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 (https://drive.google.com/open?id=1-1z7iDB52cB6_JpO7Dqa-eOYSs-mivpq)](https://drive.google.com/open?id=1-1z7iDB52cB6_JpO7Dqa-eOYSs-mivpq)
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

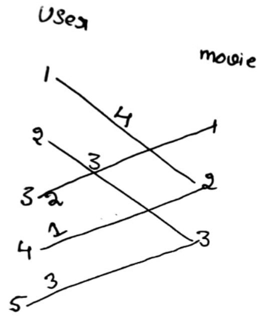
$$L = \min_{b,c,\{u_i\}_{i=1}^N,\{v_j\}_{j=1}^M} \quad \alpha\Big(\sum_j \sum_k v_{jk}^2 + \sum_i \sum_k u_{ik}^2 + \sum_i b_i^2 + \sum_j c_i^2\Big) + \sum_{i,j \in \mathcal{I}^{\text{train}}} (y_{ij} - \mu - b_i - c_j -$$

- 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 (https://en.wikipedia.org/wiki/Bipartite_graph)](https://en.wikipedia.org/wiki/Bipartite_graph) and the weight of each edge is the rating given by user to the movie



the Adjacency matrix

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

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

1. We will Apply SVD decomposition on the Adjaceny matrix [link1 (https://stackoverflow.com/a/31528944/4084039)](https://stackoverflow.com/a/31528944/4084039), [link2 (https://machinelearningmastery.com/singular-value-decomposition-for-machine-learning/)](https://machinelearningmastery.com/singular-value-decomposition-for-machine-learning/) 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 (https://drive.google.com/open?id=1PHFdJh_4gIPiLH5Q4UErH8GK71hTrzlY)](https://drive.google.com/open?id=1PHFdJh_4gIPiLH5Q4UErH8GK71hTrzlY) 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

Solution

In [1]:

```
import pandas as pd
from scipy.sparse import csr_matrix
import matplotlib.pyplot as plt
from sklearn.utils.extmath import randomized_svd
import numpy as np
from sklearn.metrics import mean_squared_error
from tqdm import tqdm
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

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

In [4]:

```
def create_adjanceny_matrix(df):
  rows = df["user_id"].values
  cols = df["item_id"].values
  vals = df["rating"].values

  mat = csr_matrix((vals, (rows,cols)))
  return mat
```

In [5]:

```
adjacency_matrix = create_adjanceny_matrix(data)
```

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]:

```
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]:

```
# Please use adjacency_matrix as matrix for SVD decompostion
# You can choose n_components as your choice
u, sig, vt = randomized_svd(adjacency_matrix, n_components=10, n_iter=5, random_state=1
0)
print(u.shape)
print(sig.shape)
print(vt.T.shape)
```

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

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.
    mean_rating = data["rating"].mean()

    return mean_rating
```

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

    return np.zeros(dim)
```

In [14]:

```
dim= adjacency_matrix.shape[0]
b_i=initialize(dim)
```

In [15]:

```
dim= adjacency_matrix.shape[1]
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

In [17]:

```
def derivative_db(user_id,item_id,rating,U,V,mu,alpha):
    '''In this function, we will compute dL/db_i'''
    global b_i, c_j
    utv = np.matmul( U[user_id], V.T[item_id].reshape(-1,1) )[0]
    res = 2*(rating - mu - utv - b_i[user_id] - c_j[item_id])*-1

    res = res + (alpha*2*b_i[user_id])
    return res
```

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=
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[18]:

```
True
```

Compute dL/dc_j

In [19]:

```
def derivative_dc(user_id,item_id,rating,U,V,mu):
    '''In this function, we will compute dL/dc_j'''
    global b_i, c_j, alpha
    utv = np.matmul( U[user_id], V.T[item_id].reshape(-1,1) )[0]
    res = 2*(rating - mu - utv - c_j[item_id] - b_i[user_id])*-1

    res = res + (alpha*2*c_j[item_id])
    return res
```

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=
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)
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]:

```
def SGD(epochs, mu, U, VT):
  # global b_i, c_j, alpha
  lr = 0.001
  MSE_vals = []
  for _ in tqdm(range(epochs)):
    preds = []
    for user_id, item_id ,rating in zip(data["user_id"],data["item_id"],data["rating"
]):

      db = derivative_db(user_id,item_id,rating,U,VT,mu,alpha)
      dc = derivative_dc(user_id,item_id,rating,U,VT,mu)

      b_i[user_id] = b_i[user_id] - lr * db
      c_j[item_id] = c_j[item_id] - lr * dc

      y_pred = mu + b_i[user_id] + c_j[item_id] + np.dot(U[user_id] ,VT.T[item_id])
      preds.append(y_pred)

    #MSE
    # print(b_i[:10])
    MSE_vals.append(mean_squared_error(preds,data["rating"]))
    # print(b_i - temp_bi)

  return MSE_vals
```

In [22]:

```
dim= adjacency_matrix.shape[1]
c_j=initialize(dim)
dim= adjacency_matrix.shape[0]
b_i=initialize(dim)
```

In [23]:

```
mse_vals =  SGD(20, mu, U1, V1)
mse_vals
```

```
100%|██████████| 20/20 [00:22<00:00,  1.12s/it]
```

Out[23]:

```
[1.1510800462034154,
 1.0338159129841364,
 0.9802011498748437,
 0.9493699260333147,
 0.929140692641979,
 0.9146979130957634,
 0.903775454396206,
 0.8951716056692925,
 0.8881894568551321,
 0.8823953618661482,
 0.877503700335892,
 0.8733173878694866,
 0.8696951142679584,
 0.866532288869014,
 0.8637494252426519,
 0.8612847654078504,
 0.8590894253887283,
 0.8571240978722742,
 0.8553567492765248,
 0.8537609712118557]
```

Plot epoch number vs MSE

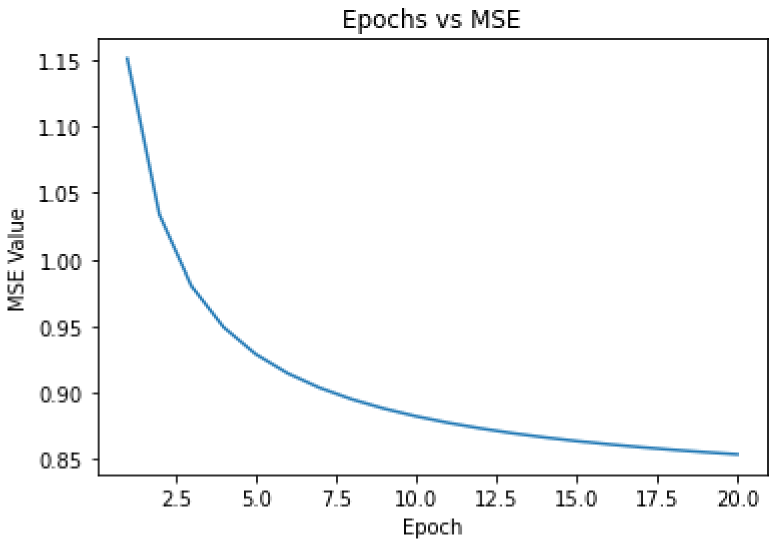
- epoch number on X-axis
- MSE on Y-axis

In [24]:

```
plt.plot(list(range(1,21)), mse_vals)
plt.xlabel("Epoch")
plt.ylabel("MSE Value")
plt.title("Epochs vs MSE")
```

Out[24]:

```
Text(0.5, 1.0, 'Epochs vs MSE')
```



Scaling U and V : Standard Scaler

In [25]:

```
dim= adjacency_matrix.shape[1]
c_j=initialize(dim)
dim= adjacency_matrix.shape[0]
b_i=initialize(dim)
```

In [26]:

```
scaler = StandardScaler()
U_scaled = scaler.fit_transform(U1)
V_scaled = scaler.fit_transform(V1)
```

In [27]:

```
mse_vals =  SGD(20, mu, U_scaled, V_scaled)
mse_vals
```

```
100%|██████████| 20/20 [00:23<00:00,  1.16s/it]
```

Out[27]:

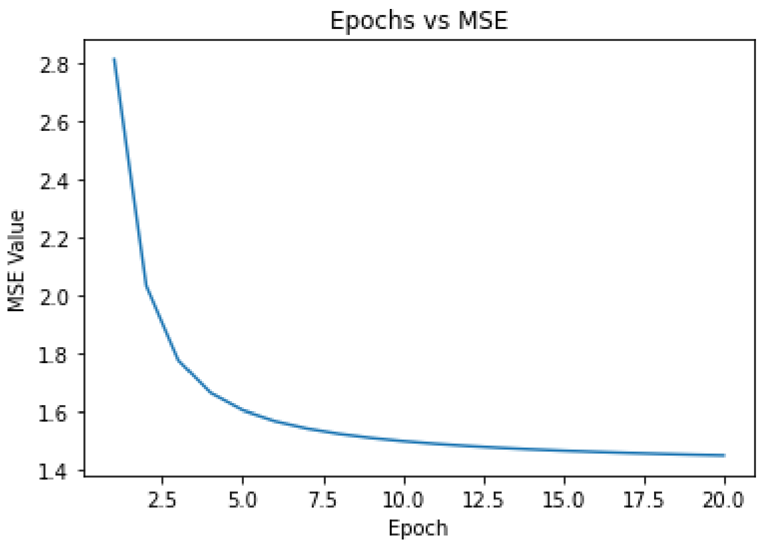
```
[2.812036763482004,
 2.032864970681036,
 1.7751283941860778,
 1.663474951434347,
 1.6038064008726933,
 1.5669164083427567,
 1.541761892594558,
 1.523419331318378,
 1.5093918815265124,
 1.4982822312795714,
 1.4892466745945994,
 1.4817444180590518,
 1.475411797447112,
 1.4699944720969078,
 1.465308656240855,
 1.4612178393538342,
 1.457618217400404,
 1.4544292476050622,
 1.4515873346520594,
 1.4490414953994843]
```

In [28]:

