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

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

- 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 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* $r{ij}$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]:
```python
import pandas as pd
data=pd.read_csv('ratings_train.csv')
data = data.drop_duplicates()

df = data.copy()
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]:
```python
df['user_id'] = "UID_" + data.user_id.astype('str')
df['item_id'] = "IID_" + data.item_id.astype('str')
```

In [3]:
```python
print("user_id : ", data.user_id.shape)
print("item_id : ", data.item_id.shape)
print("rating : ", data.rating.shape)

print("unique user_id : ", data.user_id.unique().shape)
print("unique item_id : ", data.item_id.unique().shape)
print("unique rating : ", data.rating.unique().shape)
```

```
user_id :  (89992,)
item_id :  (89992,)
rating :  (89992,)
unique user_id :  (943,)
unique item_id :  (1662,)
unique rating :  (5,)
```

Create your adjacency matrix

In [4]:
```python
import numpy as np

table = pd.pivot(data, index='user_id', columns='item_id', values='rating')
table[np.isnan(table.values)] = 0
print(table.shape)

user_ids = list(table.index)
item_ids = list(table.columns)
ratings = table.values
```

```
(943, 1662)
```

In [5]:
```python
from scipy.sparse import csr_matrix
from networkx.algorithms import bipartite
import networkx as nx

# Initialise the graph
B = nx.Graph()# Add nodes with the node attribute "bipartite"

top_nodes = df['user_id'].values.tolist()
bottom_nodes = df['item_id'].values.tolist()

B.add_nodes_from(top_nodes, bipartite=0)
B.add_nodes_from(bottom_nodes, bipartite=1)# Add edges with weights

for i in df.itertuples():
    B.add_edge(i[1], i[2], weight = i[3])#Obtain the minimum weight full matching

print(nx.is_connected(B))

A = (B.subgraph(c) for c in nx.connected_components(B))
A = list(A)[0]

print("number of nodes", A.number_of_nodes())
print("number of edges", A.number_of_edges())
```

```
True
number of nodes 2605
number of edges 89992
```

In [6]:
```python
print(nx.info(B))
```

```
Name:
Type: Graph
Number of nodes: 2605
Number of edges: 89992
Average degree:  69.0917
```

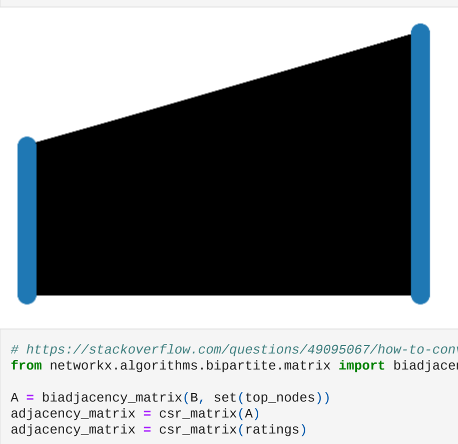
In [7]:
```python
print(nx.is_bipartite(B), B.is_directed(), B.is_multigraph())
```

```
True False False
```

In [8]:
```python
import matplotlib.pyplot as plt

l, r = bipartite.sets(B)

pos = {}
pos.update((node, (1, index)) for index, node in enumerate(l))
pos.update((node, (2, index)) for index, node in enumerate(r))
nx.draw(B, pos=pos, label=True)
plt.show()
```



In [9]:
```python
# https://stackoverflow.com/questions/49095067/how-to-convert-weighted-edge-list-to-adjacency-matrix-in-python
from networkx.algorithms.bipartite.matrix import biadjacency_matrix

A = biadjacency_matrix(B, set(top_nodes))
adjacency_matrix = csr_matrix(A)
adjacency_matrix = csr_matrix(ratings)
```

Grader function - 1

In [10]:
```python
def grader_matrix(matrix):
  assert(matrix.shape==(943,1662))
  return True

grader_matrix(adjacency_matrix)
```

Out[10]: True

Write your code for SVD decompostion

In [11]:
```python
# Please use adjacency_matrix as matrix for SVD decompostion
from sklearn.utils.extmath import randomized_svd
import numpy as np

U, Sigma, VT = randomized_svd(adjacency_matrix, n_components=2,n_iter=5, random_state=24)

print(U.shape)
print(Sigma.shape)
print(VT.T.shape)
```

```
(943, 2)
(2,)
(1662, 2)
```

Compute mean of ratings

In [12]:
```python
def m_u(ratings):
    '''In this function, we will compute mean for all the ratings'''
    return ratings.mean()
```

In [13]:
```python
mu=m_u(data['rating'])
print(mu)
```

```
3.529480398257623
```

Grader function -2

In [14]:
```python
def grader_mean(mu):
  assert(np.round(mu,3)==3.529)
  return True
mu=m_u(data['rating'])
grader_mean(mu)
```

Out[14]: 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 [15]:
```python
def initialize(dim):
    '''In this function, we will initialize bias value 'B' and 'C'.'''
    return np.zeros((dim)).tolist()
```

In [16]:
```python
dim= data.user_id.unique().shape# give the number of dimensions for b_i (Here b_i corresponds to users)
b_i=initialize(dim)
```

In [17]:
```python
dim= data.item_id.unique().shape# give the number of dimensions for c_j (Here c_j corresponds to movies)
c_j=initialize(dim)

len(b_i), len(c_j)
```

Out[17]: (943, 1662)

Grader function -3

In [18]:
```python
def grader_dim(b_i,c_j):
  assert(len(b_i)==943 and np.sum(b_i)==0)
  assert(len(c_j)==1662 and np.sum(c_j)==0)
  return True
grader_dim(b_i,c_j)
```

Out[18]: True

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

- 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

Compute dL/db_i

In [19]:
```python
db_gradient = 0

predicted = np.dot(U, VT)

def derivative_db(user_id,item_id,rating,U,V,mu,alpha):
    '''In this function, we will compute dL/db_i'''

    user_index = user_ids.index(user_id)
    item_index = item_ids.index(item_id)

    user_bias = b_i[user_id]
    item_bias = c_j[item_index]

    cost = (2 * alpha * user_bias) - (2*(rating - mu - user_bias - item_bias - predicted[user_index, item_index]))

    return cost
```

Grader function -4

In [20]:
```python
def grader_db(value):
    assert(np.round(value,3)==-0.931)
    return True

alpha=0.01
value=derivative_db(312,98,4,U,VT,mu,alpha)
print(value)
grader_db(value)
```

```
-0.9308283758773338
```

Out[20]: True

Compute dL/dc_j

In [21]:
```python
def derivative_dc(user_id,item_id,rating,U,V,mu,alpha):
    '''In this function, we will compute dL/dc_j'''
    user_index = user_ids.index(user_id)
    item_index = item_ids.index(item_id)

    user_bias = b_i[user_id]
    item_bias = c_j[item_index]

    cost = (2 * alpha * item_bias) - (2*(rating - mu - user_bias - item_bias - predicted[user_index, item_index]))

    return cost
```

Grader function - 5

In [22]:
```python
def grader_dc(value):
    assert(np.round(value,3)==-2.929)
    return True

r=0.01
value=derivative_dc(58,504,5,U,VT,mu,r)
grader_dc(value)
```

Out[22]: 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 [23]:
```python
from sklearn.metrics import mean_squared_error

epochs = 500
alpha = 0.01
learning_rate = 0.01
mse = []
early_stopping = 5
epoch_list = []

for epoch in range(epochs):
    act_ratings = []
    pred_ratings = []
    for pair in data.itertuples():
        user_id = pair[1]
        item_id = pair[2]
        act_ratings.append(pair[3])

        item_index = item_ids.index(item_id)

        user_bias = b_i[user_id]
        item_bias = c_j[item_index]

        cost = derivative_db(pair[1], pair[2], pair[3], U, VT, mu, alpha)
        b_i[user_id] = b_i[user_id]- (learning_rate*cost)

        cost = derivative_dc(pair[1], pair[2], pair[3], U, VT, mu, alpha)
        c_j[item_index] = c_j[item_index]- (learning_rate*cost)

        y_hat = mu + b_i[user_id] + c_j[item_index] + np.dot(b_i[user_id], c_j[item_index])
        pred_ratings.append(y_hat)

    MSE = mean_squared_error(np.asarray(act_ratings), np.asarray(pred_ratings))


    if epoch >= 10 and round(sum(mse[::-1][:early_stopping]) / early_stopping, 4) == mse[-1]:
        break;

    else:
        mse.append(round(MSE, 4))
        epoch_list.append(epoch)
        print(f'Epoch : {epoch+1}, MSE : {round(MSE, 4)}')
```

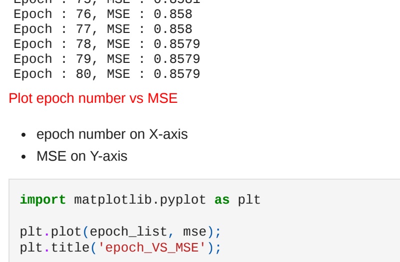
```
Epoch : 1, MSE : 0.9275
Epoch : 2, MSE : 0.8692
Epoch : 3, MSE : 0.8646
Epoch : 4, MSE : 0.8642
Epoch : 5, MSE : 0.8647
Epoch : 6, MSE : 0.8652
Epoch : 7, MSE : 0.8655
Epoch : 8, MSE : 0.8658
Epoch : 9, MSE : 0.8659
Epoch : 10, MSE : 0.8659
Epoch : 11, MSE : 0.8659
Epoch : 12, MSE : 0.8658
Epoch : 13, MSE : 0.8657
Epoch : 14, MSE : 0.8655
Epoch : 15, MSE : 0.8654
Epoch : 16, MSE : 0.8652
Epoch : 17, MSE : 0.865
Epoch : 18, MSE : 0.8648
Epoch : 19, MSE : 0.8646
Epoch : 20, MSE : 0.8645
Epoch : 21, MSE : 0.8643
Epoch : 22, MSE : 0.8641
Epoch : 23, MSE : 0.8639
Epoch : 24, MSE : 0.8637
Epoch : 25, MSE : 0.8635
Epoch : 26, MSE : 0.8633
Epoch : 27, MSE : 0.8631
Epoch : 28, MSE : 0.8629
Epoch : 29, MSE : 0.8628
Epoch : 30, MSE : 0.8626
Epoch : 31, MSE : 0.8624
Epoch : 32, MSE : 0.8623
Epoch : 33, MSE : 0.8621
Epoch : 34, MSE : 0.8619
Epoch : 35, MSE : 0.8618
Epoch : 36, MSE : 0.8616
Epoch : 37, MSE : 0.8615
Epoch : 38, MSE : 0.8613
Epoch : 39, MSE : 0.8612
Epoch : 40, MSE : 0.8611
Epoch : 41, MSE : 0.8609
Epoch : 42, MSE : 0.8608
Epoch : 43, MSE : 0.8607
Epoch : 44, MSE : 0.8606
Epoch : 45, MSE : 0.8604
Epoch : 46, MSE : 0.8603
Epoch : 47, MSE : 0.8602
Epoch : 48, MSE : 0.8601
Epoch : 49, MSE : 0.86
Epoch : 50, MSE : 0.8599
Epoch : 51, MSE : 0.8598
Epoch : 52, MSE : 0.8597
Epoch : 53, MSE : 0.8596
Epoch : 54, MSE : 0.8595
Epoch : 55, MSE : 0.8594
Epoch : 56, MSE : 0.8593
Epoch : 57, MSE : 0.8592
Epoch : 58, MSE : 0.8592
Epoch : 59, MSE : 0.8591
Epoch : 60, MSE : 0.859
Epoch : 61, MSE : 0.8589
Epoch : 62, MSE : 0.8588
Epoch : 63, MSE : 0.8588
Epoch : 64, MSE : 0.8587
Epoch : 65, MSE : 0.8586
Epoch : 66, MSE : 0.8586
Epoch : 67, MSE : 0.8585
Epoch : 68, MSE : 0.8585
Epoch : 69, MSE : 0.8584
Epoch : 70, MSE : 0.8583
Epoch : 71, MSE : 0.8583
Epoch : 72, MSE : 0.8582
Epoch : 73, MSE : 0.8582
Epoch : 74, MSE : 0.8581
Epoch : 75, MSE : 0.8581
Epoch : 76, MSE : 0.858
Epoch : 77, MSE : 0.858
Epoch : 78, MSE : 0.8579
Epoch : 79, MSE : 0.8579
Epoch : 80, MSE : 0.8579
```

Plot epoch number vs MSE

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

In [24]:
```python
import matplotlib.pyplot as plt

plt.plot(epoch_list, mse);
plt.title('epoch_VS_MSE');
```



Task 2

In [25]:
```python
df = pd.read_csv("user_info.csv.txt")
df.head()
```

Out[25]:

	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

USER - USER Vector

In [26]:
```python
target = df['is_male']

df = df['age'].values

df = np.column_stack((df, U))
```

In [27]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

model = LogisticRegression(random_state=2)

parameters = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}

grid = GridSearchCV(estimator=model, param_grid=parameters, cv=5, return_train_score=True, scoring='f1')

result = grid.fit(df, target)

result.best_score_, result.best_params_
```

Out[27]: (0.8307520719958464, {'C': 0.001})

USER - MOVIE Vector

In [28]:
```python
predicted.shape
```

Out[28]: (943, 1662)

In [29]:
```python
df = pd.read_csv("user_info.csv.txt")

target = df['is_male']

df = df['age'].values
df = np.column_stack((df, predicted))
```

In [30]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

model = LogisticRegression(random_state=2)

parameters = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}

grid = GridSearchCV(estimator=model, param_grid=parameters, cv=5, return_train_score=True, scoring='f1')

result = grid.fit(df, target)

result.best_score_, result.best_params_
```

Out[30]: (0.8307520719958464, {'C': 0.001})

USER - MOVIE vector after Scaling

In [39]:
```python
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df = scaler.fit_transform(df)

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

model = LogisticRegression(random_state=2)

parameters = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}

grid = GridSearchCV(estimator=model, param_grid=parameters, cv=5, return_train_score=True, scoring='f1')

result = grid.fit(df, target)

result.best_score_, result.best_params_
```

Out[39]: (0.8307520719958464, {'C': 0.001})

CONCLUSION:

```
1. USER - USER Vector before scaling - f1-Score - 83.07

2. USER - MOVIE Vector before scaling - f1-Score - 83.07

3. USER - MOVIE Vector after Scaling - f1-Score - 83.07
```

In [ ]:
```python
jupyter nbconvert –to pdf Recommendation_system_assignment.ipynb
```