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

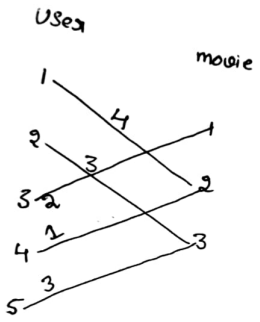
$$L = \min_{b,c,\{u_i\}_{i=1}^{N},\{v_j\}_{j=1}^{M}} \alpha\Big(\sum_j \sum_k v_{jk}^2 + \sum_i \sum_k u_{ik}^2 + \sum_i b_i^2 + \sum_j c_i^2\Big) + \sum_{i,j \in \mathcal{I}^{\text{train}}} (y_{ij} - \mu - b_i - c_j -$$

- 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 and the weight of each edge is the rating given by user to the movie



The Adjacency matrix

	1	2	3
1	0	4	0
2	0	0	3
3	2	0	0
4	0	1	0
5	0	0	3

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

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]:
```
import pandas as pd
data=pd.read_csv('ratings_train.csv')
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]:
```
data.shape
```

Out[2]:
```
(89992, 3)
```

Create your adjacency matrix

In [3]:
```
from scipy.sparse import csr_matrix

import numpy as np

row = np.array(data['user_id'])
col = np.array(data['item_id'])
data_2 = np.array(data['rating'])

adjacency_matrix = csr_matrix((data_2, (row, col)))


#Custom Implementation
# adjacency_matrix = csr_matrix(    )
# for user in unq_users:
#     mov = []

#     for movie in unq_movies:
#         df_ratings = np.array(data.rating[(data.user_id == user) & (data.item_id == m

#         if df_ratings.size == 0:
#             mov.append(0)
#         else:
#             mov.append(df_ratings[0])
#     adj_mat.append(mov)

# adjacency_matrix = np.array(adj_mat)
```

In [4]:
```
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

SVD decompostion

Sample code for SVD decompostion

In [6]:
```
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 [7]:
```
# Please use adjacency_matrix as matrix for SVD decompostion
# You can choose n_components as your choice

from sklearn.utils.extmath import randomized_svd
import numpy as np


U, Sigma, VT = randomized_svd(adjacency_matrix, n_components=5,n_iter=5, random_state=N

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

```
(943, 5)
(5,)
(1681, 5)
```

Compute mean of ratings

In [8]:
```
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.Dat
    return ratings.mean()
```

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

```
3.529480398257623
```

Grader function -2

In [10]:
```
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]:
```
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).tolist()
```

In [12]:
```
dim= 943# give the number of dimensions for b_i (Here b_i corresponds to users)
b_i=initialize(dim)
```

In [13]:
```
dim= 1681# give the number of dimensions for c_j (Here c_j corresponds to movies)
c_j=initialize(dim)
```

Grader function -3

In [14]:
```
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

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

In [15]:
```
def derivative_db(user_id,item_id,rating,U,V,mu,alpha):
    '''In this function, we will compute dL/db_i'''

    db = ( 2 * alpha * b_i[user_id] ) - ( 2* (  rating -mu -b_i[user_id] -c_j[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=
# 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

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

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

    db = ( 2 * alpha * c_j[item_id] ) - ( 2* (  rating -mu -c_j[item_id] -b_i[user_id]-

    return db
```

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=
# 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,r)
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]:
```
from sklearn.metrics import mean_squared_error


row = np.array(data['user_id'])
col = np.array(data['item_id'])
data_2 = np.array(data['rating'])

dim= 943
b_i=initialize(dim)

dim= 1681
c_j=initialize(dim)

ep = []
M_lst = []
for epoch in range(100):
    y_pred = []
```

```
    for user,movie,rating in zip(row,col,data_2):
        b_i[user] = b_i[user] - (0.001 * derivative_db(user,movie,rating,U,VT,mu,0.01))
        c_j[movie] = c_j[movie] - (0.001 * derivative_dc(user,movie,rating,U,VT,mu,0.01
        y_pred.append(mu +b_i[user] +c_j[movie] + np.dot(U[user],VT.T[movie]))
    MSE = mean_squared_error(data_2 ,y_pred)
    print ("MSE after epoch {} : {}".format(epoch+1,round(MSE,4) ))

    ep.append(epoch)
    M_lst.append(round(MSE,4))
```

```
MSE after epoch 1 : 1.1503
MSE after epoch 2 : 1.0331
MSE after epoch 3 : 0.9795
MSE after epoch 4 : 0.9487
MSE after epoch 5 : 0.9285
MSE after epoch 6 : 0.914
MSE after epoch 7 : 0.9031
MSE after epoch 8 : 0.8945
MSE after epoch 9 : 0.8875
MSE after epoch 10 : 0.8817
MSE after epoch 11 : 0.8768
MSE after epoch 12 : 0.8727
MSE after epoch 13 : 0.869
MSE after epoch 14 : 0.8659
MSE after epoch 15 : 0.8631
MSE after epoch 16 : 0.8606
MSE after epoch 17 : 0.8584
MSE after epoch 18 : 0.8565
MSE after epoch 19 : 0.8547
MSE after epoch 20 : 0.8531
MSE after epoch 21 : 0.8517
MSE after epoch 22 : 0.8503
MSE after epoch 23 : 0.8491
MSE after epoch 24 : 0.848
MSE after epoch 25 : 0.847
MSE after epoch 26 : 0.8461
MSE after epoch 27 : 0.8453
MSE after epoch 28 : 0.8445
MSE after epoch 29 : 0.8437
MSE after epoch 30 : 0.8431
MSE after epoch 31 : 0.8424
MSE after epoch 32 : 0.8418
MSE after epoch 33 : 0.8413
MSE after epoch 34 : 0.8408
MSE after epoch 35 : 0.8403
MSE after epoch 36 : 0.8399
MSE after epoch 37 : 0.8394
MSE after epoch 38 : 0.839
MSE after epoch 39 : 0.8387
MSE after epoch 40 : 0.8383
MSE after epoch 41 : 0.838
MSE after epoch 42 : 0.8377
MSE after epoch 43 : 0.8374
MSE after epoch 44 : 0.8371
MSE after epoch 45 : 0.8368
MSE after epoch 46 : 0.8366
MSE after epoch 47 : 0.8363
MSE after epoch 48 : 0.8361
MSE after epoch 49 : 0.8359
MSE after epoch 50 : 0.8357
MSE after epoch 51 : 0.8355
MSE after epoch 52 : 0.8353
MSE after epoch 53 : 0.8351
MSE after epoch 54 : 0.8349
```

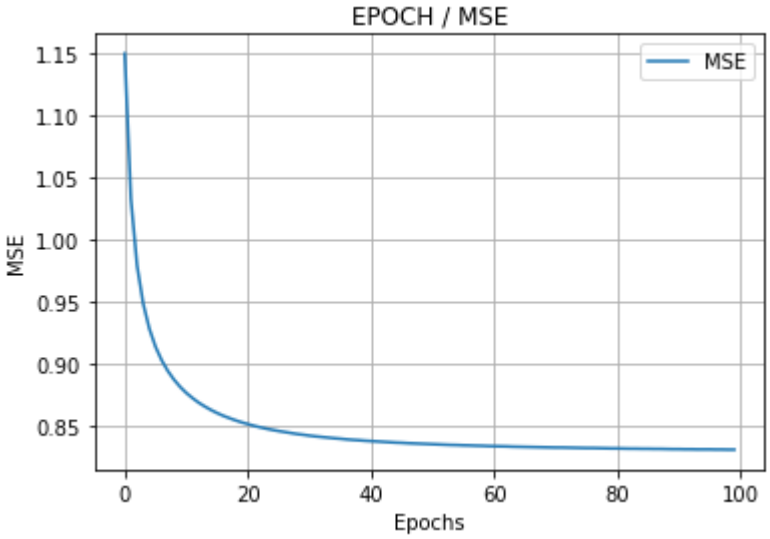
```
MSE after epoch 55 : 0.8348
MSE after epoch 56 : 0.8346
MSE after epoch 57 : 0.8345
MSE after epoch 58 : 0.8343
MSE after epoch 59 : 0.8342
MSE after epoch 60 : 0.834
MSE after epoch 61 : 0.8339
MSE after epoch 62 : 0.8338
MSE after epoch 63 : 0.8337
MSE after epoch 64 : 0.8335
MSE after epoch 65 : 0.8334
MSE after epoch 66 : 0.8333
MSE after epoch 67 : 0.8332
MSE after epoch 68 : 0.8331
MSE after epoch 69 : 0.833
MSE after epoch 70 : 0.8329
MSE after epoch 71 : 0.8328
MSE after epoch 72 : 0.8328
MSE after epoch 73 : 0.8327
MSE after epoch 74 : 0.8326
MSE after epoch 75 : 0.8325
MSE after epoch 76 : 0.8324
MSE after epoch 77 : 0.8324
MSE after epoch 78 : 0.8323
MSE after epoch 79 : 0.8322
MSE after epoch 80 : 0.8321
MSE after epoch 81 : 0.8321
MSE after epoch 82 : 0.832
MSE after epoch 83 : 0.8319
MSE after epoch 84 : 0.8319
MSE after epoch 85 : 0.8318
MSE after epoch 86 : 0.8318
MSE after epoch 87 : 0.8317
MSE after epoch 88 : 0.8317
MSE after epoch 89 : 0.8316
MSE after epoch 90 : 0.8315
MSE after epoch 91 : 0.8315
MSE after epoch 92 : 0.8314
MSE after epoch 93 : 0.8314
MSE after epoch 94 : 0.8313
MSE after epoch 95 : 0.8313
MSE after epoch 96 : 0.8313
MSE after epoch 97 : 0.8312
MSE after epoch 98 : 0.8312
MSE after epoch 99 : 0.8311
MSE after epoch 100 : 0.8311
```

Plot epoch number vs MSE

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

In [20]:

```
import matplotlib.pyplot as plt

plt.plot(ep, M_lst,label = "MSE")
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("MSE")
plt.title("EPOCH / MSE")
plt.grid()
plt.show()
```



Task 2

In [52]:
```
import pandas as pd
data=pd.read_csv('user_info.csv.txt')
data.head()
```

Out[52]:

	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 [53]:
```
data.shape
```

Out[53]:
```
(943, 4)
```

In [54]:
```
from sklearn.linear_model import LogisticRegression
```

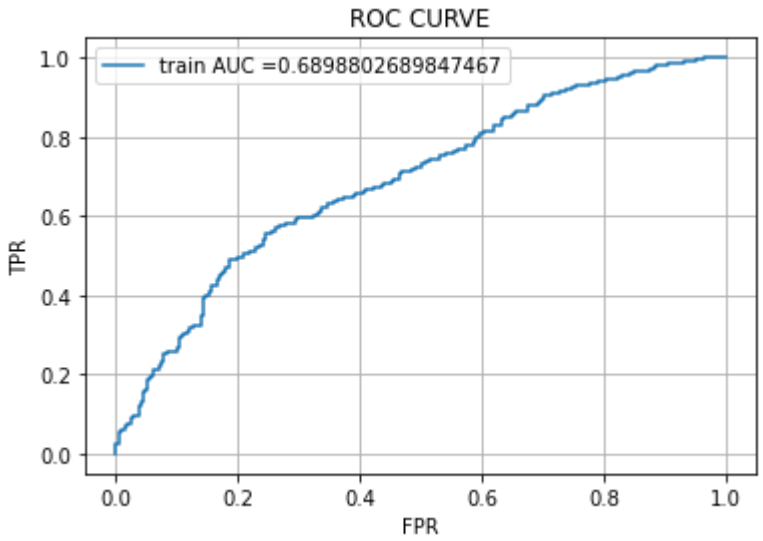
In [55]:
```
is_Male = data['is_male'].values
```

In [56]:
```
clf = LogisticRegression()
clf.fit(U,is_Male)
```

Out[56]:
```
LogisticRegression()
```

In [57]:
```
from sklearn.metrics import roc_curve, auc

y_train_pred = clf.predict_proba(U)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(is_Male, y_train_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.legend()
```

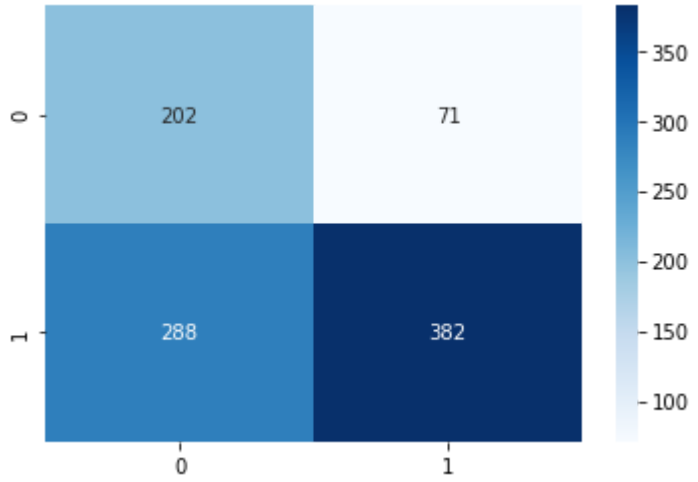
```
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC CURVE")
plt.grid()
plt.show()
```



In [58]:

```
import seaborn as sns
def find_best_threshold(threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.rou
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

print("="*100)
from sklearn.metrics import confusion_matrix

best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)

print("Train confusion matrix")

train_cm = confusion_matrix(is_Male, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(train_cm, annot=True,fmt="d",cmap='Blues')
plt.show()
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.4218686785850965 for threshold 0.708
Train confusion matrix
```

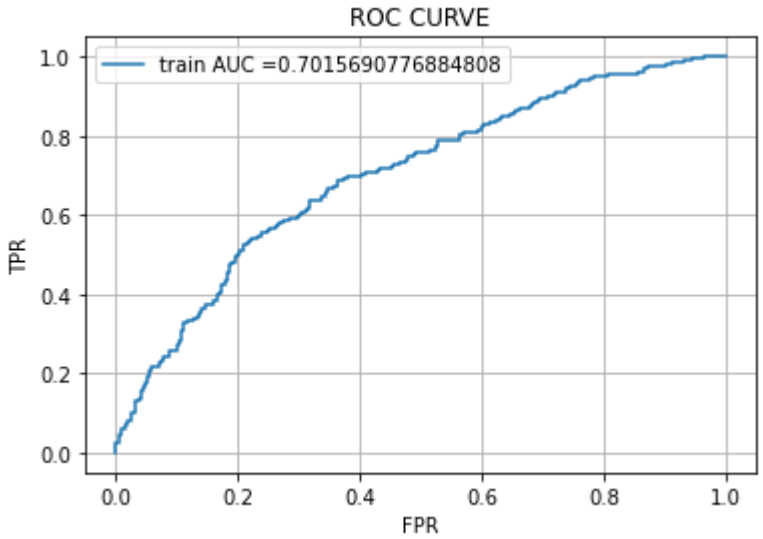


Unscaled U : High true positives is good but low true negative is not good. AUC score : 0.69

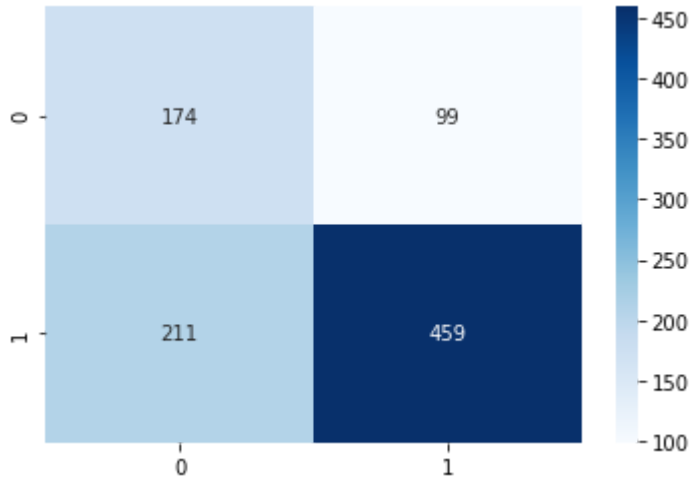
In [59]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
U_scaled = scaler.fit_transform(U)
```

In [60]:
```
clf = LogisticRegression()
clf.fit(U_scaled,is_Male)


from sklearn.metrics import roc_curve, auc

y_train_pred = clf.predict_proba(U_scaled)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(is_Male, y_train_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC CURVE")
plt.grid()
plt.show()


import seaborn as sns
def find_best_threshold(threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.rou
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

print("="*100)

from sklearn.metrics import confusion_matrix
```

```
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)

print("Train confusion matrix")

train_cm = confusion_matrix(is_Male, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(train_cm, annot=True,fmt="d",cmap='Blues')
plt.show()
```




```
========================================================================================
============
the maximum value of tpr*(1-fpr) 0.4366409709693292 for threshold 0.68
Train confusion matrix
```



Scaled U : After scaling, true positives have gone up but True negatives have come down. AUC score has marginally increased.