learning models

In [40]:

```
# store for models performance:
models_evaluation_train = dict()
models_evaluation_test = dict()

```

In [41]:

```
# 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
    """

    # fit the model
    print('Training the model..')
    start = datetime.now()
    grid = sklearnGridSearchCV(estimator=algo,param_grid={'max_depth':[3,5], 'learning_
    grid.fit(x_train, y_train)
    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 = grid.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 = grid.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)

    model = grid.best_estimator_
    # return these train and test results...
    return train_results, test_results,model

```

In [42]:

```
# 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)

```

In [43]:

```
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 ''pr
    '''
    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...')
    #grid = surpriseGridSearchCV(algo_class=algo,param_grid=params,cv=3,measures=['rmse
    algo.fit(trainset)
    #grid.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())
    #train_preds = grid.test(trainset)
    # 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))
```

```
58
59      #store them in the train dictionary
60      if verbose:
61          print('adding train results in the dictionary..')
62      train['rmse'] = train_rmse
63      train['mape'] = train_mape
64      train['predictions'] = train_pred_ratings
65
66      #------------ Evaluating Test data---------------#
67      st = datetime.now()
68      print('\nEvaluating for test data...')
69      # get the predictions( list of prediction classes) of test data
70      test_preds = algo.test(testset)
71      # get the predicted ratings from the list of predictions
72      test_actual_ratings, test_pred_ratings = get_ratings(test_preds)
73      # get error metrics from the predicted and actual ratings
74      test_rmse, test_mape = get_errors(test_preds)
75      print('time taken : {}'.format(datetime.now()-st))
76
77      if verbose:
78          print('-'*15)
79          print('Test Data')
80          print('-'*15)
81          print("RMSE : {}\n\nMAPE : {}\n".format(test_rmse, test_mape))
82      # store them in test dictionary
83      if verbose:
84          print('storing the test results in test dictionary...')
85      test['rmse'] = test_rmse
86      test['mape'] = test_mape
87      test['predictions'] = test_pred_ratings
88
89      print('\n'+'-'*45)
90      print('Total time taken to run this algorithm :', datetime.now() - start)
91
92      # return two dictionaries train and test
93      return train, test
94
```

In [44]:

```
1  del df, train_df, test_df, train_sparse_matrix, test_sparse_matrix, sample_train_sparse
```

10.1.1. XGBoost with initial 13 features

In [45]:

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

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

# initialize Our first XGBoost model...
first_xgb = xgb.XGBRegressor(silent=False, n_jobs=-1, random_state=15)
train_results, test_results, model = run_xgboost(first_xgb, x_train, y_train, x_test, y

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

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

```
[05:36:34] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 12 extra nodes, 0 pruned nodes, max_depth
=3
[05:36:34] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[05:36:34] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 12 extra nodes, 0 pruned nodes, max_depth
=3
[05:36:34] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[05:36:34] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[05:36:34] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[05:36:34] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
```

10.1.2. Suprise BaselineModel

options are to specify.., how to compute those user and item biases

In [46]:

```
bsl_options = {'method': 'sgd',
               'learning_rate': 0.001
               }
#param_grid = {'bsl_options' : {'method': ['sgd'],'learning_rate': [0.1,0.01,0.001]}}
bsl_algo = BaselineOnly(bsl_options=bsl_options,verbose=False)
#bsl_algo = BaselineOnly(verbose=False)
# run this algorithm.., It will return the train and test results..
#bsl_train_results, bsl_test_results = run_surprise(BaselineOnly, train_data, testset,
bsl_train_results, bsl_test_results = run_surprise(bsl_algo, trainset, testset, verbos

# 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...
Done. time taken : 0:00:00.002991

Evaluating the model with train data..
time taken : 0:00:00.003022
---------------
Train Data
---------------
RMSE : 1.0675920828174126

MAPE : 38.980203043007606

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.619373
---------------
Test Data
---------------
RMSE : 1.1689034758645582

MAPE : 33.373601497866325

storing the test results in test dictionary...

------------------------------------------------
Total time taken to run this algorithm : 0:00:00.627345
```

10.1.3. XGBoost with initial 13 features + Surprise Baseline predictor

In [47]:

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

```

Out[47]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	
0	124742	5	3.583034	4	5.0	1.0	1.0	3.0	4.0	3.0	5.0	3.0	5.0	3.7
1	273956	5	3.583034	3	5.0	4.0	5.0	5.0	4.0	2.0	3.0	4.0	3.0	3.2

In [48]:

