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

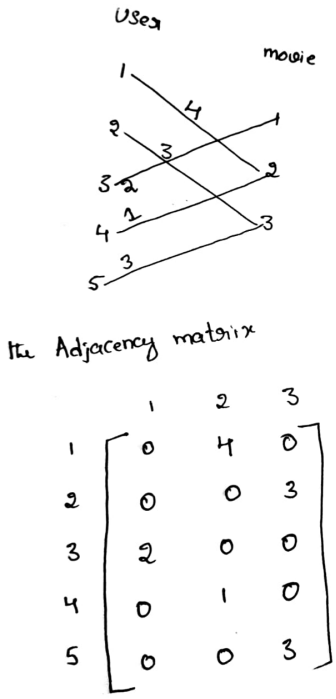
$$L = \min_{b,c,\{u_i\}_{i=1}^{N},\{v_j\}_{j=1}^{M}} \alpha \left(\right.$$

- 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 weighted un-directed bi-partited 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 movie***id and $r{ij}*$* $is rating given by user i to the movie j$**

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]:

```
print ("Unique users :",len(data['user_id'].unique()))
print ("Unique movies :",len(data['item_id'].unique()))
```

```
Unique users : 943
Unique movies : 1662
```

In [4]:

```
u_i  = data['user_id'].tolist()
v_j  = data['item_id'].tolist()
r_ij = data['rating'].tolist()
```

In [5]:

```
from scipy.sparse import csr_matrix
adjacency_matrix = csr_matrix((r_ij,(u_i,v_j))).toarray()
```

In [6]:

```
adjacency_matrix.shape
```

Out[6]:

```
(943, 1681)
```

Grader function - 1

In [7]:

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

Out[7]:

```
True
```

SVD decompostion

Sample code for SVD decompostion

In [8]:

```
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 [9]:

```
U, Sigma, VT = randomized_svd(adjacency_matrix, n_components=5,n_iter=5, random_state=No
ne)
print(U.shape)
print(Sigma.shape)
print(VT.T.shape)
```

```
(943, 5)
```

```
(943, 5)
(5,)
(1681, 5)
```

Compute mean of ratings

In [10]:

```
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.DataFrame.mean.html) link for more details.


    return np.mean(r_ij)
```

In [11]:

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

```
3.529480398257623
```

Grader function -2

In [12]:

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

Out[12]:

```
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 [13]:

```
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

    B = np.zeros(dim)

    return B
```

In [14]:

```
dim=adjacency_matrix.shape[0] # give the number of dimensions for b_i (Here b_i corresponds to users)
b_i=initialize(dim)
```

In [15]:

```
dim= adjacency_matrix.shape[1] # give the number of dimensions for c_j (Here c_j corresponds to movies)
c_j=initialize(dim)
```

Grader function -3

In [16]:

```
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[16]:

```
True
```

Compute dL/db_i

$$L = \min_{b,c,\{u_i\}_{i=1}^{N},\{v_j\}_{j=1}^{M}} \alpha \qquad \Big($$

In [17]:

```
def derivative_db(user_id,item_id,rating,U,V,mu,alpha):

    '''In this function, we will compute dL/db_i'''
    first_term  = 2*alpha*b_i[user_id]
    second_term = -(2*(rating-mu-b_i[user_id]-c_j[item_id]-np.dot(U[user_id],np.transpos
e(V)[item_id])))
    dL_db_i =  first_term + second_term

    return dL_db_i
```

Grader function -4

In [18]:

```
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=2
4)
# 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[18]:

```
True
```

Compute dL/dc_j

$$L = \min_{b,c,\{u_i\}_{i=1}^{N},\{v_j\}_{j=1}^{M}} \alpha \qquad \Big($$

In [19]:

```
def derivative_dc(user_id,item_id,rating,U,V,mu, alpha):
    '''In this function, we will compute dL/dc_j'''
    first_term =  2*alpha*c_j[item_id]
    second_term = -(2*(rating-mu-b_i[user_id]-c_j[item_id]-np.dot(U[user_id],np.transpos
e(V)[item_id])))
    dL_dc_j = first_term + second_term
    return dL_dc_j
```

Grader function - 5

In [20]:

```
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=2
4)
# Please don't change random state
# Here we are considering n_componets = 2 for our convinence
alpha=0.01
value=derivative_dc(58,504,5,U1,V1,mu,alpha)
grader_dc(value)
```

Out[20]:

```
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 [21]:

```
np.array(data)[0]
```

Out[21]:

```
array([772,  36,   3], dtype=int64)
```

In [22]:

```
from sklearn.metrics import mean_squared_error

def compute_mse(data,U,V,mu,alpha):
    y_true = data['rating']
    mse = []
    for epoch in range(0,50):
        for each_data_point in np.array(data):
            b_i[each_data_point[0]] = b_i[each_data_point[0]] - 0.001 * derivative_db(ea
ch_data_point[0],each_data_point[1],each_data_point[2],U,V,mu,alpha)
            c_j[each_data_point[1]] = c_j[each_data_point[1]] - 0.001 * derivative_dc(ea
ch_data_point[0],each_data_point[1],each_data_point[2],U,V,mu,alpha)

        y_pred = []
        for each_data_point in np.array(data):
            y_pred.append(mu + b_i[each_data_point[0]] + c_j[each_data_point[1]] + np.do
t(U[each_data_point[0]], V.T[each_data_point[1]]))

        loss = mean_squared_error(y_true, y_pred)
```

```
        mse.append(loss)
        if epoch%10 == 0:
            print("MSE: ",loss,"at  epoch: ",epoch)
    return mse
```

In [23]:

```
alpha=0.01
mse = compute_mse(data,U1,V1,mu,alpha)
```

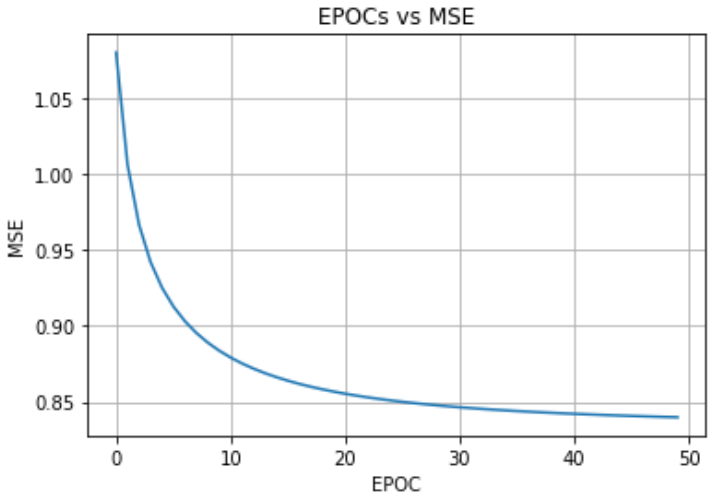
```
MSE:  1.0797989802961563 at  epoch:  0
MSE:  0.8790872671841912 at  epoch:  10
MSE:  0.8552947912172346 at  epoch:  20
MSE:  0.8463986335346735 at  epoch:  30
MSE:  0.842069050149986 at  epoch:  40
```

Plot epoch number vs MSE

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

In [24]:

```
import matplotlib.pyplot as plt
epoc=np.array(list(range(0,50)))
plt.plot(epoc,mse)
plt.xlabel("EPOC")
plt.ylabel("MSE")
plt.title("EPOCs vs MSE")
plt.grid()
plt.show()
```



In [25]:

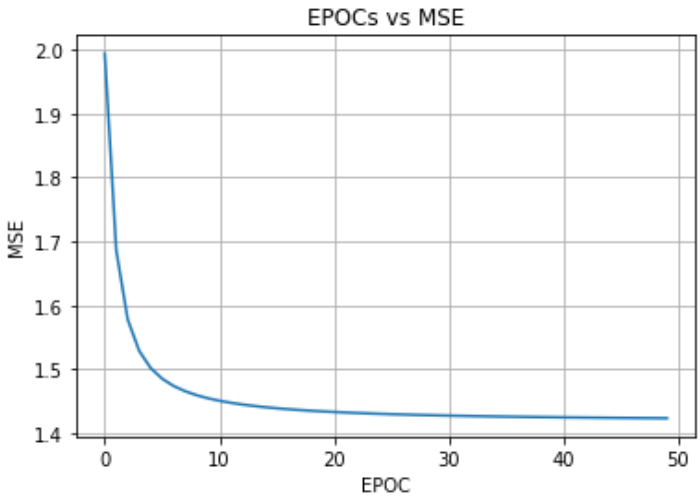
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
U1_scaled = scaler.fit_transform(U1)
V1_scaled = scaler.fit_transform(V1)

alpha=0.01
mse_scaled = compute_mse(data,U1_scaled,V1_scaled,mu,alpha)

epoc=np.array(list(range(0,50)))
plt.plot(epoc,mse_scaled)
plt.xlabel("EPOC")
plt.ylabel("MSE")
plt.title("EPOCs vs MSE")
plt.grid()
plt.show()
```

```
MSE:  1.0926460232757198 at  epoch:  0
```

```
MSE:  1.9936460232757198 at  epoch:  0
MSE:  1.451115058101147 at  epoch:  10
MSE:  1.4335822384462036 at  epoch:  20
MSE:  1.4279195542191574 at  epoch:  30
MSE:  1.4252166277566716 at  epoch:  40
```



