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

$$L = \min_{b, c, \{u_i\}_{i=1}^N, \{v_j\}_{j=1}^M} \alpha\Big(\sum_j \sum_k v_{jk}^2 + \sum_i \sum_k u_{ik}^2 + \sum_i b_i^2 + \sum_j c_i^2\Big) + \sum_{i,j \in I^{\text{train}}} (y_{ij} - \mu - b_i - c_j - u_i^T v_j)^2$$

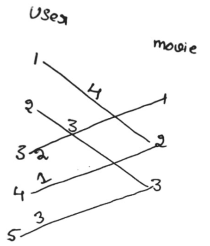
- μ : scalar mean rating
- b_i : scalar bias term for user i
- c_j : scalar bias term for movie j
- u_i : K-dimensional vector for user i
- v_j

: 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



the Adjacency matrix

$$\begin{array}{c|c} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 4 & 0 \\ 0 & 0 & 3 \\ 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{bmatrix} \end{array}$$

you can construct this matrix like $A[i][j] = r_{ij}$
here i
is user_id, j
is movie*id 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
V
can be represented as matrix representation of movies, where each row v_j
v_j
represents a k-dimensional vector for a movie.

2. Compute μ
μ
, 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
B_i
to zero, so if we have N
N
users B
B
will be a N
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

Libraries and Python Version used

```
Python 3.10.8

pip install -u matplotlib==3.6.2
pip install -u numpy==1.23.5
pip install -u pandas==1.5.2
pip install -u scikit-learn==1.1.3
pip install -u scipy==1.9.3
pip install -u seaborn==0.12.1
pip install -u tqdm==4.64.1
```

In [1]:

```
# Importing libraries

import numpy as np
import pandas as pd
import seaborn as sns
from tqdm import tqdm

import matplotlib.pyplot as plt

from scipy.sparse import csr_matrix

from sklearn.utils.extmath import randomized_svd
```

Reading the csv file

In [2]:

```
data = pd.read_csv('ratings_train.csv')
data.head()
```

Out[2]:

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 [3]:
```
data.shape
```
Out[3]:
```
(89992, 3)
```

Create your adjacency matrix

https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html

```
>>> row = np.array([0, 1, 2, 0])

>>> col = np.array([0, 1, 1, 0])

>>> data = np.array([1, 2, 4, 8])

>>> csr_matrix((data, (row, col)), shape=(3, 3)).toarray()

array([[9, 0, 0],
       [0, 2, 0],
       [0, 4, 0]])
```

In [4]:
```
# https://analyticsindiamag.com/singular-value-decomposition-svd-application-recommender-system/

rows = np.array(data.user_id.values)
colums = np.array(data.item_id.values)

values = np.array(data['rating'])

adjacency_matrix = csr_matrix((values, (rows, colums)), shape = (943, 1681)).toarray()

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]:
```
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]:
```python
# https://scikit-learn.org/stable/modules/generated/sklearn.utils.extmath.randomized_svd.html

n_comp = data.user_id.nunique()
U, Sigma, VT = randomized_svd(adjacency_matrix, n_components = n_comp, n_iter = 5, random_state = None)
```

Compute mean of ratings

In [8]:
```python
def m_u(ratings):
    '''In this function, we will compute mean for all the ratings'''

    return ratings.mean()
```

In [9]:
```python
mu = m_u(data['rating'])

print(mu)
```

```
3.529480398257623
```

Grader function -2

In [10]:
```python
def grader_mean(mu):

    assert(np.round(mu,3)==3.529)
    return True

mu = m_u(data['rating'])

grader_mean(mu)
```

Out[10]:
```
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 [11]:
```python
def initialize(dim):

    '''In this function, we will initialize bias value 'B' and 'C'.'''

    return np.zeros(dim)
```

In [12]:
```python
dim = adjacency_matrix.shape[0]
b_i = initialize(dim)
```

In [13]:
```python
dim = adjacency_matrix.shape[1]
c_j = initialize(dim)
```

Grader function -3

In [14]:
```python
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[14]:
```
True
```

Compute dL/db_i

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

$$L = \min_{b, c, \{u_i\}_{i=1}^N, \{v_j\}_{j=1}^M} \alpha\Big(\sum_j \sum_k v_{jk}^2 + \sum_i \sum_k u_{ik}^2 + \sum_i b_i^2 + \sum_j c_i^2\Big) + \sum_{i,j \in \mathcal{I}^{\text{train}}} (y_{ij} - \mu - b_i - c_j - u_i^T v_j)^2$$

In [15]:
```
def derivative_db(user_id, item_id, rating, U, V, mu, alpha):

    '''In this function, we will compute dL/db_i'''

    db = (alpha * 2 * b_i[user_id]) - 2 * (rating - mu - b_i[user_id] - c_j[item_id] \
                                               - np.dot(U[user_id], V.T[item_id]))

    return db
```

