# 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

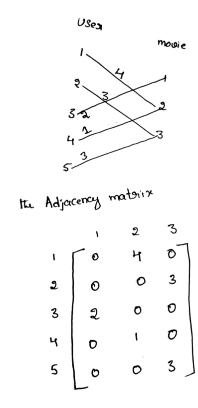
$$L = \min_{b,c,\{u_i\}_{i=1}^N,\{v_j\}_{j=1}^M} \quad \alpha\Big(\sum_j \sum_k v_{jk}^2 + \sum_i \sum_k u_{ik}^2 + \sum_i b_i^2 + \sum_j c_i^2\Big) + \sum_{i,j\in\mathcal{I}^{\text{train}}} (y_{ij} - \mu - b_i - c_j - u_i^T v_j)^2$$

- $\mu$ : 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




you can construct this matrix like $A[i][j] = r_{ij}$ here $i$ is user_id, $j$ is movie*id and $r{ij}isratinggivenbyuseritothemoviej$*

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 $\mu$ , $\mu$ 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 $c_j$

In [ ]:

In [ ]:

## 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

In [1]:
```python
import pandas as pd
from scipy.sparse import csr_matrix
from sklearn.utils.extmath import randomized_svd
import numpy as np
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
```

Reading the csv file

In [2]:
```python
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]:
```python
data.shape
```

Out[3]:
```
(89992, 3)
```

Create your adjacency matrix

In [4]:
```python
adjacency_matrix = csr_matrix((data.rating.values,(data.user_id.values,data.item_id.values)))
```

In [5]:
```python
adjacency_matrix.shape
```

Out[5]:
```
(943, 1681)
```

Grader function - 1

In [6]:
```python
def grader_matrix(matrix):
  assert(matrix.shape==(943,1681))
  return True
grader_matrix(adjacency_matrix)
```

Out[6]:
```
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 [7]:
```python
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 [8]:
```python
# Please use adjacency_matrix as matrix for SVD decompostion
# You can choose n_components as your choice
U, Sigma, VT = randomized_svd(adjacency_matrix, n_components=15,n_iter=5, random_state=0)
print(U.shape)
print(Sigma.shape)
print(VT.T.shape)
```

```
(943, 15)
(15,)
(1681, 15)
```

Compute mean of ratings

In [9]:
```python
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 i

    return ratings.mean()
```

In [10]:
```python
mu=m_u(data['rating'])
print(mu)
```

```
3.529480398257623
```

Grader function -2

In [11]:
```python
def grader_mean(mu):
  assert(np.round(mu,3)==3.529)
  return True
mu=m_u(data['rating'])
grader_mean(mu)
```

Out[11]:
```
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 [12]:
```python
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 [13]:
```python
dim= 943 # dimension of user bias = Number of users
b_i=initialize(dim)
```

In [14]:
```python
dim= 1681 # dimension of item bias = Number of items
c_j=initialize(dim)
```

Grader function -3

In [15]:
```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[15]:
```
True
```

Compute dL/db_i

In [16]:
```python
def derivative_db(user_id,item_id,rating,U,V,mu,alpha):
    '''In this function, we will compute dL/db_i'''
    dl_db = (2*alpha*b_i[user_id]) + ((2*(-1))*(rating-mu-b_i[user_id]-c_j[item_id]-(np.dot(U[user_id],V.T[item
    return dl_db
```

Grader function -4

In [17]:
```python
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[17]:
```
True
```

Compute dL/dc_j

In [18]:
```python
def derivative_dc(user_id,item_id,rating,U,V,mu, alpha=0.01):
    '''In this function, we will compute dL/dc_j'''
    dl_dc= (2*alpha*c_j[item_id]) + ((2*(-1))*(rating-mu-b_i[user_id]-c_j[item_id]-(np.dot(U[user_id],V.T[item_
    return dl_dc
```

Grader function - 5

In [19]:
```python
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[19]:
```
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
```