Task 2

In [26]:

```
data_info = pd.read_csv('user_info.csv.txt')
data_info.head()
```

Out[26]:

	user_id	age	is_male	orig_user_id
0	0	24	1	1
1	1	53	0	2
2	2	23	1	3
3	3	24	1	4
4	4	33	0	5

In [27]:

```
X = pd.DataFrame(data = U1)
X.head()
```

Out[27]:

	0	1
0	0.066226	0.007888
1	0.013644	-0.048895
2	0.005438	-0.025128
3	0.005704	-0.018211
4	0.034122	0.009005

In [28]:

```
y = data_info['is_male']
```

In [29]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.3, random_state=0
```

```
)
```

In [30]:

```
clf = LogisticRegression(random_state=0)
```

In [31]:

```
clf.fit(X_train,y_train)
```

Out[31]:

```
LogisticRegression(random_state=0)
```

In [32]:

```
y_train_pred = clf.predict(X_train)
y_test_pred = clf.predict(X_test)
```

In [42]:

```
from sklearn.metrics import confusion_matrix
C_train = confusion_matrix(y_train, y_train_pred)
C_train
```

Out[42]:

```
array([[  0, 194],
       [  0, 466]], dtype=int64)
```

In [43]:

```
C_test = confusion_matrix(y_test, y_test_pred)
C_test
```

Out[43]:

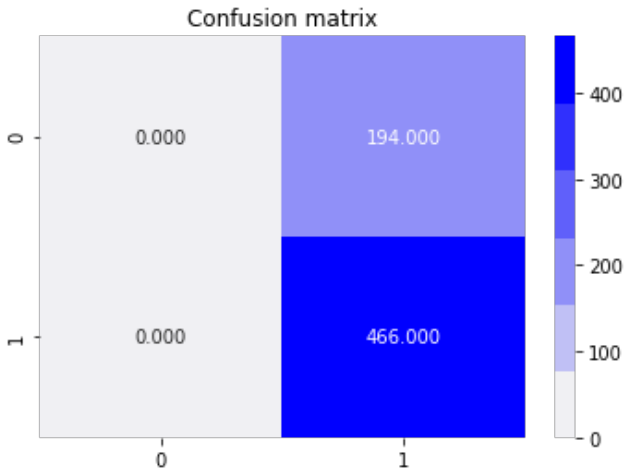
```
array([[  0,  79],
       [  0, 204]], dtype=int64)
```

In [45]:

```
plt.figure(figsize=(20,4))
labels = [0,1]
cmap=sns.light_palette("blue")
plt.subplot(1, 3, 1)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.title("Train Confusion matrix")
sns.heatmap(C_train, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=l
abels)
```

Out[45]:

```
<AxesSubplot:title={'center':'Confusion matrix'}>
```

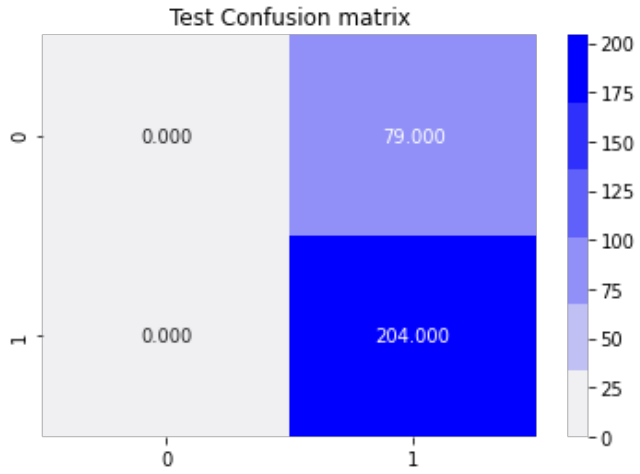


In [46]:

```
plt.title("Test Confusion matrix")
sns.heatmap(C_test, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=la
bels)
```

Out[46]:

```
<AxesSubplot:title={'center':'Test Confusion matrix'}>
```



In [ ]: