SGD Algorithm to predict movie ratings

There will be some functions that start with the word "grader" ex: grader_matrix(), grader_mean(), grader_dim() etc, you should not change those function definition.

Every Grader function has to return True.

```
1. Download the data from  here 

2. The data will be of this format, each data point is represented as a triplet of
   user_id, movie_id and rating
```

user_id	movie_id	rating
77	236	3
471	208	5
641	401	4
31	298	4
58	504	5
235	727	5

Task 1

Predict the rating for a given (user_id, movie_id) pair

Predicted rating $\hat{y}_{ij}$ for user i, movied j pair is calcuated as $\hat{y}_{ij} = \mu + b_i + c_j + u_i^T v_j$, here we will be finding the best values of b_i and c_j using SGD algorithm with the optimization problem for N users and M movies is defined as

$$ L = \min_{ b, c, { ui }{i=1}^N, { vj }{j=1}^M} \quad \alpha \Big(\sum{j} \sum{k} v_{jk}^2

```
+ \sum_{i} \sum_{k} u_{ik}^2
+ \sum_{i} b_i^2
+ \sum_{j} c_i^2
\Big)
```

- \sum{i,j \in \mathcal{I}^{\text{train}}} (y{ij} - \mu - b_i - c_j - u_i^T v_j)^2 $$

- μ : scalar mean rating
- b_i : scalar bias term for user i
- c_j : scalar bias term for movie j
- u_i : K-dimensional vector for user i
- v_j : K-dimensional vector for movie j

*. We will be giving you some functions, please write code in that functions only.

*. After every function, we will be giving you expected output, please make sure that you get that output.

1. Construct adjacency matrix with the given data, assuming its graph and the weight of each edge is the rating given by user to the movie

you can construct this matrix like $A[i][j] = r_{ij}$ here i is user_id, j is movieid and $r{ij}isratinggivenbyuser i tothemovie j$

Hint : you can create adjacency matrix using csr_matrix

1. We will Apply SVD decomposition on the Adjaceny matrix link1, link2 and get three matrices $U, \sum, V$ such that $U \times \sum \times V^T = A$,
   if A is of dimensions $N \times M$ then
   U is of $N \times k$,
   $\sum$ is of $k \times k$ and
   V is $M \times k$ dimensions.

   *. So the matrix U can be represented as matrix representation of users, where each row u_i represents a k-dimensional vector for a user

   *. So the matrix V can be represented as matrix representation of movies, where each row v_j represents a k-dimensional vector for a movie.
2. Compute μ , μ represents the mean of all the rating given in the dataset.(write your code in def m_u())
3. For each unique user initilize a bias value B_i to zero, so if we have N users B will be a N dimensional vector, the i^{th} value of the B will corresponds to the bias term for i^{th} user (write your code in def initialize())
4. For each unique movie initilize a bias value C_j zero, so if we have M movies C will be a M dimensional vector, the j^{th} value of the C will corresponds to the bias term for j^{th} movie (write your code in def initialize())
5. Compute dL/db_i (Write you code in def derivative_db())
6. Compute dL/dc_j(write your code in def derivative_dc()
7. Print the mean squared error with predicted ratings.

```
for each epoch:
    for each pair of (user, movie):
        b_i =  b_i - learning_rate * dL/db_i
        c_j =  c_j - learning_rate * dL/dc_j
predict the ratings with formula
```

$\hat{y}_{ij} = \mu + b_i + c_j + \text{dot\_product}(u_i, v_j)$

1. you can choose any learning rate and regularization term in the range 10^{-3} to 10^2
2. **bonus**: instead of using SVD decomposition you can learn the vectors u_i, v_j with the help of SGD algo similar to b_i and c_j

Task 2

As we know U is the learned matrix of user vectors, with its i-th row as the vector ui for user i. Each row of U can be seen as a "feature vector" for a particular user.

The question we'd like to investigate is this: do our computed per-user features that are optimized for predicting movie ratings contain anything to do with gender?

The provided data file user_info.csv contains an is_male column indicating which users in the dataset are male. Can you predict this signal given the features U?

> **Note 1** : there is no train test split in the data, the goal of this assignment is to give an intution about how to do matrix factorization with the help of SGD and application of truncated SVD. for better understanding of the collabarative fillerting please check netflix case study.
>
> **Note 2** : Check if scaling of U, V matrices improve the metric

Reading the csv file

In [1]:
```
import pandas as pd
data=pd.read_csv('ratings_train.csv')
data.head()
```

Out[1]:

	user_id	item_id	rating
0	772	36	3
1	471	228	5
2	641	401	4
3	312	98	4
4	58	504	5

In [2]:
```
data.shape
```

Out[2]: (89992, 3)

Create your adjacency matrix

In [3]:
```
from scipy.sparse import csr_matrix
adjacency_matrix = csr_matrix((data.rating.values,(data.user_id.values,data.item_id.values
))).toarray()
```

In [4]:
```
adjacency_matrix.shape
```

Out[4]: (943, 1681)

Grader function - 1

In [5]:
```
def grader_matrix(matrix):
  assert(matrix.shape==(943,1681))
  return True
grader_matrix(adjacency_matrix)
```

Out[5]: True

The unique items in the given csv file are 1662 only . But the id's vary from 0-1681 but they are not continuous and hence you'll get matrix of size 943x1681.

SVD decompostion

Sample code for SVD decompostion

In [6]:
```
from sklearn.utils.extmath import randomized_svd
import numpy as np
matrix = np.random.random((20, 10))
U, Sigma, VT = randomized_svd(matrix, n_components=5,n_iter=5, random_state=None)
print(U.shape)
print(Sigma.shape)
print(VT.T.shape)
```

```
(20, 5)
(5,)
(10, 5)
```

Write your code for SVD decompostion

In [7]:
```
# Please use adjacency_matrix as matrix for SVD decompostion
# You can choose n_components as your choice
U, Sigma, VT = randomized_svd(adjacency_matrix ,n_components=10, n_iter = 5, random_state=No
ne)
print(U.shape)
print(Sigma.shape)
print(VT.T.shape)
```

```
(943, 10)
(10,)
(1681, 10)
```

Compute mean of ratings

In [14]:
```
def m_u(ratings):
    '''In this function, we will compute mean for all the ratings'''
    # you can use mean() function to do this
    # check this (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFram
e.mean.html) link for more details.
    mean = pd.DataFrame.mean(ratings)

    return mean
```

In [15]:
```
mu=m_u(data['rating'])
print(mu)
```

```
3.529480398257623
```

Grader function -2

In [16]:
```
def grader_mean(mu):
  assert(np.round(mu,3)==3.529)
  return True
mu=m_u(data['rating'])
grader_mean(mu)
```

Out[16]: True

Initialize B_i and C_j

Hint : Number of rows of adjacent matrix corresponds to user dimensions(B_i), number of columns of adjacent matrix corresponds to movie dimensions (C_j)

In [25]:
```
def initialize(dim):
    '''In this function, we will initialize bias value 'B' and 'C'.'''
    # initalize the value to zeros
    # return output as a list of zeros
    z = np.zeros(dim)


    return z
```

In [26]:
```
dim = 943 # give the number of dimensions for b_i (Here b_i corresponds to users)
b_i=initialize(dim)
```

In [27]:
```
dim = 1681 # give the number of dimensions for c_j (Here c_j corresponds to movies)
c_j=initialize(dim)
```

Grader function -3

In [28]:
```
def grader_dim(b_i,c_j):
  assert(len(b_i)==943 and np.sum(b_i)==0)
  assert(len(c_j)==1681 and np.sum(c_j)==0)
  return True
grader_dim(b_i,c_j)
```

Out[28]: True

Compute dL/db_i

In [31]:
```
def derivative_db(user_id,item_id,rating,U,V,mu,alpha):
    '''In this function, we will compute dL/db_i'''
    db = 2*alpha*b_i[user_id] - 2*(rating - mu - b_i[user_id] - c_j[item_id] - np.dot(U[user
_id].T,V[:,item_id]))
    return db
```

Grader function -4

In [32]:
```
def grader_db(value):
    assert(np.round(value,3)==-0.931)
    return True
U1, Sigma, V1 = randomized_svd(adjacency_matrix, n_components=2,n_iter=5, random_state=24)
# Please don't change random state
# Here we are considering n_componets = 2 for our convinence
alpha=0.01
value=derivative_db(312,98,4,U1,V1,mu,alpha)
grader_db(value)
```

Out[32]: True

Compute dL/dc_j

In [38]:
```
def derivative_dc(user_id,item_id,rating,U,V,mu, alpha):
    '''In this function, we will compute dL/dc_j'''
    dc = 2*alpha*c_j[item_id] - 2*(rating - mu - b_i[user_id] - c_j[item_id] - np.dot(U[user
_id].T,V[:,item_id]))
    return dc
```

Grader function - 5

In [39]:
```
def grader_dc(value):
    assert(np.round(value,3)==-2.929)
    return True
U1, Sigma, V1 = randomized_svd(adjacency_matrix, n_components=2,n_iter=5, random_state=24)
# Please don't change random state
# Here we are considering n_componets = 2 for our convinence
r=0.01
value=derivative_dc(58,504,5,U1,V1,mu,alpha)
grader_dc(value)
```

Out[39]: True

Compute MSE (mean squared error) for predicted ratings

for each epoch, print the MSE value

```
for each epoch:

    for each pair of (user, movie):

        b_i =  b_i - learning_rate * dL/db_i

        c_j =  c_j - learning_rate * dL/dc_j

predict the ratings with formula
```

$\hat{y}_{ij} = \mu + b_i + c_j + \text{dot\_product}(u_i, v_j)$

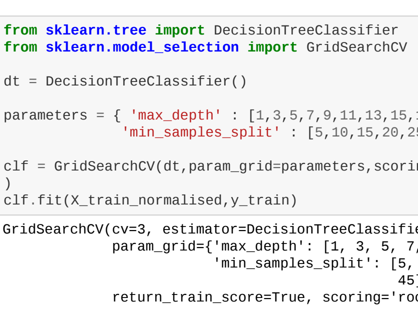
In [42]:
```
from sklearn.metrics import mean_squared_error

ratings = list(data['rating'])
user = list(data['user_id'])
movie = list(data['item_id'])
epochs = 15
learning_rate = 0.1
b_i = initialize(943)
c_j = initialize(1681)
mse = []
for epoch in range(1,epochs):
    y_pred=[]
    for i,j,k in zip(user,movie,ratings):
        b_i[i] = b_i[i] - learning_rate*derivative_db(i,j,k,U,VT,mu,alpha)
        c_j[j] = c_j[j] - learning_rate*derivative_dc(i,j,k,U,VT,mu, alpha)
        pred = mu + b_i[i] + c_j[j] + np.dot(U[i],VT.T[j])
        y_pred.append(pred)

    m = mean_squared_error(ratings,y_pred)
    mse.append(m)
    print('Epoch Number = ',epoch, " MSE = ",m)
```

```
Epoch Number =  1  MSE =  0.4327679167960899
Epoch Number =  2  MSE =  0.42423005409439357
Epoch Number =  3  MSE =  0.42364126450556383
Epoch Number =  4  MSE =  0.42339941570475276
Epoch Number =  5  MSE =  0.42326920056848666
Epoch Number =  6  MSE =  0.42318976568872463
Epoch Number =  7  MSE =  0.423138022614376
Epoch Number =  8  MSE =  0.42310301308544
Epoch Number =  9  MSE =  0.42307875074017
Epoch Number =  10  MSE =  0.42306165911399424
Epoch Number =  11  MSE =  0.4230494731868743
Epoch Number =  12  MSE =  0.4230407014679118
Epoch Number =  13  MSE =  0.42303433530136864
Epoch Number =  14  MSE =  0.4230296797554165
```

Plot epoch number vs MSE

- epoch number on X-axis
- MSE on Y-axis

In [48]:
```
import matplotlib.pyplot as plt
epoch = list(range(0,14))
plt.plot(epoch,mse,label='y_mse')
plt.xlabel("epoch")
plt.ylabel("MSE")
plt.title("MSE vs epoch")
plt.legend()
plt.grid()
plt.show()
```



Task 2

- For this task you have to consider the user_matrix U and the user_info.csv file.
- You have to consider is_male columns as output features and rest as input features. Now you have to fit a model by posing this problem as binary classification task.
- You can apply any model like Logistic regression or Decision tree and check the performance of the model.
- Do plot confusion matrix after fitting your model and write your observations how your model is performing in this task.
- Optional work- You can try scaling your U matrix.Scaling means changing the values of n_componenets while performing svd and then check your results.

In [50]:
```
user_info = pd.read_csv('user_info.csv.txt')
user_info.head(3)
```

Out[50]:

	user_id	age	is_male	orig_user_id
0	0	24	1	1
1	1	53	0	2
2	2	23	1	3

In [51]:
```
input_features = pd.DataFrame(U)
```

Splitting the dataset:

In [52]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(input_features ,user_info.is_male ,test_siz
e = 0.3 ,random_state=25)
```

In [54]:
```
print(len(X_train))
print(len(X_test))
```

```
660
283
```

Normalizing the data:

In [56]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

X_train_normalised = scaler.fit_transform(X_train)
X_test_normalised  = scaler.transform(X_test)
```

Applying Decision Trees for classification :

Hyperparameter Tuning :

In [70]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

dt = DecisionTreeClassifier()

parameters = { 'max_depth' : [1,3,5,7,9,11,13,15,17],
              'min_samples_split' : [5,10,15,20,25,30,35,40,45]}

clf = GridSearchCV(dt,param_grid=parameters,scoring = 'roc_auc',cv=3,return_train_score=True
)
clf.fit(X_train_normalised,y_train)
```

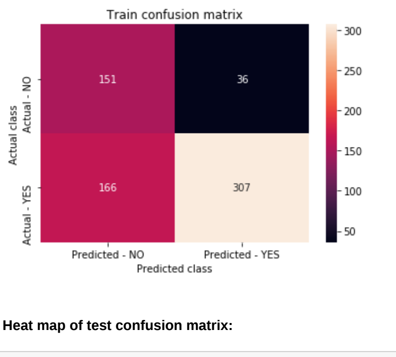
Out[70]:
```
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [1, 3, 5, 7, 9, 11, 13, 15, 17],
                         'min_samples_split': [5, 10, 15, 20, 25, 30, 35, 40,
                                               45]},
             return_train_score=True, scoring='roc_auc')
```

In [71]:
```
clf.best_params_
```

Out[71]: {'max_depth': 5, 'min_samples_split': 40}

ROC_AUC plot:

In [72]:
```
from sklearn.metrics import roc_curve, auc

model = DecisionTreeClassifier(max_depth=5,min_samples_split=40,class_weight='balanced')
model.fit(X_train_normalised,y_train)

y_train_pred = model.predict_proba(X_train_normalised)[:,1]
y_test_pred = model.predict_proba(X_test_normalised)[:,1]

train_fpr,train_tpr,train_threshold = roc_curve(y_train,y_train_pred)
test_fpr,test_tpr,test_threshold = roc_curve(y_test,y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Confusion Matrix:

In [73]:
```
def best_threshold(threshold,fpr,tpr):
    t=threshold[np.argmax(tpr*(1-fpr))]
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,
3))
    return t

def predict_best_t(proba,threshold):
    predictions=[]
    global y_pred
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)
    y_pred=predictions
    return predictions
```

In [74]:
```
from sklearn.metrics import confusion_matrix
best_t= best_threshold(train_threshold,train_fpr,train_tpr)
print('Train confusion matrix')
cm_train=confusion_matrix(y_train,predict_best_t(y_train_pred,best_t))
print(cm_train)
print('Test confusion matrix')
cm_test=confusion_matrix(y_test,predict_best_t(y_test_pred,best_t))
print(cm_test)
```

```
the maximum value of tpr*(1-fpr) 0.5240980882070299 for threshold 0.493
Train confusion matrix
[[151  36]
 [166 307]]
Test confusion matrix
[[ 65  21]
 [ 88 109]]
```

Heat map of train confusion matrix:

In [76]:
```
import seaborn as sns
ax=plt.subplot()
sns.heatmap(cm_train,xticklabels=['Predicted - NO','Predicted - YES'],yticklabels=['Actual -
NO','Actual - YES'],annot=True,fmt='d')
ax.set_title('Train confusion matrix')
ax.set_xlabel('Predicted class')
ax.set_ylabel('Actual class')
```

Out[76]: Text(33,0.5,'Actual class')

Heat map of test confusion matrix:

In [77]:
```
ax=plt.subplot()
sns.heatmap(cm_test,xticklabels=['Predicted - NO','Predicted - YES'],yticklabels=['Actual -
 NO','Actual - YES'],annot=True,fmt='d')
ax.set_title('Test confusion matrix')
ax.set_xlabel('Predicted class')
ax.set_ylabel('Actual class')
```

Out[77]: Text(33,0.5,'Actual class')

Observation:

After applying the Decision Tree model, we can see that from AUC values the model is performing better than a simple random model. Hence we can use the features to determine the gender.