```
# 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[48]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	sı
0	3321	5	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587
1	508584	5	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587

In [49]:

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

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

# initialize Our first XGBoost model...
xgb_bsl = xgb.XGBRegressor(silent=False, n_jobs=-1, random_state=15)
train_results, test_results, model = run_xgboost(xgb_bsl, x_train, y_train, x_test, y_

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

xgb.plot_importance(model)
plt.show()

```

```
Training the model..
[05:36:47] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[05:36:47] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[05:36:47] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[05:36:47] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[05:36:47] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[05:36:47] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
```

10.1.4. Surprise KNNBaseline predictor

10.1.4.1. Surprise KNNBaseline with user user similarities

In [50]:

```
# we specify , how to compute similarities and what to consider with sim_options to ou
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
bsl_options = {'method': 'sgd'}

knn_bsl_u = KNNBaseline(k=40, sim_options = sim_options, bsl_options = bsl_options, ve
knn_bsl_u_train_results, knn_bsl_u_test_results = run_surprise(knn_bsl_u, trainset, te

# 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...
Done. time taken : 0:00:00.008976

Evaluating the model with train data..
time taken : 0:00:00.030916
---------------
Train Data
---------------
RMSE : 0.0018378838387751617

MAPE : 0.015845450195162498

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.648268
---------------
Test Data
---------------
RMSE : 1.1688806621784829

MAPE : 33.37307053293579

storing the test results in test dictionary...

------------------------------------------------
Total time taken to run this algorithm : 0:00:00.690165
```

10.1.4.2. Surprise KNNBaseline with movie movie similarities

In [51]:

```
# we specify , how to compute similarities and what to consider with sim_options to ou
# 'user_based' : Fals => this considers the similarities of movies instead of users
sim_options = {'user_based' : False,
               'name': 'pearson_baseline',
               'shrinkage': 100,
               'min_support': 2
              }
# we keep other parameters like regularization parameter and learning_rate as default
bsl_options = {'method': 'sgd'}

knn_bsl_m = KNNBaseline(k=40, sim_options = sim_options, bsl_options = bsl_options, ve
knn_bsl_m_train_results, knn_bsl_m_test_results = run_surprise(knn_bsl_m, trainset, te

# 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...
Done. time taken : 0:00:00.001994

Evaluating the model with train data..
time taken : 0:00:00.004987
---------------
Train Data
---------------
RMSE : 0.0035705539547701537

MAPE : 0.02886108918725288

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.700125
---------------
Test Data
---------------
RMSE : 1.1688806621784829

MAPE : 33.37307053293579

storing the test results in test dictionary...

------------------------------------------
Total time taken to run this algorithm : 0:00:00.708105
```

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

In [52]:

```
# add the predicted values from both knns to this dataframe
reg_train_df['knn_bsl_u'] = models_evaluation_train['knn_bsl_u']['predictions']
reg_train_df['knn_bsl_m'] = models_evaluation_train['knn_bsl_m']['predictions']
reg_train_df.head(2)

```

Out[52]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	
0	124742	5	3.583034	4	5.0	1.0	1.0	3.0	4.0	3.0	5.0	3.0	5.0	3.7
1	273956	5	3.583034	3	5.0	4.0	5.0	5.0	4.0	2.0	3.0	4.0	3.0	3.2

In [53]:

```
reg_test_df['knn_bsl_u'] = models_evaluation_test['knn_bsl_u']['predictions']
reg_test_df['knn_bsl_m'] = models_evaluation_test['knn_bsl_m']['predictions']
reg_test_df.head(2)

```

Out[53]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	sı
0	3321	5	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587
1	508584	5	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587581	3.587

In [54]:

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

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

# declare the model
xgb_knn_bsl = xgb.XGBRegressor(silent=False, n_jobs=-1, random_state=15)
train_results, test_results, model = run_xgboost(xgb_knn_bsl, x_train, y_train, x_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(model)
plt.show()

```

```
Training the model..
[05:37:00] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[05:37:00] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[05:37:00] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[05:37:00] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[05:37:00] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[05:37:00] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
```

11.1. Matrix Factorization Techniques

11.1.1. SVD Matrix Factorization User Movie interactions

In [55]:

```
# initiallize the model
svd = SVD(n_factors=100, biased=True, random_state=15, verbose=False)
svd_train_results, svd_test_results = run_surprise(svd, trainset, testset, verbose=Tru

# 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...
Done. time taken : 0:00:00.031915

Evaluating the model with train data..
time taken : 0:00:00.005011
---------------
Train Data
---------------
RMSE : 0.7463638215626787

MAPE : 27.25371834203118

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.599427
---------------
Test Data
---------------
RMSE : 1.1688853693610264

MAPE : 33.37295795845765

storing the test results in test dictionary...

------------------------------------------
Total time taken to run this algorithm : 0:00:00.640312
```

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

In [56]:

```
# initiallize the model
svdpp = SVDpp(n_factors=50, random_state=15, verbose=False)
svdpp_train_results, svdpp_test_results = run_surprise(svdpp, trainset, testset, verbo

# 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...
Done. time taken : 0:00:00.049891

Evaluating the model with train data..
time taken : 0:00:00.006982
---------------
Train Data
---------------
RMSE : 0.723851431118292

MAPE : 26.489342645042257

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.673208
---------------
Test Data
---------------
RMSE : 1.1688866271595812

MAPE : 33.372675910891786

storing the test results in test dictionary...

------------------------------------------
Total time taken to run this algorithm : 0:00:00.731078
```

11.1.3. XgBoost with 13 features + Surprise Baseline + Surprise KNNbaseline + MF Techniques

In [57]:

```
reg_train_df['svd'] = models_evaluation_train['svd']['predictions']
reg_train_df['svdpp'] = models_evaluation_train['svdpp']['predictions']
reg_train_df.head(2)

reg_test_df['svd'] = models_evaluation_test['svd']['predictions']
reg_test_df['svdpp'] = models_evaluation_test['svdpp']['predictions']
reg_test_df.head(2)

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

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

xgb_final = xgb.XGBRegressor(silent=False, n_jobs=-1, random_state=15)
train_results, test_results, model = run_xgboost(xgb_final, x_train, y_train, x_test, y

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

xgb.plot_importance(model)
plt.show()

```

```
3
[05:37:15] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 12 extra nodes, 0 pruned nodes, max_depth
=3
[05:37:15] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 8 extra nodes, 0 pruned nodes, max_depth=
3
[05:37:15] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3
[05:37:15] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 14 extra nodes, 0 pruned nodes, max_depth
=3

[05:37:15] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 8 extra nodes, 0 pruned nodes, max_depth=
3
[05:37:15] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 12 extra nodes, 0 pruned nodes, max_depth
=3
```

11.1.4. XgBoost with Surprise Baseline + Surprise KNNbaseline + MF Techniques

In [58]:

```
# prepare train data
x_train = reg_train_df.loc[:,['knn_bsl_u', 'knn_bsl_m', 'svd', 'svdpp']]
y_train = reg_train_df.loc[:,'rating']

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


xgb_all_models = xgb.XGBRegressor(silent=False, n_jobs=-1, random_state=15)
train_results, test_results, model = run_xgboost(xgb_all_models, x_train, y_train, x_t

# 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(model)
plt.show()

```

```
Training the model..
[05:37:26] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 6 extra nodes, 0 pruned nodes, max_depth=
3
[05:37:26] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 10 extra nodes, 0 pruned nodes, max_depth
=3
[05:37:26] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 10 extra nodes, 0 pruned nodes, max_depth
=3
[05:37:26] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 12 extra nodes, 0 pruned nodes, max_depth
=3
[05:37:26] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 8 extra nodes, 0 pruned nodes, max_depth=
3
[05:37:26] C:\Users\Administrator\Desktop\xgboost\src\tree\updater_prune.c
c:74: tree pruning end, 1 roots, 12 extra nodes, 0 pruned nodes, max_depth
=3
```

12. Comparision between all models

In [59]:

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

Out[59]:

```
first_algo        1.0919342032973887
xgb_bsl           1.0994878447416228
xgb_knn_bsl       1.1075294007177148
xgb_final         1.1139816678822365
knn_bsl_u         1.1688806621784829
knn_bsl_m         1.1688806621784829
svd               1.1688853693610264
svdpp             1.1688866271595812
bsl_algo          1.1689034758645582
xgb_all_models    1.4344114306676312
Name: rmse, dtype: object
```

In [60]:

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

Out[60]:

```
knn_bsl_u         0.0018378838387751617
knn_bsl_m         0.0035705539547701537
xgb_final           0.47529593076806237
xgb_knn_bsl          0.4886132118878459
xgb_bsl              0.5017121670051167
first_algo           0.6352350008338273
svdpp                 0.723851431118292
svd                  0.7463638215626787
xgb_all_models       0.8864385885024016
bsl_algo             1.0675920828174126
Name: rmse, dtype: object
```