Netflix Movie Recommendation System - Model Optimization

By Aziz Presswala

In [0]:

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

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

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

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

3.3.6.1 Creating sparse matrix from train data frame

In [0]:

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

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

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

```
It is present in your pwd, getting it from disk....
DONE..
0:00:04.463750
```

The Sparsity of Train Sparse Matrix

In [0]:

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

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

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

3.3.6.2 Creating sparse matrix from test data frame

In [0]:

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

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

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

```
It is present in your pwd, getting it from disk....
DONE..
0:00:01.172240
```

The Sparsity of Test data Matrix

In [0]:

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

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

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

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

In [0]:

```
# get the user averages in dictionary (key: user_id/movie_id, value: avg rating)

def get_average_ratings(sparse_matrix, of_users):

    # average ratings of user/axes
    ax = 1 if of_users else 0 # 1 - User axes,0 - Movie axes

    # ".A1" is for converting Column_Matrix to 1-D numpy array
    sum_of_ratings = sparse_matrix.sum(axis=ax).A1
    # Boolean matrix of ratings ( whether a user rated that movie or not)
    is_rated = sparse_matrix!=0
    # no of ratings that each user OR movie..
    no_of_ratings = is_rated.sum(axis=ax).A1

    # max_user  and max_movie ids in sparse matrix
    u,m = sparse_matrix.shape
    # creae a dictonary of users and their average ratigns..
    average_ratings = { i : sum_of_ratings[i]/no_of_ratings[i]
                                 for i in range(u if of_users else m)
                                    if no_of_ratings[i] !=0}

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

3.3.7.1 finding global average of all movie ratings

In [0]:

```
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[36]:

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

3.3.7.2 finding average rating per user

In [0]:

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

```
Average rating of user 10 : 3.3781094527363185
```

3.3.7.3 finding average rating per movie

In [0]:

```
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. Machine Learning Models

In [3]:

```
!pip install kaggle
from google.colab import files
files.upload()
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.6/dist-packa
ges (1.5.3)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/pytho
n3.6/dist-packages (from kaggle) (1.22)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.6/dist-pa
ckages (from kaggle) (1.11.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.6/dist-pack
ages (from kaggle) (2019.3.9)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.6/d
ist-packages (from kaggle) (2.5.3)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-pac
kages (from kaggle) (2.18.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-package
s (from kaggle) (4.28.1)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.6/di
st-packages (from kaggle) (3.0.1)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/pytho
n3.6/dist-packages (from requests->kaggle) (3.0.4)
Requirement already satisfied: idna<2.7,>=2.5 in /usr/local/lib/python3.6/di
st-packages (from requests->kaggle) (2.6)
Requirement already satisfied: text-unidecode==1.2 in /usr/local/lib/python
3.6/dist-packages (from python-slugify->kaggle) (1.2)
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

Out[3]:

{'kaggle.json': b'{"username":"pankajkarki","key":"563115ce1ea9892ab835dfbe5
b8acba1"}'}

In [4]:

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/

# This permissions change avoids a warning on Kaggle tool startup.
!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d pankajkarki/netflix

!ls
```

```
Downloading netflix.zip to /content
  0% 0.00/2.13M [00:00<?, ?B/s]
100% 2.13M/2.13M [00:00<00:00, 70.8MB/s]
kaggle.json  netflix.zip  sample_data
```

In [5]:

```
!unzip netflix.zip
```

```
Archive:  netflix.zip
  inflating: sample_train_sparse_matrix.npz
  inflating: reg_train.csv
  inflating: reg_test.csv
  inflating: sample_test_sparse_matrix.npz
```

In [0]:

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

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

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

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

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

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

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

    return sample_sparse_matrix
```

4.1 Sampling Data

4.1.1 Build sample train data from the train data

In [7]:

```
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_users=25(
                                             path = path)

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

```
It is present in your pwd, getting it from disk....
DONE..
0:00:00.054409
```

4.1.2 Build sample test data from the test data

In [8]:

```
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 9k users and 1500 movies from available data
    sample_test_sparse_matrix = get_sample_sparse_matrix(test_sparse_matrix, no_users=9000,
                                                 path = "sample/small/sample_test_sparse_ma
print(datetime.now() - start)
```

```
It is present in your pwd, getting it from disk....
DONE..
0:00:00.045440
```

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

In [0]:

```
sample_train_averages = dict()
```

4.2.1 Finding Global Average of all movie ratings

In [10]:

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

Out[10]:

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

4.2.2 Finding Average rating per User

In [11]:

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

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

4.2.3 Finding Average rating per Movie

In [12]:

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

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

4.3 Featurizing data

In [13]:

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

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


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

4.3.1 Featurizing data for regression problem

4.3.1.1 Featurizing train data

In [0]:

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

In [15]:

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


            #--------------------- Ratings by "user"  to similar movies of "movie" --------
            # compute the similar movies of the "movie"
            movie_sim = cosine_similarity(sample_train_sparse_matrix[:,movie].T, sample_tra
            top_sim_movies = movie_sim.argsort()[::-1][1:] # we are ignoring 'The User' fro
            # 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_
        #     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...
0:00:00.001911
```