```
plt.plot(list(range(1,21)), mse_vals)
plt.xlabel("Epoch")
plt.ylabel("MSE Value")
plt.title("Epochs vs MSE")
```

Out[28]:

```
Text(0.5, 1.0, 'Epochs vs MSE')
```



Scaling U & V : MinMaxScaler

In [29]:

```
dim= adjacency_matrix.shape[1]
c_j=initialize(dim)
dim= adjacency_matrix.shape[0]
b_i=initialize(dim)
```

In [30]:

```
scaler = MinMaxScaler()
U_scaled_1 = scaler.fit_transform(U1)
V_scaled_1 = scaler.fit_transform(V1)
```

In [31]:

```
mse_vals =  SGD(20, mu, U_scaled_1, V_scaled_1)
mse_vals
```

```
100%|██████████| 20/20 [00:22<00:00,  1.10s/it]
```

Out[31]:

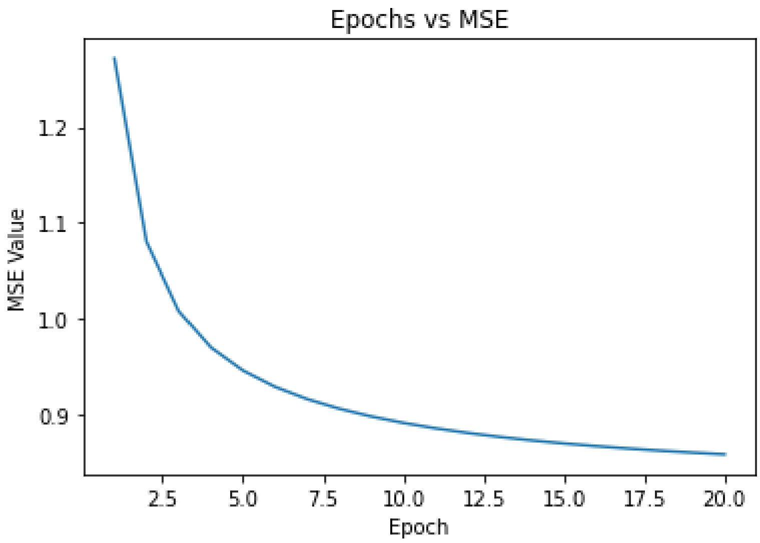
```
[1.2716382069322814,
 1.0805301205419193,
 1.0084493375581296,
 0.9704589418683304,
 0.946340201148767,
 0.9293412766828105,
 0.9165673862516862,
 0.9065479906821067,
 0.8984453460236629,
 0.891741914152932,
 0.8860979559538947,
 0.8812795711469558,
 0.8771194154987817,
 0.873493922017643,
 0.8703094243539014,
 0.8674933441204182,
 0.8649883901153547,
 0.8627486190790832,
 0.860736686345218,
 0.858921880054836]
```

In [32]:

```
plt.plot(list(range(1,21)), mse_vals)
plt.xlabel("Epoch")
plt.ylabel("MSE Value")
plt.title("Epochs vs MSE")
```

Out[32]:

```
Text(0.5, 1.0, 'Epochs vs MSE')
```



Task 2

In [46]:

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

In [47]:

```
user_mat = U1
user_mat.shape
```

Out[47]:

```
(943, 2)
```

In [48]:

```
df_user = pd.read_csv("user_info.csv")
```

Predict without scaling

In [49]:

```
X = user_mat
y = df_user["is_male"]
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2,random_state=0)

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

In [50]:

```
acc = accuracy_score(y_test, clf.predict(X_test))
print("Accuracy : ",acc)
```

```
Accuracy :  0.7354497354497355
```

Predict with Scaling

In [51]:

```
X = user_mat
y = df_user["is_male"]
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2,random_state=0)
```

In [52]:

```
scaler.fit(X_train)
X_train =  scaler.transform(X_train)
X_test =  scaler.transform(X_test)
clf1 = LogisticRegression().fit(X_train, y_train)
```

In [53]:

```
acc = accuracy_score(y_test, clf1.predict(X_test))
print("Accuracy : ",acc)
```

```
Accuracy :  0.7354497354497355
```

Conclusion

- Scaling does not seem to affect the results too much
- Using vector U to predict is_male, the LogisticRegression model gives an accuracy of : 0.7354497354497355

In [ ]:

```

```