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=1-1z7iDB5
2cB6_JpO7Dqa-eOYSs-mivpq)
2. The data will be of this format, each data point is represented as a trip
let 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

In [1]:

```
# Importing the required Packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.sparse import csr_matrix
```

In [2]:

```
#loading the required file
movie_rating = pd.read_csv('ratings_train.csv')
movie_rating.head()
movie_rating.shape[0]
```

Out[2]:

```
89992
```

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

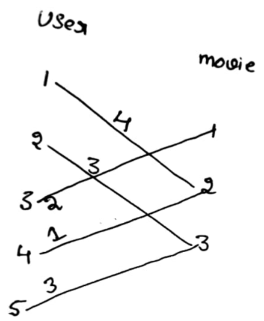
$$L = \min_{b,c,\{u_i\}_{i=1}^N,\{v_j\}_{j=1}^M} \quad \alpha\Big(\sum_j \sum_k v_{jk}^2 + \sum_i \sum_k u_{ik}^2 + \sum_i b_i^2 + \sum_j c_i^2\Big) + \sum_{i,j \in \mathcal{I}^{\text{train}}} (y_{ij} - \mu - b_i - c_j - u_i^T v_j)^2$$

- 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 [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

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

you can construct this matrix like $A[i][j] = r_{ij}$ here i is user_id, j is movie_id and r_{ij} is rating given by user i to the movie 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)

In [3]:

```
# from scipy.linalg import svd
# U, sigma, V_T = svd(adjacent_matrix)
```

In [4]:

```
# U.shape, sigma.shape, V_T.shape
```

2. 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.
3. Compute μ , μ represents the mean of all the rating given in the dataset.(write your code in def m_u())
4. 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())
5. 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())
6. Compute dL/db_i (Write you code in def derivative_db())
7. Compute dL/dc_j(write your code in def derivative_dc()
8. 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
```

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

9. you can choose any learning rate and regularization term in the range 10^{-3} to 10^2
10. **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

Type *Markdown* and LaTeX: α^2

Reading the csv file

In [5]:

```
#displaying sample contents
import pandas as pd
data=pd.read_csv('ratings_train.csv')
data.head()
```

Out[5]:

	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 [6]:

```
data.shape
```

Out[6]:

```
(89992, 3)
```

In [7]:

```
#doing the initilisation
import pandas as pd
user_info_data=pd.read_csv('ratings_train.csv')
μ   = np.mean(user_info_data['rating'])
b_i = np.ones(user_info_data.shape[0]) * 0.1
c_i = np.ones(user_info_data.shape[0]) * 0.1
u_i = user_info_data['user_id']
v_j = user_info_data['item_id']

ratings_details  = user_info_data['rating'].tolist()
users_details    = u_i.tolist()
movies_details   = v_j.tolist()

len(np.unique(ratings_details)), len(np.unique(users_details)), len(np.unique(movies
```

Out[7]:

```
(5, 943, 1662)
```

Create your adjacency matrix

In [8]:

```
#creating adjacent matrix
from scipy.sparse import csr_matrix
adjacency_matrix = csr_matrix((ratings_details, (users_details,movies_details))).toc
```

In [9]:

```
adjacency_matrix.shape
```

Out[9]:

```
(943, 1681)
```

Grader function - 1

In [10]:

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

Out[10]:

```
True
```

SVD decompostion

Sample code for SVD decompostion

In [11]:

```
#computing svd with components 5
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 [12]:

```
# Please use adjacency_matrix as matrix for SVD decompostion
from sklearn.utils.extmath import randomized_svd
import numpy as np
matrix = np.random.random((20, 10))
U, Sigma, VT = randomized_svd(adjacency_matrix, n_components=943,n_iter=5, random_st
print(U.shape)
print(Sigma.shape)
print(VT.T.shape)
# You can choose n_components as your choice
```

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

Compute mean of ratings

Compute mean of ratings

In [13]:

```
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.


    return ratings.mean()
```

In [14]:

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

```
3.529480398257623
```

Grader function -2

In [15]:

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

Out[15]:

```
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 [16]:

```
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 [17]:

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

In [18]:

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

Grader function -3

In [19]:

```
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[19]:

```
True
```

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

Compute dL/db_i

In [20]:

```
def derivative_db(user_id,item_id,rating,U1,V1,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(U1[user_i
    derivative_of_db_i = first_term + second_term
    return derivative_of_db_i
```

Grader function -4

In [21]:

```
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_sta
# 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)
print(value)
grader_db(value)
```

```
-0.9308283758773337
```

Out[21]:

```
True
```

Compute dL/dc_j

In [22]:

```
def derivative_dc(user_id,item_id,rating,U1,V1,mu,alpha):
    '''In this function, we will compute dL/dc_j'''
    first_term  =  2 * alpha * c_j[user_id]
    second_term = -2 * (rating - mu - b_i[user_id] - c_j[item_id]- (np.dot(U1[user_i
    derivative_of_dc_j =  first_term + second_term
    return derivative_of_dc_j
```

Grader function - 5

In [23]:

```
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_sta
# 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)
print(value)
grader_dc(value)
```

```
-2.9290787114434913
```

Out[23]:

```
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 [24]:

```
''' This was my inital try'''
# from tqdm import tqdm
# import random

# def predictions(users_details,movies_details,b_i, c_j):
#     all_predictions = []
#     for bi, cj, usrid, mvid in zip(b_i, c_j,users_details,movies_details):
#         prediction = (mu - bi - cj - (np.dot(usrid, mvid)))
#         all_predictions.append(prediction)
#     return all_predictions
# '''  intintializing the values '''
# learning_rate = alpha = 0.01
# ''' Looping through each epoch '''
# for each_point in tqdm(range(0, 20)):
#     ''' Iterating for each batch '''
#     for usr_id, itm_id, ratngs in zip(u_i,v_j,ratings_details):
#         ''' Getting random index '''
#         b_i = b_i - learning_rate * derivative_db(usr_id,itm_id,ratngs,U,VT,mu,alp
#         c_j = c_j - learning_rate * derivative_dc(usr_id,itm_id,ratngs,U,VT,mu,alp
#     ''' stroing the optimized weigths and bais for each epoch'''
#     model_preditions = predictions(u_i,v_j,b_i, c_j)
```

Out[24]:

```
' This was my inital try'
```

In [25]:

```
from tqdm import tqdm
from sklearn.metrics import mean_squared_error
# required details
learning_rate= 0.001
y=ratings_details
all_predictions = []
#running through 50 epochs
for epoch in tqdm(range(50)):
    #for all records of userdetails
    for user_id,item_id,rating in zip(users_details,movies_details,ratings_details):
        #calculating the derivates of b_i & c_j
        b_i[user_id]= b_i[user_id] - learning_rate *derivative_db(user_id,item_id,ra
        c_j[item_id]= c_j[item_id] - learning_rate *derivative_dc(user_id,item_id,ra
    #calculating hte predicitons after each epoch
    y_pred=[]
    for user_id,item_id in zip(users_details,movies_details):
        y_pred.append(mu+b_i[user_id]+c_j[item_id]+( np.dot(U[user_id].T, VT.T[item_
    #calculating mse for each epoch
    all_predictions.append(mean_squared_error(ratings_details,y_pred))
```

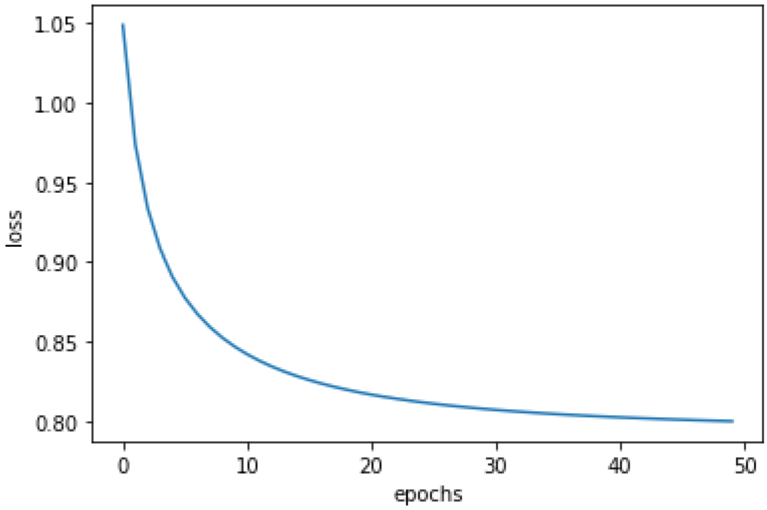
```
100%|██████████| 50/50 [01:17<00:00,  1.55s/it]
```

In [26]:

```
import matplotlib.pyplot as plt
#plotting results
plt.plot(all_predictions)
plt.xlabel('epochs')
plt.ylabel('loss')
```

Out[26]:

```
Text(0, 0.5, 'loss')
```



In [27]:

```
model_predictions = []
for user_id,item_id in zip(users_details,movies_details):
        model_predictions.append(mu+b_i[user_id]+c_j[item_id]+(np.dot(U[user_id].T,V
```

In [28]:

```
model_predictions
```

Out[28]:

```
[2.7611824800809774,
 4.059265007772838,
 3.884913542263527,
 3.63837658315012,
 4.235702221797333,
 3.2313053811216133,
 4.0963460897476285,
 3.88212715789788 3,
 3.887364309072743,
 2.841358113657961,
 3.0146833340519334,
 4.140630734252557,
 3.5062154752840877,
 3.3597031437355644,
 2.8093818371144974,
 3.6688883924342455,
 3.2857568867308906,
 3.880827861813851,
```

In [ ]:

```

```

In [29]:

```
# len(users_details), len(movies_details)
len(u_i),len(v_j),len(all_predictions)
```

Out[29]:

```
(89992, 89992, 50)
```

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

In [30]:

```
import pandas as pd
```

In [31]:

```
U.shape
```

Out[31]:

```
(943, 943)
```

In [32]:

```
data = pd.read_csv('user_info.csv.txt')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 943 entries, 0 to 942
Data columns (total 4 columns):
user_id         943 non-null int64
age             943 non-null int64
is_male         943 non-null int64
orig_user_id    943 non-null int64
dtypes: int64(4)
memory usage: 29.6 KB
```

In [33]:

```
np_age = np.array(data['age'])
```

In [34]:

```
#adding the new axis of age to users
age_newaxis = np_age[:,np.newaxis]
age_newaxis.shape
```

Out[34]:

```
(943, 1)
```

In [35]:

```
user_info_new_data = np.hstack((U, age_newaxis))
```

In [36]:

```
user_info_new_data.shape
```

Out[36]:

```
(943, 944)
```

In [37]:

```
#appyling logestic regression on the data
from sklearn.linear_model import LogisticRegression

User_models = LogisticRegression(random_state=0).fit(user_info_new_data, data['is_ma
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-p
ackages/sklearn/linear_model/logistic.py:432: FutureWarning: Default s
olver will be changed to 'lbfgs' in 0.22. Specify a solver to silence
this warning.
  FutureWarning)
```

In [38]:

```
#doing model predictions
model_predictions = User_models.predict(user_info_new_data)
```

In [39]:

```
#check f1 scrore metris
from sklearn.metrics import f1_score, accuracy_score
f1_score(list(data['is_male']),list(model_predictions),average='micro')
```

Out[39]:

```
0.7104984093319194
```

In [40]:

```
#checking the accuracy
accuracy_score(list(data['is_male']),list(model_predictions))
```

Out[40]:

```
0.7104984093319194
```

'''

1. Even data is important ***
2. This assignment gave intution of adjacent matrix - SVD
3. After adding other features how it behavous show in task-2

'''

In [ ]:

```

```