Reading from the file to make a Train_dataframe

In [17]:

```
reg_train = pd.read_csv('reg_train.csv', names = ['user', 'movie', 'GAvg', 'sur1', 'sur2',
reg_train.head()
```

Out[17]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	
0	53406	33	3.581679	4.0	5.0	5.0	4.0	1.0	5.0	2.0	5.0	3.0	1.0	3.3
1	99540	33	3.581679	5.0	5.0	5.0	4.0	5.0	3.0	4.0	4.0	3.0	5.0	3.5
2	99865	33	3.581679	5.0	5.0	4.0	5.0	3.0	5.0	4.0	4.0	5.0	4.0	3.7
3	101620	33	3.581679	2.0	3.0	5.0	5.0	4.0	4.0	3.0	3.0	4.0	5.0	3.5
4	112974	33	3.581679	5.0	5.0	5.0	5.0	5.0	3.0	5.0	5.0	5.0	3.0	3.7

- **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 [0]:

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

In [19]:

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

Out[19]:

```
3.581679377504138
```

In [20]:

```
start = datetime.now()

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

    print('preparing {} tuples for the dataset..\n'.format(len(sample_test_ratings)))
    with open('reg_test.csv', mode='w') as reg_data_file:
        count = 0
        for (user, movie, rating)  in zip(sample_test_users, sample_test_movies, sample_tes
            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_trair
                top_sim_users = user_sim.argsort()[::-1][1:] # we are ignoring 'The User' f
                # get the ratings of most similar users for this movie
                top_ratings = sample_train_sparse_matrix[top_sim_users, movie].toarray().ra
                # 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 - l
                # 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
                ########## Cold STart Problem ##########
                top_sim_users_ratings.extend([sample_train_averages['global']]*(5 - len(top
                #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
                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().ra
                # 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)
            except (IndexError, KeyError):
                #print(top_sim_movies_ratings, end=" : -- ")
                top_sim_movies_ratings.extend([sample_train_averages['global']]*(5-len(top_
                #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)
```

It is already created...

__Reading from the file to make a test dataframe __

In [21]:

```
reg_test_df = pd.read_csv('reg_test.csv', names = ['user', 'movie', 'GAvg', 'sur1', 'sur2',
                                                    'smr1', 'smr2', 'smr3', 'smr4', '
                                                    'UAvg', 'MAvg', 'rating'], header
reg_test_df.head(4)
```

Out[21]:

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

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

4.3.2 Transforming data for Surprise models

In [23]:

```
!pip install surprise
from surprise import Reader, Dataset
```

```
Collecting surprise
  Downloading https://files.pythonhosted.org/packages/61/de/e5cba8682201fcf9
c3719a6fdda95693468ed061945493dea2dd37c5618b/surprise-0.1-py2.py3-none-any.w
hl (https://files.pythonhosted.org/packages/61/de/e5cba8682201fcf9c3719a6fdd
a95693468ed061945493dea2dd37c5618b/surprise-0.1-py2.py3-none-any.whl)
Collecting scikit-surprise (from surprise)
  Downloading https://files.pythonhosted.org/packages/4d/fc/cd4210b247d1dca4
21c25994740cbbf03c5e980e31881f10eaddf45fdab0/scikit-surprise-1.0.6.tar.gz (h
ttps://files.pythonhosted.org/packages/4d/fc/cd4210b247d1dca421c25994740cbbf
03c5e980e31881f10eaddf45fdab0/scikit-surprise-1.0.6.tar.gz) (3.3MB)
    100% |████████████████████████████████| 3.3MB 5.8MB/s
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist
-packages (from scikit-surprise->surprise) (0.12.5)
Requirement already satisfied: numpy>=1.11.2 in /usr/local/lib/python3.6/dis
t-packages (from scikit-surprise->surprise) (1.14.6)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.6/dist
-packages (from scikit-surprise->surprise) (1.1.0)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-
packages (from scikit-surprise->surprise) (1.11.0)
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (setup.py) ... done
  Stored in directory: /root/.cache/pip/wheels/ec/c0/55/3a28eab06b53c2200150
63ebbdb81213cd3dcbb72c088251ec
Successfully built scikit-surprise
Installing collected packages: scikit-surprise, surprise
Successfully installed scikit-surprise-1.0.6 surprise-0.1
```

4.3.2.1 Transforming train data

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

In [0]:

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

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

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

4.3.2.2 Transforming test data

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

In [25]:

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

Out[25]:

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

4.4 Applying Machine Learning models

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

> **keys** : model names(string)

> **value**: dict(**key** : metric, **value** : value)

In [26]:

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

models_evaluation_train, models_evaluation_test
```

Out[26]:

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

> **Utility functions for running regression models**

In [0]:

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

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

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


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

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

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

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

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

Utility functions for Surprise modes

In [0]:

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

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

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

    return actual, pred

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

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

    return rmse, mape*100

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

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

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

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

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

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

    #------------ Evaluating Test data---------------#
    st = datetime.now()
    print('\nEvaluating for test data...')
    # get the predictions( list of prediction classes) of test data
    test_preds = algo.test(testset)
    # get the predicted ratings from the list of predictions
    test_actual_ratings, test_pred_ratings = get_ratings(test_preds)
    # get error metrics from the predicted and actual ratings
    test_rmse, test_mape = get_errors(test_preds)
    print('time taken : {}'.format(datetime.now()-st))

    if verbose:
        print('-'*15)
        print('Test Data')
        print('-'*15)
        print("RMSE : {}\n\nMAPE : {}\n".format(test_rmse, test_mape))
    # store them in test dictionary
    if verbose:
        print('storing the test results in test dictionary...')
    test['rmse'] = test_rmse
    test['mape'] = test_mape
    test['predictions'] = test_pred_ratings

    print('\n'+'-'*45)
    print('Total time taken to run this algorithm :', datetime.now() - start)

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

4.4.1 XGBoost with initial 13 features

In [29]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit
import xgboost as xgb

# Hyperparameters
parameters = {'max_depth':[1,2,3],
              'learning_rate':[0.001,0.01,0.1],
              'n_estimators':[100,300,500,700]}

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

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

start = datetime.now()
# initialize Our first XGBoost model...
first_xgb = xgb.XGBRegressor(nthread=-1)

# Cross validation
gsv = GridSearchCV(first_xgb,
                    param_grid = parameters,
                    scoring="neg_mean_squared_error",
                    cv = TimeSeriesSplit(n_splits=5),
                    n_jobs = -1,
                    verbose = 1)
gsv_result = gsv.fit(x_train, y_train)

# Summarizing results
print("Best: %f using %s" % (gsv_result.best_score_, gsv_result.best_params_))
means = gsv_result.cv_results_['mean_test_score']
stds = gsv_result.cv_results_['std_test_score']
params = gsv_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

print("\nTime Taken: ",start - datetime.now())
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  3.8min
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed: 19.9min finished

Best: -0.717589 using {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators':
100}
-9.003600 (0.421070) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estim
ators': 100}
-6.396660 (0.323360) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estim
ators': 300}
-4.635431 (0.245576) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estim
ators': 500}
-3.444588 (0.195216) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estim
ators': 700}
-8.968829 (0.374920) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estim
```

```
ators': 100}
-6.330400 (0.267078) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estim
ators': 300}
-4.550069 (0.191141) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estim
ators': 500}
-3.349752 (0.140934) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estim
ators': 700}
-8.950769 (0.371915) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estim
ators': 100}
-6.284471 (0.244525) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estim
ators': 300}
-4.494928 (0.168086) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estim
ators': 500}
-3.290015 (0.117240) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estim
ators': 700}
-2.322729 (0.136005) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estima
tors': 100}
-0.891127 (0.051646) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estima
tors': 300}
-0.800698 (0.043965) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estima
tors': 500}
-0.767167 (0.038827) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estima
tors': 700}
-2.224923 (0.096179) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estima
tors': 100}
-0.800935 (0.035338) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estima
tors': 300}
-0.739654 (0.031972) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estima
tors': 500}
-0.726500 (0.029573) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estima
tors': 700}
-2.157529 (0.065650) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estima
tors': 100}
-0.767973 (0.029754) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estima
tors': 300}
-0.724136 (0.028816) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estima
tors': 500}
-0.718945 (0.027510) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estima
tors': 700}
-0.741297 (0.033748) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimat
ors': 100}
-0.721031 (0.027272) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimat
ors': 300}
-0.721005 (0.027328) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimat
ors': 500}
-0.721288 (0.027454) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimat
ors': 700}
-0.720860 (0.027226) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimat
ors': 100}
-0.718212 (0.026533) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimat
ors': 300}
-0.718551 (0.027519) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimat
ors': 500}
-0.719009 (0.028215) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimat
ors': 700}
-0.717589 (0.026650) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimat
ors': 100}
-0.718294 (0.028735) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimat
ors': 300}
-0.720595 (0.031825) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimat
ors': 500}
```

```
-0.722039 (0.033317) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimat
ors': 700}

Time Taken:  -1 day, 23:39:58.113017
```

In [30]:

```
xgb_bsl = xgb.XGBRegressor(max_depth=3, learning_rate = 0.1, n_estimators=100, nthread=-1)
xgb_bsl
```

Out[30]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=-1, objective='reg:linear', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

In [31]:

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

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

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

```
Training the model..
Done. Time taken : 0:00:06.609408

Done

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

TEST DATA
------------------------------
RMSE :  1.0761851474385373
MAPE :  34.504887593204884

<IPython.core.display.Javascript object>
```

4.4.2 Suprise BaselineModel

In [0]:

```
from surprise import BaselineOnly
```

_*Predicted*rating : (baseline prediction) __

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

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

- $\boldsymbol{\mu}$: Average of all trainings in training data.
- $\boldsymbol{b_u}$: User bias
- $\boldsymbol{b_i}$: Item bias (movie biases)

__Optimization function (Least Squares Problem) __

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

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

In [33]:

```
# options are to specify.., how to compute those user and item biases
bsl_options = {'method': 'sgd',
               'reg':0.01,
               'learning_rate': 0.001,
               'n_epochs':120
               }
bsl_algo = BaselineOnly(bsl_options=bsl_options)
# run this algorithm.., It will return the train and test results..
bsl_train_results, bsl_test_results = run_surprise(bsl_algo, trainset, testset, verbose=Tru


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

```
Training the model...
Estimating biases using sgd...
Done. time taken : 0:00:06.571262

Evaluating the model with train data..
time taken : 0:00:01.249900
---------------
Train Data
---------------
RMSE : 0.8883104307239651

MAPE : 27.274344731601268

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.211147
---------------
Test Data
---------------
RMSE : 1.0725953793894922

MAPE : 35.01055341888157

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:00:08.034151
```

4.4.3 XGBoost with initial 13 features + Surprise Baseline predictor

Updating Train Data

In [34]:

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

Out[34]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	
0	53406	33	3.581679	4.0	5.0	5.0	4.0	1.0	5.0	2.0	5.0	3.0	1.0	3.37
1	99540	33	3.581679	5.0	5.0	5.0	4.0	5.0	3.0	4.0	4.0	3.0	5.0	3.55

Updating Test Data

In [35]:

```
# 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[35]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	sı
0	808635	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581
1	941866	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581

In [38]:

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

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

# initialize Our first XGBoost model...
start = datetime.now()

# Initialize Our first XGBoost model
xgb = xgb.XGBRegressor(nthread=-1)

# Cross validation
gsv = GridSearchCV(xgb,
                    param_grid = parameters,
                    scoring="neg_mean_squared_error",
                    cv = TimeSeriesSplit(n_splits=5),
                    n_jobs = -1,
                    verbose = 1)
gsv_result = gsv.fit(x_train, y_train)

# Summarizing results
print("Best: %f using %s" % (gsv_result.best_score_, gsv_result.best_params_))
print()
means = gsv_result.cv_results_['mean_test_score']
stds = gsv_result.cv_results_['std_test_score']
params = gsv_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

print("\nTime Taken: ",datetime.now() -start)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  4.3min
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed: 22.7min finished

Best: -0.718201 using {'learning_rate': 0.1, 'max_depth': 3, 'n_estimator
s': 100}

-9.003600 (0.421070) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_est
imators': 100}
-6.396660 (0.323360) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_est
imators': 300}
-4.635431 (0.245576) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_est
imators': 500}
-3.444588 (0.195216) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_est
imators': 700}
-8.968829 (0.374920) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_est
imators': 100}
-6.330400 (0.267078) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_est
imators': 300}
```

```
-4.550069 (0.191141) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_est
imators': 500}
-3.349752 (0.140934) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_est
imators': 700}
-8.950769 (0.371915) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_est
imators': 100}
-6.284471 (0.244525) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_est
imators': 300}
-4.494928 (0.168086) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_est
imators': 500}
-3.290015 (0.117240) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_est
imators': 700}
-2.322729 (0.136005) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_esti
mators': 100}
-0.891127 (0.051646) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_esti
mators': 300}
-0.800698 (0.043965) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_esti
mators': 500}
-0.767167 (0.038827) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_esti
mators': 700}
-2.224923 (0.096179) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_esti
mators': 100}
-0.800935 (0.035338) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_esti
mators': 300}
-0.739654 (0.031972) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_esti
mators': 500}
-0.726500 (0.029573) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_esti
mators': 700}
-2.157529 (0.065650) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_esti
mators': 100}
-0.767979 (0.029763) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_esti
mators': 300}
-0.724202 (0.028836) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_esti
mators': 500}
-0.719040 (0.027658) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_esti
mators': 700}
-0.741297 (0.033748) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estim
ators': 100}
-0.721238 (0.027466) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estim
ators': 300}
-0.721127 (0.027470) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estim
ators': 500}
-0.721381 (0.027617) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estim
ators': 700}
-0.720922 (0.027313) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estim
ators': 100}
-0.718334 (0.026914) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estim
ators': 300}
-0.718794 (0.027844) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estim
ators': 500}
-0.719570 (0.028418) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estim
ators': 700}
-0.718201 (0.027065) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estim
ators': 100}
-0.718827 (0.028910) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estim
ators': 300}
-0.720730 (0.030690) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estim
ators': 500}
-0.722689 (0.032462) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estim
ators': 700}
```

```
Time Taken:  0:22:48.697097
```

In [39]:

```
import xgboost as xgb
xgb_bsl = xgb.XGBRegressor(max_depth=3,learning_rate = 0.1,n_estimators=100,nthread=-1)
xgb_bsl
```

Out[39]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=-1, objective='reg:linear', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

In [40]:

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

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

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

```
Training the model..
Done. Time taken : 0:00:07.398671

Done

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

TEST DATA
------------------------------
RMSE :  1.0762300237782054
MAPE :  34.5017727794818
```

```
<IPython.core.display.Javascript object>
```

4.4.4 Surprise KNNBaseline predictor

In [0]:

```
from surprise import KNNBaseline
```

- KNN BASELINE

- http://surprise.readthedocs.io/en/stable/knn_inspired.html#surprise.prediction_algorithms.knns.KNNBase (http://surprise.readthedocs.io/en/stable/knn_inspired.html#surprise.prediction_algorithms.knns.KNNBas

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

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

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

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

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

- __ Predicted rating __ (based on Item Item similarity):

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

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

4.4.4.1 Surprise KNNBaseline with user user similarities

In [42]:

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

# 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 : 0:00:35.523655

Evaluating the model with train data..
time taken : 0:02:13.628979
---------------
Train Data
---------------
RMSE : 0.33642097416508826

MAPE : 9.145093375416348

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.088311
---------------
Test Data
---------------
RMSE : 1.0726493739667242

MAPE : 35.02094499698424

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:02:49.243706
```

4.4.4.2 Surprise KNNBaseline with movie movie similarities

In [43]:

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

# '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 value
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

# 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 : 0:00:01.581487

Evaluating the model with train data..
time taken : 0:00:10.509544
---------------
Train Data
---------------
RMSE : 0.32584796251610554

MAPE : 8.447062581998374

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.084750
---------------
Test Data
---------------
RMSE : 1.072758832653683

MAPE : 35.02269653015042

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:00:12.179255
```

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 [44]:

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

reg_train.head(2)
```

Out[44]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	
0	53406	33	3.581679	4.0	5.0	5.0	4.0	1.0	5.0	2.0	5.0	3.0	1.0	3.37
1	99540	33	3.581679	5.0	5.0	5.0	4.0	5.0	3.0	4.0	4.0	3.0	5.0	3.55

__Preparing Test data __

In [45]:

```
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[45]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	sı
0	808635	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581
1	941866	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581

In [46]:

```
import xgboost as xgb
# prepare the train data....
x_train = reg_train.drop(['user', 'movie', 'rating'], axis=1)
y_train = reg_train['rating']

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

start = datetime.now()

# Initialize Our first XGBoost model
model = xgb.XGBRegressor(nthread=-1)

# Cross validation
gsv = GridSearchCV(model,
                    param_grid = parameters,
                    scoring="neg_mean_squared_error",
                    cv = TimeSeriesSplit(n_splits=5),
                    n_jobs = -1,
                    verbose = 1)
gsv_result = gsv.fit(x_train, y_train)

# Summarizing results
print("Best: %f using %s" % (gsv_result.best_score_, gsv_result.best_params_))
print()
means = gsv_result.cv_results_['mean_test_score']
stds = gsv_result.cv_results_['std_test_score']
params = gsv_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  5.4min
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed: 28.6min finished


Best: -0.718291 using {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators':
100}

-9.003600 (0.421070) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estim
ators': 100}
-6.396660 (0.323360) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estim
ators': 300}
-4.635431 (0.245576) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estim
ators': 500}
-3.444588 (0.195216) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estim
ators': 700}
-8.968829 (0.374920) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estim
ators': 100}
-6.330400 (0.267078) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estim
ators': 300}
-4.550069 (0.191141) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estim
ators': 500}
-3.349752 (0.140934) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estim
ators': 700}
```

```
-8.950769 (0.371915) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estim
ators': 100}
-6.284471 (0.244525) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estim
ators': 300}
-4.494928 (0.168086) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estim
ators': 500}
-3.290015 (0.117240) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estim
ators': 700}
-2.322729 (0.136005) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estima
tors': 100}
-0.891127 (0.051646) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estima
tors': 300}
-0.800698 (0.043965) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estima
tors': 500}
-0.767167 (0.038827) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estima
tors': 700}
-2.224923 (0.096179) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estima
tors': 100}
-0.800935 (0.035338) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estima
tors': 300}
-0.739654 (0.031972) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estima
tors': 500}
-0.726516 (0.029594) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estima
tors': 700}
-2.157529 (0.065650) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estima
tors': 100}
-0.767934 (0.029759) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estima
tors': 300}
-0.724220 (0.028937) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estima
tors': 500}
-0.719122 (0.027818) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estima
tors': 700}
-0.741297 (0.033748) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimat
ors': 100}
-0.721245 (0.027476) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimat
ors': 300}
-0.721152 (0.027528) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimat
ors': 500}
-0.721382 (0.027697) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimat
ors': 700}
-0.720899 (0.027299) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimat
ors': 100}
-0.718792 (0.027022) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimat
ors': 300}
-0.720042 (0.028150) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimat
ors': 500}
-0.720999 (0.028898) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimat
ors': 700}
-0.718291 (0.027369) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimat
ors': 100}
-0.719218 (0.028862) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimat
ors': 300}
-0.721670 (0.030927) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimat
ors': 500}
-0.723704 (0.032277) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimat
ors': 700}
```

In [47]:

```
import xgboost as xgb
xgb_knn_bsl = xgb.XGBRegressor(max_depth=3,learning_rate = 0.1,n_estimators=100,nthread=-1)
xgb_knn_bsl
```

Out[47]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=-1, objective='reg:linear', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

In [48]:

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

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


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

```
Training the model..
Done. Time taken : 0:00:09.333497

Done

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

TEST DATA
------------------------------
RMSE :  1.0763318643343687
MAPE :  34.49434555998239

<IPython.core.display.Javascript object>
```

4.4.6 Matrix Factorization Techniques

4.4.6.1 SVD Matrix Factorization User Movie intractions

In [0]:

```
from surprise import SVD
```

http://surprise.readthedocs.io/en/stable/matrix_factorization.html#surprise.prediction_algorithms.matrix_factorizati (http://surprise.readthedocs.io/en/stable/matrix_factorization.html#surprise.prediction_algorithms.matrix_factoriza

- __ Predicted Rating : __
  -
  - $\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$
    - $\boldsymbol{q}_i$ - Representation of item(movie) in latent factor space
    - $\boldsymbol{p}_u$ - Representation of user in new latent factor space

- A BASIC MATRIX FACTORIZATION MODEL in https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf (https://datajobs.com/data-science-repo/Recommender-Systems-%5BNetflix%5D.pdf)

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

In [50]:

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

# Just store these error metrics in our models_evaluation datastructure
models_evaluation_train['svd'] = svd_train_results
models_evaluation_test['svd'] = svd_test_results
```

```
Training the model...
Processing epoch 0
Processing epoch 1
Processing epoch 2
Processing epoch 3
Processing epoch 4
Processing epoch 5
Processing epoch 6
Processing epoch 7
Processing epoch 8
Processing epoch 9
Processing epoch 10
Processing epoch 11
Processing epoch 12
Processing epoch 13
Processing epoch 14
Processing epoch 15
Processing epoch 16
Processing epoch 17
Processing epoch 18
Processing epoch 19
Done. time taken : 0:00:09.778854

Evaluating the model with train data..
time taken : 0:00:01.560933
---------------
Train Data
---------------
RMSE : 0.6574721240954099

MAPE : 19.704901088660474

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.081963
---------------
Test Data
---------------
RMSE : 1.0726046873826458

MAPE : 35.01953535988152

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:00:11.424479
```

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

In [0]:

```
from surprise import SVDpp
```

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

- __ Predicted Rating : __
  -
  - $\large \hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j\right)$

- $\pmb{I_u}$ --- the set of all items rated by user u

- $\pmb{y_j}$ --- Our new set of item factors that capture implicit ratings.

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

In [52]:

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

# Just store these error metrics in our models_evaluation datastructure
models_evaluation_train['svdpp'] = svdpp_train_results
models_evaluation_test['svdpp'] = svdpp_test_results
```

```
Training the model...
 processing epoch 0
 processing epoch 1
 processing epoch 2
 processing epoch 3
 processing epoch 4
 processing epoch 5
 processing epoch 6
 processing epoch 7
 processing epoch 8
 processing epoch 9
 processing epoch 10
 processing epoch 11
 processing epoch 12
 processing epoch 13
 processing epoch 14
 processing epoch 15
 processing epoch 16
 processing epoch 17
 processing epoch 18
 processing epoch 19
Done. time taken : 0:02:59.274613

Evaluating the model with train data..
time taken : 0:00:06.705638
---------------
Train Data
---------------
RMSE : 0.6032438403305899

MAPE : 17.49285063490268

adding train results in the dictionary..