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

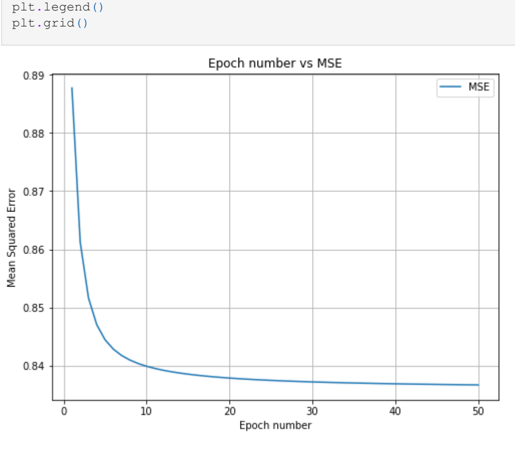
In [20]:
```python
epochs = 50
learning_rate = 0.01
alpha=0.01
mse=[]
for epo in range(epochs):
    predicted_ratings = []
    for user,movie,actual_ratings in zip(data['user_id'],data['item_id'],data['rating']):
        b_i[user] = b_i[user] - (learning_rate * derivative_db(user,movie,actual_ratings,U,VT,mu,alpha))
        c_j[movie] = c_j[movie] - (learning_rate * derivative_dc(user,movie,actual_ratings,U,VT,mu,alpha))
    for user,movie,actual_ratings in zip(data['user_id'],data['item_id'],data['rating']):
        pred = mu + b_i[user] + c_j[movie] + np.dot(U[user],VT.T[movie])
        predicted_ratings.append(pred)
    error = mean_squared_error(data['rating'],predicted_ratings)
    mse.append(error)
    print('Epoch: ',(epo+1),' MSE:', error)
```

```
Epoch:  1  MSE: 0.887744908816375
Epoch:  2  MSE: 0.861211187496819
Epoch:  3  MSE: 0.8516073319985425
Epoch:  4  MSE: 0.8470048400568961
Epoch:  5  MSE: 0.8444241174440302
Epoch:  6  MSE: 0.842811608288731
Epoch:  7  MSE: 0.8417196291321507
Epoch:  8  MSE: 0.8409331868060556
Epoch:  9  MSE: 0.8403391906087672
Epoch:  10  MSE: 0.8398736487583762
Epoch:  11  MSE: 0.8394980672675522
Epoch:  12  MSE: 0.8391880432701668
Epoch:  13  MSE: 0.8389273898553165
Epoch:  14  MSE: 0.8387049416013288
Epoch:  15  MSE: 0.8385127331296647
Epoch:  16  MSE: 0.8383449154609872
Epoch:  17  MSE: 0.8381970861253613
Epoch:  18  MSE: 0.8380658604356827
Epoch:  19  MSE: 0.8379485883663315
Epoch:  20  MSE: 0.8378431622246415
Epoch:  21  MSE: 0.8377478826349509
Epoch:  22  MSE: 0.8376613630014064
Epoch:  23  MSE: 0.8375824599970265
Epoch:  24  MSE: 0.8375102220574625
Epoch:  25  MSE: 0.8374438505881794
Epoch:  26  MSE: 0.8373826703175612
Epoch:  27  MSE: 0.8373261063417028
Epoch:  28  MSE: 0.8372736661408825
Epoch:  29  MSE: 0.8372249253415656
Epoch:  30  MSE: 0.8371795163360307
Epoch:  31  MSE: 0.8371371191073381
Epoch:  32  MSE: 0.8370974537740673
Epoch:  33  MSE: 0.8370602744889609
Epoch:  34  MSE: 0.8370253644127146
Epoch:  35  MSE: 0.8369925315483314
Epoch:  36  MSE: 0.8369616052693144
Epoch:  37  MSE: 0.8369324334110245
Epoch:  38  MSE: 0.8369048798219667
Epoch:  39  MSE: 0.8368788222928475
Epoch:  40  MSE: 0.836854150797583
Epoch:  41  MSE: 0.8368307659931725
Epoch:  42  MSE: 0.8368085779354056
Epoch:  43  MSE: 0.8367875049752963
Epoch:  44  MSE: 0.8367674728074833
Epoch:  45  MSE: 0.8367484136469137
Epoch:  46  MSE: 0.8367302655142133
Epoch:  47  MSE: 0.8367129716134772
Epoch:  48  MSE: 0.8366964797889095
Epoch:  49  MSE: 0.8366807420489502
Epoch:  50  MSE: 0.8366657141483538
```

Plot epoch number vs MSE

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

In [21]:
```python
plt.figure(figsize=(8,6))
plt.title('Epoch number vs MSE')
plt.xlabel('Epoch number')
plt.ylabel('Mean Squared Error')
plt.plot(range(1,epochs+1),mse,label='MSE')
plt.legend()
plt.grid()
```




## 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.

In [22]:
```python
data_new = pd.read_csv('user_info.csv.txt',sep=',')
```

In [23]:
```python
data_new.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]:
```python
dt = DecisionTreeClassifier(max_depth=5,random_state=0)
dt.fit(U, data_new['is_male'])
```

Out[24]:
```
DecisionTreeClassifier(max_depth=5, random_state=0)
```

In [25]:
```python
print('Accuracy score: ',accuracy_score(data_new['is_male'],dt.predict(U)))
```

```
Accuracy score:  0.7963944856839873
```

In [26]:
```python
print(confusion_matrix(data_new['is_male'],dt.predict(U)))
```

```
[[135 138]
 [ 54 616]]
```

### Observations:

1. After using a Decision Tree Classifier with max_depth=5, we get a good accuracy score of 79.6%.
2. On the confusion matrix, we see that there is a high number of True positives(616) and some amount of True Negatives (135) . This shows that the User Feature Vector(U) might contain some Gender information.