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

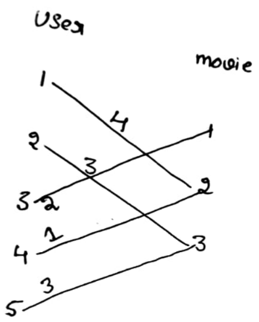
$$L = \min_{b,c,\{u_i\}_{i=1}^N,\{v_j\}_{j=1}^M} \quad \alpha\Big(\sum_j \sum_k v_{jk}^2 + \sum_i \sum_k u_{ik}^2 + \sum_i b_i^2 + \sum_j c_i^2\Big) + \sum_{i,j\in\mathcal{I}^{\text{train}}} (y_{ij} - \mu - b_i - c_j - u_i^T v_j)^2$$

- 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



The Adjacency matrix

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

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

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]:
```python
from scipy.sparse import csr_matrix
adjacency_matrix = csr_matrix((data['rating'], (data['user_id'], data['item_id'])))# write your code of adjacency mat
```

In [4]:
```python
adjacency_matrix.shape
```

Out[4]:
```
(943, 1681)
```

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

Out[5]:
```
<943x1681 sparse matrix of type '<class 'numpy.int64'>'
	with 89992 stored elements in Compressed Sparse Row format>
```

Grader function - 1

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

Out[6]:
```
True
```

SVD decompostion

Sample code for SVD decompostion

In [7]:
```python
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 [8]:

```
# Please use adjacency_matrix as matrix for SVD decompostion
# You can choose n_components as your choice

#https://stackoverflow.com/questions/31523575/get-u-sigma-v-matrix-from-truncated-svd-in-scikit-learn/315289
U, Sigma, VT = randomized_svd(adjacency_matrix, n_components=50,n_iter=5, random_state=None)
print(U.shape)
print(Sigma.shape)
print(VT.T.shape)
```

```
(943, 50)
(50,)
(1681, 50)
```

Compute mean of ratings

In [9]:

```
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 mo
    global_avg=ratings.mean(axis=0, skipna=True)

    return global_avg
```

In [10]:

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

```
3.529480398257623
```

Grader function -2

In [11]:

```
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]:
```
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 list(np.zeros(dim))
```

In [13]:
```
adjacency_matrix.shape
```

Out[13]: (943, 1681)

In [14]:
```
adjacency_matrix.shape[0], adjacency_matrix.shape[1]
```

Out[14]: (943, 1681)

In [15]:
```
dim= adjacency_matrix.todense().shape[0] # give the number of dimensions for b_i (Here b_i corresponds to users)
b_i=initialize(dim)
```

In [16]:
```
dim= adjacency_matrix.todense().shape[1] # give the number of dimensions for c_j (Here c_j corresponds to movies)
c_j=initialize(dim)
```

In [17]:
```
# b_i
# c_j
```

In [ ]:
```

```

Grader function -3

In [18]:
```
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[18]: True

Compute dL/db_i

In [19]:
```
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]-np.dot(U[user_id],V[:,item_id])))
    return db
```

Grader function -4

In [20]:
```
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[20]: True

Compute dL/dc_j

In [21]:
```
def derivative_dc(user_id,item_id,rating,U,V,mu, alpha=0.01):
    '''In this function, we will compute dL/dc_j'''
```

```
    dc = (2*alpha*c_j[item_id]) - (2*(rating-mu-b_i[user_id]-c_j[item_id]-np.dot(U[user_id],V[:,item_id])))
    return dc
```

Grader function - 5

In [22]:
```
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[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]:
```
from sklearn.metrics import mean_squared_error
from tqdm import tqdm

tp=tuple(data.iloc[:,[0,1]].values.tolist())
```

```
rating = data['rating']
mse = []
epochs = 35
lr= 0.01
alpha = 0.01
for i in tqdm(range(epochs)):
    y_pred = []
    j=0
    for (user_id,item_id) in tp:
        b_i[user_id] = b_i[user_id] - lr*(derivative_db(user_id,item_id,data['rating'][j],U,VT,mu,alpha))
        c_j[item_id] = c_j[item_id] - lr*(derivative_dc(user_id,item_id,data['rating'][j],U,VT,mu))

        j += 1
        y_pred.append(mu+b_i[user_id]+c_j[item_id]+np.dot(U[user_id],VT[:,item_id]))

    final_error = mean_squared_error(list(rating),y_pred)
    print("MSE:",final_error)
    mse.append(final_error)
```