Evaluating for test data...
time taken : 0:00:00.081318
---------------
Test Data
---------------
RMSE : 1.0728491944183447

MAPE : 35.03817913919887

storing the test results in test dictionary...

---------------------------------------------
Total time taken to run this algorithm : 0:03:06.064030
```

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

Preparing Train data

In [53]:

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

reg_train.head(2)
```

Out[53]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	...	smr4	smr5	UAv
0	53406	33	3.581679	4.0	5.0	5.0	4.0	1.0	5.0	2.0	...	3.0	1.0	3.3703
1	99540	33	3.581679	5.0	5.0	5.0	4.0	5.0	3.0	4.0	...	3.0	5.0	3.5555

2 rows × 21 columns

__Preparing Test data __

In [54]:

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

reg_test_df.head(2)
```

Out[54]:

	user	movie	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	sı
0	808635	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581
1	941866	71	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581679	3.581

2 rows × 21 columns

In [56]:

```
import xgboost as xgb
# prepare x_train and y_train
x_train = reg_train.drop(['user', 'movie', 'rating',], axis=1)
y_train = reg_train['rating']

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



start = datetime.now()

# Initialize Our first XGBoost model
model = xgb.XGBRegressor(nthread=-1)

# Cross validation
gsv = GridSearchCV(model,
                    param_grid = parameters,
                    scoring="neg_mean_squared_error",
                    cv = TimeSeriesSplit(n_splits=5),
                    n_jobs = -1,
                    verbose = 1)
gsv_result = gsv.fit(x_train, y_train)

# Summarizing results
print("Best: %f using %s" % (gsv_result.best_score_, gsv_result.best_params_))
print()
means = gsv_result.cv_results_['mean_test_score']
stds = gsv_result.cv_results_['std_test_score']
params = gsv_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

print("\nTime Taken: ",datetime.now() - start)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  6.5min
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed: 34.1min finished

Best: -0.718429 using {'learning_rate': 0.1, 'max_depth': 3, 'n_estimator
s': 100}

-9.003600 (0.421070) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_est
imators': 100}
-6.396660 (0.323360) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_est
imators': 300}
-4.635431 (0.245576) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_est
imators': 500}
-3.444588 (0.195216) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_est
imators': 700}
-8.968829 (0.374920) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_est
imators': 100}
-6.330400 (0.267078) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_est
```

```
imators': 300}
-4.550069 (0.191141) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_est
imators': 500}
-3.349752 (0.140934) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_est
imators': 700}
-8.950769 (0.371915) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_est
imators': 100}
-6.284471 (0.244525) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_est
imators': 300}
-4.494928 (0.168086) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_est
imators': 500}
-3.290015 (0.117240) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_est
imators': 700}
-2.322729 (0.136005) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_esti
mators': 100}
-0.891127 (0.051646) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_esti
mators': 300}
-0.800698 (0.043965) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_esti
mators': 500}
-0.767167 (0.038827) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_esti
mators': 700}
-2.224923 (0.096179) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_esti
mators': 100}
-0.800935 (0.035338) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_esti
mators': 300}
-0.739654 (0.031972) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_esti
mators': 500}
-0.726522 (0.029603) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_esti
mators': 700}
-2.157529 (0.065650) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_esti
mators': 100}
-0.767935 (0.029759) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_esti
mators': 300}
-0.724250 (0.028990) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_esti
mators': 500}
-0.719172 (0.027883) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_esti
mators': 700}
-0.741297 (0.033748) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estim
ators': 100}
-0.721278 (0.027492) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estim
ators': 300}
-0.721165 (0.027519) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estim
ators': 500}
-0.721400 (0.027637) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estim
ators': 700}
-0.721041 (0.027456) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estim
ators': 100}
-0.719075 (0.026907) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estim
ators': 300}
-0.720134 (0.027696) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estim
ators': 500}
-0.721321 (0.028525) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estim
ators': 700}
-0.718429 (0.027198) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estim
ators': 100}
-0.719790 (0.028153) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estim
ators': 300}
-0.721935 (0.029614) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estim
ators': 500}
-0.724118 (0.031109) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estim
ators': 700}
```

```
Time Taken:  0:34:16.297832
```

In [57]:

```
xgb_final = xgb.XGBRegressor(max_depth=3,learning_rate = 0.1,n_estimators=100,nthread=-1)
xgb_final
```

Out[57]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=-1, objective='reg:linear', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

In [58]:

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

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


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

```
Training the model..
Done. Time taken : 0:00:11.354137

Done

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

TEST DATA
------------------------------
RMSE :  1.07637090548753
MAPE :  34.486102340513916

<IPython.core.display.Javascript object>
```

4.4.8 XgBoost with Surprise Baseline + Surprise KNNbaseline + MF Techniques

In [59]:

```
# 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']


start = datetime.now()