Grader function -4

In [16]:
```
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[16]: True

Compute dL/dc_j

$$L = \min_{b, c, \{u_i\}_{i=1}^N, \{v_j\}_{j=1}^M} \alpha\Big(\sum_j \sum_k v_{jk}^2 + \sum_i \sum_k u_{ik}^2 + \sum_i b_i^2 + \sum_j c_i^2\Big) + \sum_{i,j \in \mathcal{I}^{\text{train}}} (y_{ij} - \mu - b_i - c_j - u_i^T v_j)^2$$

In [17]:
```
def derivative_dc(user_id, item_id, rating, U, V, mu, alpha):

    '''In this function, we will compute dL/dc_j'''

    dc = (alpha * 2 * c_j[item_id]) - 2 * (rating - mu - b_i[user_id] - c_j[item_id] \
                                               - np.dot(U[user_id], V.T[item_id]))

    return dc
```

Grader function - 5

In [18]:
```
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[18]: 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 [19]:

```python
# https://www.geeksforgeeks.org/python-mean-squared-error/

def MSE_calc(actual, pred):

    return np.square(np.subtract(actual, pred)).mean()
```

In [20]:

```python
mse_list = []
lr = 0.01
from tqdm.notebook import tqdm
for epoch in tqdm(range(50)):

    Y_Hats = []
    y_hat = 0

    for idx in data.values.tolist():

        b_i[idx[0]] = b_i[idx[0]] - (lr * derivative_db(idx[0], idx[1], idx[2], U1, V1, mu, alpha))
        c_j[idx[1]] = c_j[idx[1]] - (lr * derivative_dc(idx[0], idx[1], idx[2], U1, V1, mu, alpha))

    for id_ in data.values:

        y_hat = mu + b_i[id_[0]] + c_j[id_[1]] + np.dot(U[id_[0]], VT.T[id_[1]])
        Y_Hats.append(y_hat)

    mse_value = MSE_calc(data.rating, np.array(Y_Hats))

    if (epoch+1)%5 == 0:
        print(f'MSE at epoch-{epoch+1:02} :: {round(mse_value, 4):04}')
    mse_list.append(mse_value)
```

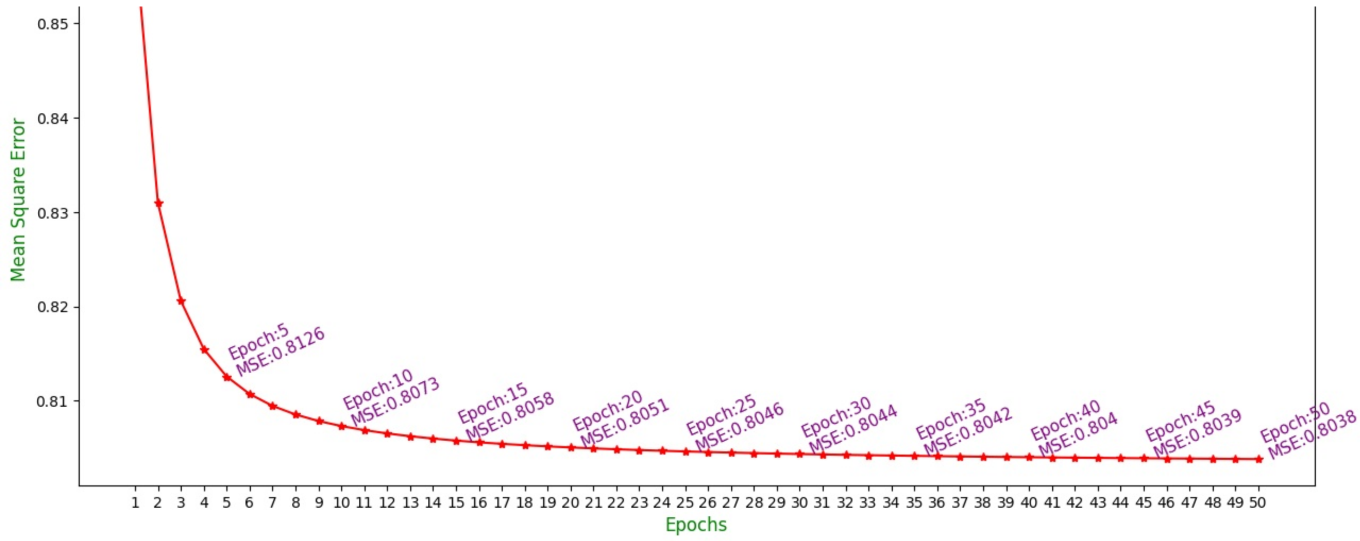
```
MSE at epoch-05 :: 0.8126
MSE at epoch-10 :: 0.8073
MSE at epoch-15 :: 0.8058
MSE at epoch-20 :: 0.8051
MSE at epoch-25 :: 0.8046
MSE at epoch-30 :: 0.8044
MSE at epoch-35 :: 0.8042
MSE at epoch-40 :: 0.804
MSE at epoch-45 :: 0.8039
MSE at epoch-50 :: 0.8038
```

Plot epoch number vs MSE

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

In [21]:

```python
# https://matplotlib.org/3.5.1/api/_as_gen/matplotlib.pyplot.annotate.html

plt.figure(figsize = (15, 7))
plt.plot(range(1,51), mse_list, c = 'r', marker= '*')
plt.title('Epoch vs MSE Curve', c = 'b', fontsize = 15)
plt.xticks(range(1,51))
plt.xlabel('Epochs', c = 'g', fontsize = 12)
plt.ylabel('Mean Square Error', c = 'g', fontsize = 12)

for mse, epo in zip(mse_list, range(1,51)):
    if epo % 5 == 0 or epo == 1:
        plt.annotate(f'Epoch:{epo}\nMSE:{round(mse,4)}', (epo, mse), c = 'purple', fontsize = 11, rotation = 25)

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.

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

In [22]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

In [23]:

```
data_with_age = pd.read_csv('user_info.csv')
data_with_age.head()
```

Out[23]:

	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 [24]:

```
data_with_age['U_1'] = U1.T[0]
data_with_age['U_2'] = U1.T[1]
```

```
data_with_age.head()
```

Out[24]:

	user_id	age	is_male	orig_user_id	U_1	U_2
0	0	24	1	1	0.066226	0.007888
1	1	53	0	2	0.013644	-0.048895
2	2	23	1	3	0.005438	-0.025128
3	3	24	1	4	0.005704	-0.018211
4	4	33	0	5	0.034122	0.009005

In [25]:

```
y = data_with_age.is_male
data_with_age.drop(['is_male','user_id', 'user_id'], axis = 1, inplace = True)

data_with_age.head()
```

Out[25]:

	age	orig_user_id	U_1	U_2
0	24	1	0.066226	0.007888
1	53	2	0.013644	-0.048895
2	23	3	0.005438	-0.025128
3	24	4	0.005704	-0.018211
4	33	5	0.034122	0.009005

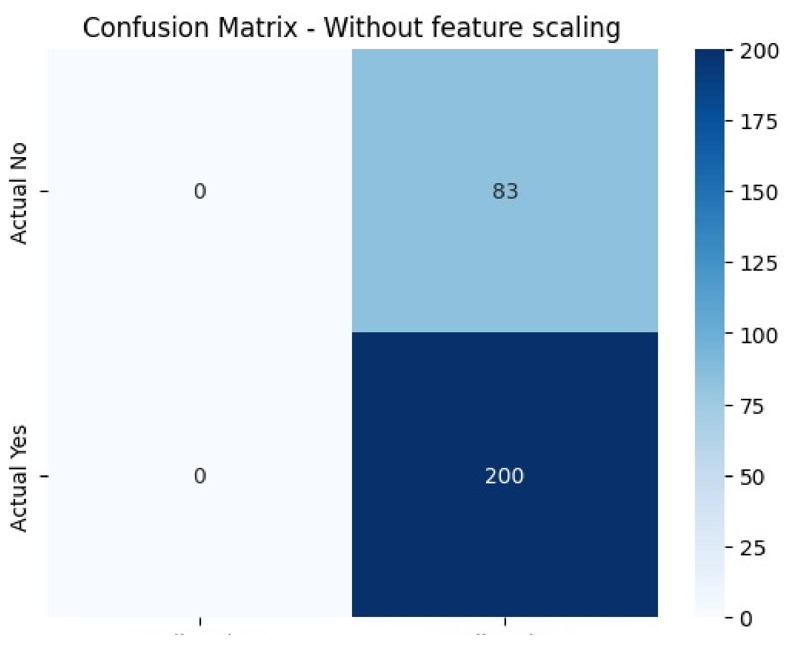
In [26]:

```
def train_nd_plot(df, y_label, text):

    X_train, X_test, y_train, y_test = train_test_split(df, y_label, test_size = 0.3, random_state = 42)

    lr_model = LogisticRegression(random_state = 52)
    lr_model.fit(X_train, y_train)
    y_pred = lr_model.predict(X_test)

    cn_matrix = confusion_matrix(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)

    print(f'\nResultant Mean Square Error : {text} :: {round(mse, 4)}\n')

    # https://stackoverflow.com/a/48018785
    ax= plt.subplot()
    sns.heatmap(cn_matrix, annot = True, fmt = 'd', cmap = 'Blues', ax = ax)
    ax.set_title(f'Confusion Matrix - {text}')
    ax.xaxis.set_ticklabels(['Predicted No', 'Predicted Yes'])
    ax.yaxis.set_ticklabels(['Actual No', 'Actual Yes'])
    plt.show()
```

In [27]:

```
train_nd_plot(data_with_age, y, 'Without feature scaling')
```

```
Resultant Mean Square Error : Without feature scaling :: 0.2933
```



Predicted No Predicted Yes

In [28]:

```
# MinMax Scaling on 'age'

age_max = data_with_age.age.max()
age_min = data_with_age.age.min()

data_with_age['age'] = (data_with_age.age - age_min) / (age_max - age_min)
```

In [29]:

```
train_nd_plot(data_with_age, y, 'With freature scaling')
```

```
Resultant Mean Square Error : With freature scaling :: 0.2933
```