```
  3%|██                                                                         | 1/35 [00:02<01:40,  2.97s/i
t]
MSE: 0.9017059075629639
  6%|█████                                                                      | 2/35 [00:05<01:35,  2.90s/i
t]
MSE: 0.8189199613277889
  9%|███████                                                                    | 3/35 [00:08<01:33,  2.93s/i
t]
MSE: 0.8036557093527108
 11%|█████████                                                                  | 4/35 [00:11<01:32,  2.98s/i
t]
MSE: 0.7973104715262753
 14%|████████████                                                               | 5/35 [00:14<01:30,  3.01s/i
t]
MSE: 0.7940308828624201
 17%|██████████████                                                             | 6/35 [00:17<01:27,  3.00s/i
t]
MSE: 0.7920948614001945
 20%|████████████████                                                           | 7/35 [00:20<01:23,  2.97s/i
t]
MSE: 0.790838805307673
 23%|███████████████████                                                        | 8/35 [00:23<01:19,  2.96s/i
```

```
t]
MSE: 0.7899639609297673
 26%|█████████████████████                                                              | 9/35 [00:26<01:17,  2.97s/i
t]
MSE: 0.7893205040759688
 29%|███████████████████████                                                           | 10/35 [00:29<01:13,  2.93s/i
t]
MSE: 0.788826821220127
 31%|██████████████████████████                                                        | 11/35 [00:32<01:12,  3.01s/i
t]
MSE: 0.7884353525355162
 34%|████████████████████████████                                                      | 12/35 [00:35<01:08,  2.98s/i
t]
MSE: 0.7881167576948707
 37%|██████████████████████████████                                                    | 13/35 [00:38<01:05,  2.97s/i
t]
MSE: 0.7878520297689855
 40%|█████████████████████████████████                                                 | 14/35 [00:41<01:01,  2.91s/i
t]
MSE: 0.7876283268687309
 43%|███████████████████████████████████                                               | 15/35 [00:44<00:57,  2.87s/i
t]
MSE: 0.7874366529007598
 46%|█████████████████████████████████████                                             | 16/35 [00:47<00:55,  2.92s/i
t]
MSE: 0.7872705072945051
 49%|████████████████████████████████████████                                          | 17/35 [00:50<00:53,  2.95s/i
t]
MSE: 0.7871250662723664
 51%|██████████████████████████████████████████                                        | 18/35 [00:53<00:50,  2.97s/i
t]
MSE: 0.7869966679114984
 54%|█████████████████████████████████████████████                                     | 19/35 [00:56<00:46,  2.93s/i
t]
MSE: 0.7868824774089583
 57%|███████████████████████████████████████████████                                   | 20/35 [00:59<00:44,  2.96s/i
t]
MSE: 0.7867802629278088
 60%|█████████████████████████████████████████████████                                 | 21/35 [01:01<00:40,  2.92s/i
t]
MSE: 0.78668824143653
```