# Initialize Our first XGBoost model
model = xgb.XGBRegressor(nthread=-1)

# Cross validation
gsv = GridSearchCV(model,
                    param_grid = parameters,
                    scoring="neg_mean_squared_error",
                    cv = TimeSeriesSplit(n_splits=5),
                    n_jobs = -1,
                    verbose = 1)
gsv_result = gsv.fit(x_train, y_train)

# Summarizing results
print("Best: %f using %s" % (gsv_result.best_score_, gsv_result.best_params_))
print()
means = gsv_result.cv_results_['mean_test_score']
stds = gsv_result.cv_results_['std_test_score']
params = gsv_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

print("\nTime Taken: ",datetime.now() - start)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  3.0min
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed: 16.1min finished


Best: -1.164610 using {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators':
100}

-9.029998 (0.425032) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estim
ators': 100}
-6.473006 (0.352038) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estim
ators': 300}
-4.752577 (0.292264) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estim
ators': 500}
-3.594032 (0.243208) with: {'learning_rate': 0.001, 'max_depth': 1, 'n_estim
ators': 700}
-9.029755 (0.425032) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estim
ators': 100}
-6.472546 (0.352162) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estim
ators': 300}
-4.752052 (0.292632) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estim
ators': 500}
```

```
-3.593538 (0.243638) with: {'learning_rate': 0.001, 'max_depth': 2, 'n_estim
ators': 700}
-9.029730 (0.425077) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estim
ators': 100}
-6.472351 (0.352617) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estim
ators': 300}
-4.751680 (0.293461) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estim
ators': 500}
-3.593233 (0.244425) with: {'learning_rate': 0.001, 'max_depth': 3, 'n_estim
ators': 700}
-2.511971 (0.184904) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estima
tors': 100}
-1.200392 (0.064240) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estima
tors': 300}
-1.166743 (0.061822) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estima
tors': 500}
-1.164822 (0.062102) with: {'learning_rate': 0.01, 'max_depth': 1, 'n_estima
tors': 700}
-2.511541 (0.185349) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estima
tors': 100}
-1.200485 (0.064306) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estima
tors': 300}
-1.167093 (0.061770) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estima
tors': 500}
-1.165347 (0.062068) with: {'learning_rate': 0.01, 'max_depth': 2, 'n_estima
tors': 700}
-2.511381 (0.186006) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estima
tors': 100}
-1.200679 (0.064457) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estima
tors': 300}
-1.167366 (0.061877) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estima
tors': 500}
-1.165760 (0.062199) with: {'learning_rate': 0.01, 'max_depth': 3, 'n_estima
tors': 700}
-1.164610 (0.062180) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimat
ors': 100}
-1.164919 (0.062251) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimat
ors': 300}
-1.165147 (0.062277) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimat
ors': 500}
-1.165345 (0.062262) with: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimat
ors': 700}
-1.165375 (0.062224) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimat
ors': 100}
-1.167409 (0.062762) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimat
ors': 300}
-1.169432 (0.063276) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimat
ors': 500}
-1.171252 (0.063716) with: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimat
ors': 700}
-1.166189 (0.062513) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimat
ors': 100}
-1.170052 (0.063368) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimat
ors': 300}
-1.173600 (0.063807) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimat
ors': 500}
-1.177181 (0.064875) with: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimat
ors': 700}

Time Taken:  0:16:08.479523
```

In [60]:

```
xgb_all_models = xgb.XGBRegressor(max_depth=1,learning_rate = 0.01,n_estimators=100,nthreac
xgb_all_models
```

Out[60]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.01, max_delta_step=0,
       max_depth=1, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=-1, objective='reg:linear', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

In [61]:

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

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

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

```
Training the model..
Done. Time taken : 0:00:02.292256

Done

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

TEST DATA
------------------------------
RMSE :  1.5597123428477524
MAPE :  38.41308775628294

<IPython.core.display.Javascript object>
```

4.5 Comparision between all models

In [62]:

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

Out[62]:

```
bsl_algo          1.0725953793894922
svd               1.0726046873826458
knn_bsl_u         1.0726493739667242
knn_bsl_m          1.072758832653683
svdpp             1.0728491944183447
xgb_bsl           1.0762300237782054
xgb_knn_bsl       1.0763318643343687
xgb_final           1.07637090548753
xgb_all_models    1.5597123428477524
Name: rmse, dtype: object
```

Procedure

1. We have sampled the data because of less computational power. I sampled (25000,3000) training dataset and (9000,1500) testing dataset.
2. After sampling the data we featurized the data for regression and for surprise library.
3. After that we have applied XGBRegressor each time with different set of features generated after featurization with tuned hyperparameter.
4. At last we will compare the model performance using RMSE.

Conclusion

1. There is very small difference in 'RMSE' score, And this can be improved by using the whole dataset.
2. bsl_algo model is best among all the models we tried.