from surprise import SVD  
import numpy as np  
import surprise  
from surprise import Reader, Dataset  
# It is to specify how to read the data frame.  
reader = Reader(rating\_scale=(1,5))  
# create the traindata from the data frame  
train\_data\_mf = Dataset.load\_from\_df(train\_data[['userId', 'movieId', 'rating']], reader)  
# build the train set from traindata.  
#It is of dataset format from surprise library  
trainset = train\_data\_mf.build\_full\_trainset()  
svd = SVD(n\_factors=100, biased=True, random\_state=15, verbose=True)  
svd.fit(trainset)  
#getting predictions of train set  
train\_preds = svd.test(trainset.build\_testset())  
train\_pred\_mf = np.array([pred.est for pred in train\_preds])# Creating a sparse matrix  
train\_sparse\_matrix = sparse.csr\_matrix((train\_data.rating.values, (train\_data.userId.values, train\_data.movieId.values)))  
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  
Output: {‘global’: 3.5199769425298757}  
Next, let’s create a function which takes the sparse matrix as input and gives the average ratings of a movie given by all users, and the average rating of all movies given by a single user.  
# 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  
# create a dictionary 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  
train\_averages['user'] = get\_average\_ratings(train\_sparse\_matrix, of\_users=True)  
# compute the similar Users of the "user"  
user\_sim = cosine\_similarity(train\_sparse\_matrix[user], train\_sparse\_matrix).ravel()  
top\_sim\_users = user\_sim.argsort()[::-1][1:] # we are ignoring 'The User' from its similar users.  
# get the ratings of most similar users for this movie  
top\_ratings = 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([train\_averages['movie'][movie]]\*(5 -len(top\_sim\_users\_ratings)))  
# compute the similar movies of the "movie"  
movie\_sim = cosine\_similarity(train\_sparse\_matrix[:,movie].T,  
train\_sparse\_matrix.T).ravel()  
top\_sim\_movies = movie\_sim.argsort()[::-1][1:]  
# we are ignoring 'The User' from its similar users.  
# get the ratings of most similar movie rated by this user  
top\_ratings = 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([train\_averages['user'][user]]\*(5-len(top\_sim\_movies\_ratings)))  
# prepare train data  
x\_train = final\_data.drop(['user', 'movie','rating'], axis=1)  
y\_train = final\_data['rating']  
# initialize XGBoost model  
xgb\_model = xgb.XGBRegressor(silent=False, n\_jobs=13,random\_state=15,n\_estimators=100)  
# fit the model  
xgb\_model.fit(x\_train, y\_train, eval\_metric = 'rmse')