```
 63%|███████████████████████████████████████████████████                               | 22/35 [01:04<00:37,  2.89s/i
t]
MSE: 0.7866049701229881
 66%|█████████████████████████████████████████████████████                             | 23/35 [01:07<00:34,  2.88s/i
t]
MSE: 0.7865292682587258
 69%|████████████████████████████████████████████████████████                          | 24/35 [01:10<00:31,  2.86s/i
t]
MSE: 0.786460159891621
 71%|██████████████████████████████████████████████████████████                        | 25/35 [01:13<00:28,  2.84s/i
t]
MSE: 0.7863968310923916
 74%|████████████████████████████████████████████████████████████                      | 26/35 [01:16<00:25,  2.84s/i
t]
MSE: 0.7863385975692134
 77%|███████████████████████████████████████████████████████████████                   | 27/35 [01:19<00:22,  2.87s/i
t]
MSE: 0.786284879799181
 80%|█████████████████████████████████████████████████████████████████                 | 28/35 [01:21<00:19,  2.85s/i
t]
MSE: 0.786235183696717
 83%|███████████████████████████████████████████████████████████████████               | 29/35 [01:24<00:16,  2.83s/i
t]
MSE: 0.7861890854196923
 86%|██████████████████████████████████████████████████████████████████████            | 30/35 [01:27<00:14,  2.83s/i
t]
MSE: 0.7861462193082627
 89%|████████████████████████████████████████████████████████████████████████          | 31/35 [01:30<00:11,  2.81s/i
t]
MSE: 0.7861062682238271
 91%|██████████████████████████████████████████████████████████████████████████        | 32/35 [01:32<00:08,  2.81s/i
t]
MSE: 0.7860689557467724
 94%|█████████████████████████████████████████████████████████████████████████████     | 33/35 [01:35<00:05,  2.82s/i
t]
MSE: 0.7860340398279682
 97%|███████████████████████████████████████████████████████████████████████████████   | 34/35 [01:38<00:02,  2.82s/i
t]
MSE: 0.786001307587486
100%|██████████████████████████████████████████████████████████████████████████████████| 35/35 [01:41<00:00,  2.90s/i
```

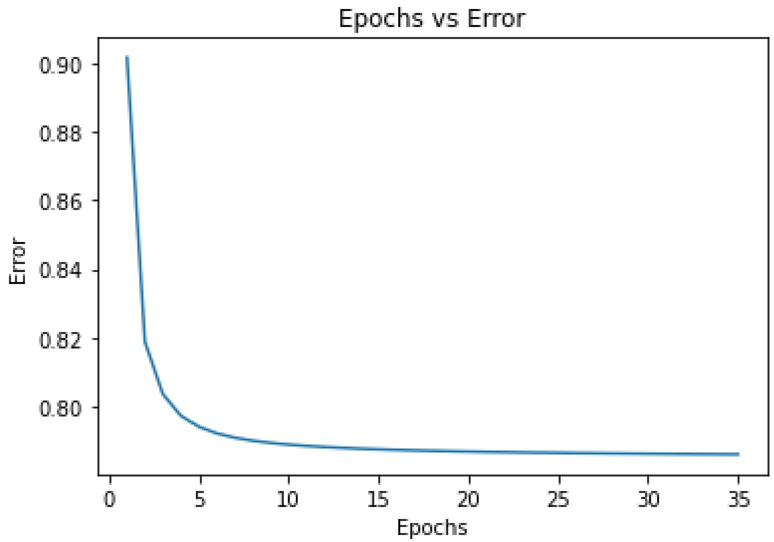
```
t]
MSE: 0.7859705710261061
```

Plot epoch number vs MSE

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

In [24]:

```
import matplotlib.pyplot as plt
plt.plot(range(1,len(mse)+1),mse,label="MEAN_SQUARED_ERROR")
plt.xlabel("Epochs")
plt.ylabel("Error")
plt.title("Epochs vs Error")
plt.show()
```



Task 2

In [25]:

```
data2 = pd.read_csv('user_info.csv.txt')
```

In [26]:
```
data2
```

Out[26]:

	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
...	...	...	...	...
938	938	26	0	939
939	939	32	1	940
940	940	20	1	941
941	941	48	0	942
942	942	22	1	943

943 rows × 4 columns

In [27]:
```
adjacency_matrix.shape, data2.shape
```

Out[27]:
```
((943, 1681), (943, 4))
```

In [28]:
```
data2['is_male'].value_counts()
```

Out[28]:
```
1    670
0    273
Name: is_male, dtype: int64
```

In [29]:
```
X = pd.DataFrame(U)
y = data2['is_male']
```

In [30]:
```
X.shape
```

Out[30]: (943, 50)

In [31]:
```
y.shape
```

Out[31]: (943,)

In [32]:
```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(C=0.1,class_weight='balanced')
clf.fit(X,y)
```

Out[32]: LogisticRegression(C=0.1, class_weight='balanced')

In [33]:
```
clf.score(X,y)
```

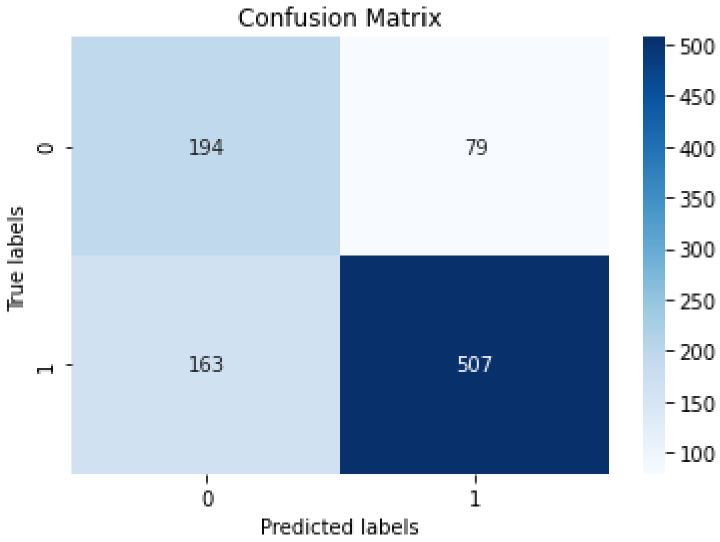
Out[33]: 0.7433722163308589

In [35]:
```
from sklearn.metrics import accuracy_score
print("Accuracy score : ",accuracy_score(y, clf.predict(X)))
```

```
Accuracy score :  0.7433722163308589
```

In [36]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y, clf.predict(X))
import seaborn as sns

ax= plt.subplot();
sns.heatmap(cm, annot=True,fmt='d',cmap='Blues',ax=ax);
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_ylim(2.0, 0)
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['0','1']);
ax.yaxis.set_ticklabels(['0','1']);
plt.show()
```



In [37]:
```
print("Confusion_Matrix")
print(cm)
```

```
Confusion_Matrix
[[194  79]
 [163 507]]
```

Effect of Scaling

In [38]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [39]:
```
X
```

Out[39]:
```
array([[ 1.95862075,  0.59064587, -0.65160275, ...,  2.85807543,
        -0.01392064,  1.44687207],
       [-0.52328929, -1.24841214,  1.53751696, ...,  0.47706637,
        -0.64041551, -0.33436841],
```

```
       [-0.91062587, -0.47865915,  0.38010501, ...,  0.91376541,
         0.30682688, -0.25899623],
       ...,
       [-0.81853694, -0.50605723, -0.05351838, ..., -0.5682914 ,
        -0.26244188,  0.19545511],
       [ 0.01267782,  0.48018606,  0.57114001, ..., -2.56983088,
        -1.83184124,  1.21479926],
       [ 0.87996273,  0.24399349, -2.18041155, ..., -0.49698596,
        -1.84214478, -0.93253443]])
```

In [40]:

```
X.shape
```

Out[40]:

```
(943, 50)
```

In [41]:

```
clf = LogisticRegression(C=0.1,class_weight='balanced')
clf.fit(X,y)
```

Out[41]:

```
LogisticRegression(C=0.1, class_weight='balanced')
```

In [42]:

```
clf.score(X,y)
```

Out[42]:

```
0.7317073170731707
```

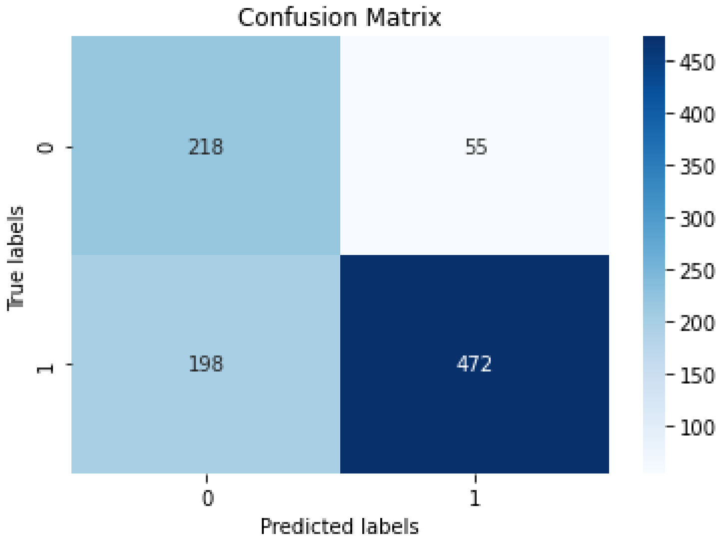
In [44]:

```
from sklearn.metrics import accuracy_score
print("Accuracy score : ",accuracy_score(y, clf.predict(X)))
```

```
Accuracy score :  0.7317073170731707
```

In [45]:

```
cm=confusion_matrix(y, clf.predict(X))
ax= plt.subplot();
sns.heatmap(cm, annot=True,fmt='d',cmap='Blues',ax=ax);
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_ylim(2.0, 0)
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['0','1']);
ax.yaxis.set_ticklabels(['0','1']);
plt.show()
```



There is not much change by Scaling, instead model performance slightly decreased.

In [ ]: