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_idmovie_idrating 772363 4712085 6414014 312984 585045 2357275

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

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

$ $

*. 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} 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.

$\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

```
import pandas as pd
data=pd.read_csv('ratings_train.csv')
data.head()
```

```
   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
```

```
data.shape
```

```
(89992, 3)
```

```
data.isnull().sum()
```

```
user_id    0
item_id    0
rating     0
dtype: int64
```

```
import numpy as np
print("unique userid : ",len(np.unique(data.user_id.values)))
print("unique itemid : ",len(np.unique(data.item_id.values)))
```

```
unique userid :  943
unique itemid :  1662
```

Create your adjacency matrix

```
from scipy.sparse import csr_matrix
adjacency_matrix = csr_matrix((data.rating.values,
(data.user_id.values,data.item_id.values)))
# write your code of adjacency matrix here
```

```
adjacency_matrix.shape
```

```
(943, 1681)
```

```
print("unique userid : ",len(np.unique(adjacency_matrix.tocoo().row)))
print("unique itemid : ",len(np.unique(adjacency_matrix.tocoo().col)))
```

```
unique userid :  943
unique itemid :  1662
```

Grader function - 1

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

```
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

```
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

```
# 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=100,n_iter=5, random_state=None)
print(U.shape)
print(Sigma.shape)
print(VT.T.shape)
```

```
(943, 100)
(100,)
(1681, 100)
```

Compute mean of ratings

```
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 more details.
    avg_rating=ratings.mean()

    return avg_rating

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

```
3.529480398257623
```

Grader function -2

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

```
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)

```
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
    initbias = np.zeros(dim)
    return initbias

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

dim= VT.T.shape[0]
# give the number of dimensions for c_j (Here c_j corresponds to
movies)
c_j=initialize(dim)
```

Grader function -3

```
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)
```

```
True
```

Compute dL/db_i

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

Grader function -4

```
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)
```

```
True
```

Compute dL/dc_j

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

Grader function - 5

```
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
alpha=0.01
value=derivative_dc(58,504,5,U1,V1,mu,alpha)
grader_dc(value)
```

```
True
```

Compute MSE (mean squared error) for predicted ratings

for each epoch, print the MSE value

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

```
from sklearn.metrics import mean_squared_error
from tqdm import tqdm
learning_rate=0.001
alpha=0.001
numberofepochs = 100
y_act=data["rating"]
epochs=[]
mse=[]
```

```
for epoch in range(numberofepochs):
  epochs.append(epoch+1)
  y_pred=[]
  for user,item,rating in zip(data.iloc[:, 0], data.iloc[:,
1],data.iloc[:, 2]):
    d_b=derivative_db(user,item,rating,U,VT,mu,alpha)
    b_i[user]=b_i[user]-learning_rate*d_b
    d_c=derivative_dc(user,item,rating,U,VT,mu,alpha)
    c_j[item]=c_j[item]-learning_rate*d_c
  for user,item,rating in zip(data.iloc[:, 0], data.iloc[:,
1],data.iloc[:, 2]):
    pred=mu+b_i[user]+c_j[item]+np.dot(U[user],VT.T[item])
    y_pred.append(pred)
  m= mean_squared_error(y_act,y_pred)
  mse.append(m)
  if (epoch+1) % 20 == 0:
    print("--"+" "+ "EPOCH"+" "+str(epoch+1))
    print("MSE :",m)
print("--"+" "+ "EPOCH"+" "+str(epoch+1))
print("MSE :",m)
```

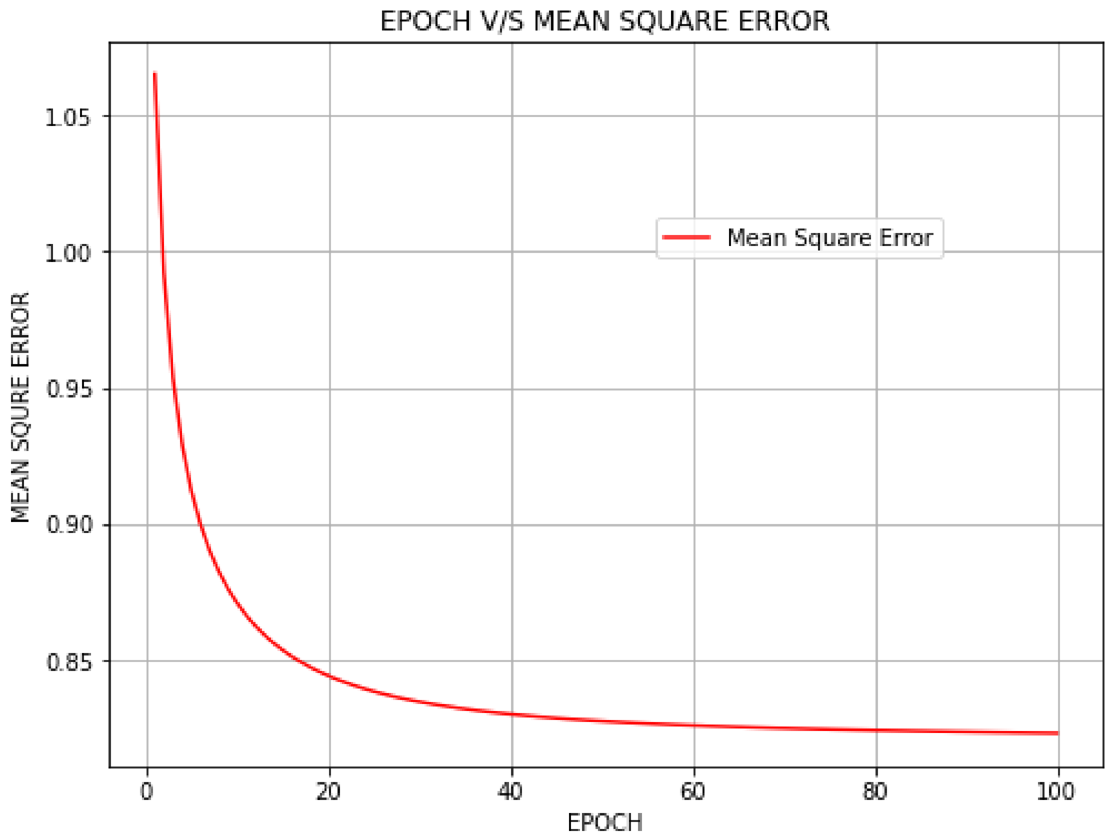
```
-- EPOCH 20
MSE : 0.8443669302577403
-- EPOCH 40
MSE : 0.830223995239063
-- EPOCH 60
MSE : 0.8261089232508129
-- EPOCH 80
MSE : 0.824282842764924
-- EPOCH 100
MSE : 0.823260293096768
-- EPOCH 100
MSE : 0.823260293096768
```

Plot epoch number vs MSE

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

```
import matplotlib.pyplot as plt
x=epochs
y=mse
plt.figure(figsize=(8,6))
plt.plot(x,y,label='Mean Square Error',color="red")
plt.grid()
plt.xlabel("EPOCH")
plt.ylabel("MEAN SQURE ERROR")
plt.title("EPOCH V/S MEAN SQUARE ERROR")
plt.legend(loc=(.55,.7))
```

```
<matplotlib.legend.Legend at 0x7f34a3a37f40>
```



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.

```
import pandas as pd
user_info=pd.read_csv('user_info.csv.txt')
user_info.head()
```

```
   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_data = np.column_stack((U,np.array(user_info.loc[:,
['age','is_male']])))
X = user_data[:,:-1]
y = user_data[:,-1]
print(X.shape)
print(y.shape)
```

```
(943, 101)
(943,)
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, f1_score,
roc_auc_score,log_loss,confusion_matrix,classification_report
import seaborn as sns
from imblearn.combine import SMOTETomek
from imblearn.pipeline import Pipeline as imbpipeline
from sklearn.model_selection import StratifiedKFold
```

Creating a result dataset to view results

```
result_dataset = pd.DataFrame(data =np.zeros((4,5)) , index =
['accuracy_score','f1_score','roc_auc_score','log_loss'],columns =
['LR','SVC','DT','RF','GBDT'])
result_dataset
```

	LR	SVC	DT	RF	GBDT
accuracy_score	0.0	0.0	0.0	0.0	0.0
f1_score	0.0	0.0	0.0	0.0	0.0
roc_auc_score	0.0	0.0	0.0	0.0	0.0
log_loss	0.0	0.0	0.0	0.0	0.0

Metrics for the Analysis

```
def metrics(classfier,X_test,y_test,name):
  y_pred = classfier.predict(X_test)
  y_prob = classfier.predict_proba(X_test)[:,[1]]
  accuracy = accuracy_score(y_test,y_pred)
  rocauc = roc_auc_score(y_test,y_prob)
  logloss = log_loss(y_test,y_prob)
```

```
  f1score = f1_score(y_test,y_pred)
  result_dataset[name]['accuracy_score'] = accuracy
  result_dataset[name]['f1_score'] = f1score
  result_dataset[name]['roc_auc_score'] = rocauc
  result_dataset[name]['log_loss'] = logloss
  print("accuracy score :",accuracy)
  print("f1 score :",f1score)
  print("auc score :",rocauc)
  print("log loss :",logloss)
  print('classification report : \n',
classification_report(y_test,y_pred))
  print('*********** confusion matrx ***********')
  sns.heatmap(confusion_matrix(y_test,y_pred),annot = True, cmap =
'Blues',fmt = '.2f')
  plt.show()
```

Lets Do train test split

```
X_train, X_test,y_train,y_test = train_test_split(X,y,test_size =
.20,stratify = y,random_state = 42)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(754, 101)
(189, 101)
(754,)
(189,)
```

```
unique, counts = np.unique(y_train, return_counts=True)
for u,c in zip(unique,counts):
  print(u,c)
```

```
0.0 218
1.0 536
```

```
scalar_for_grid =
ColumnTransformer([('scaling',StandardScaler(),list(range(X_train.shap
e[1])))],remainder='drop')
```

Logistic Regression

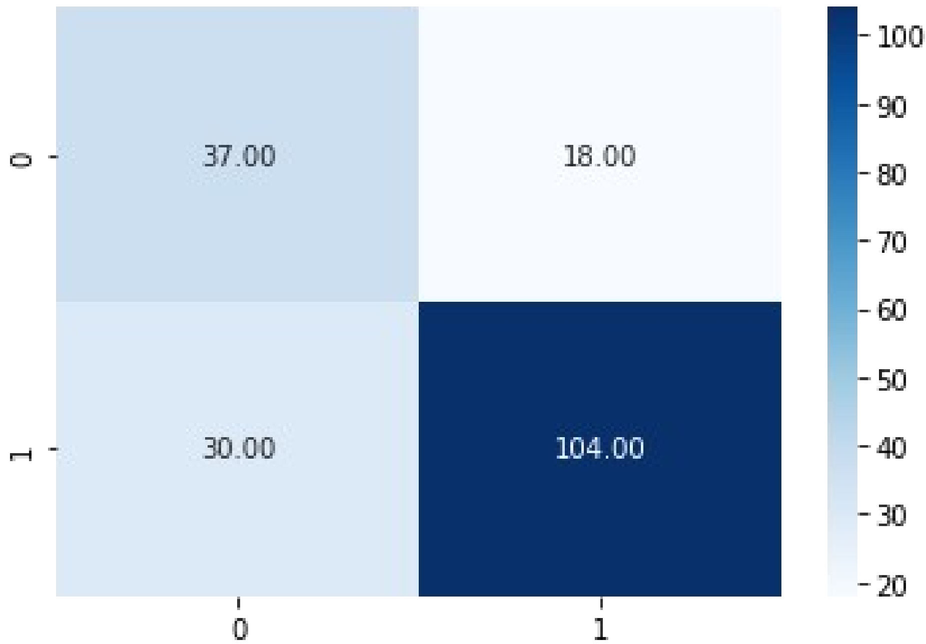
```
logistic_clf = LogisticRegression()
param_grids = {'model__C': [0.07,0.1,0.5,1.0,1.5,2.0]}
logistic_pipe = imbpipeline(steps = [['smote',
SMOTETomek(random_state=42)],['scaler', scalar_for_grid],
['model',logistic_clf]])
stratified_kfold =
StratifiedKFold(n_splits=3,shuffle=True,random_state=42)
logistic_grid = GridSearchCV(estimator=logistic_pipe,
param_grid=param_grids,cv = stratified_kfold, scoring =
'accuracy',verbose = 1)
```

```python
logistic_grid.fit(X_train, y_train)
print("------------Best Estimator---------------\n")
print(logistic_grid.best_estimator_)
print("------------Best Estimator parameters---------------\n")
print(logistic_grid.best_params_)
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
------------Best Estimator---------------

Pipeline(steps=[('smote', SMOTETomek(random_state=42)),
                ('scaler',
                 ColumnTransformer(transformers=[('scaling',
StandardScaler(),
                                                  [0, 1, 2, 3, 4, 5,
6, 7, 8, 9,
                                                   10, 11, 12, 13, 14,
15, 16,
                                                   17, 18, 19, 20, 21,
22, 23,
                                                   24, 25, 26, 27, 28,
29, ...])])),
                ['model', LogisticRegression()]])
------------Best Estimator parameters---------------

{'model__C': 1.0}
```

```python
metrics(logistic_grid.best_estimator_,X_test,y_test,'LR')
result_dataset
```

```
accuracy score : 0.746031746031746
f1 score : 0.8124999999999999
auc score : 0.7689280868385345
log loss : 0.6894758295535647
classification report :
               precision    recall  f1-score   support

         0.0       0.55      0.67      0.61        55
         1.0       0.85      0.78      0.81       134

    accuracy                           0.75       189
   macro avg       0.70      0.72      0.71       189
weighted avg       0.77      0.75      0.75       189

*********** confusion matrx ***********
```




```
                      LR  SVC   DT   RF  GBDT
accuracy_score  0.746032  0.0  0.0  0.0   0.0
f1_score        0.812500  0.0  0.0  0.0   0.0
roc_auc_score   0.768928  0.0  0.0  0.0   0.0
log_loss        0.689476  0.0  0.0  0.0   0.0
```

Support Vector Classification

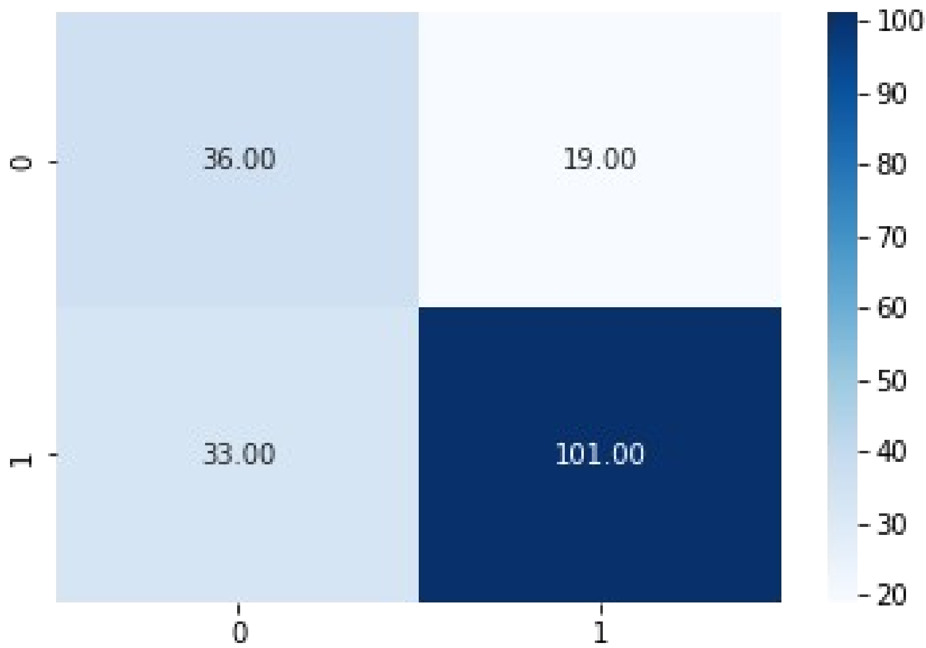
```
svc_clf = SVC(probability = True,random_state=42)
param_grids = {'model__C': [1.0, 10.0,20.0,40.0],'model__kernel' :
['linear', 'poly', 'rbf', 'sigmoid']}
svc_pipe = imbpipeline(steps = [['smote',
SMOTETomek(random_state=42)],['scaler', scalar_for_grid],
['model',svc_clf]])
stratified_kfold =
StratifiedKFold(n_splits=3,shuffle=True,random_state=42)
svc_grid = GridSearchCV(estimator=svc_pipe, param_grid=param_grids,cv
= stratified_kfold, scoring = 'accuracy',verbose = 1)
svc_grid.fit(X_train, y_train)
print("------------Best Estimator----------------\n")
print(svc_grid.best_estimator_)
print("------------Best Estimator parameters----------------\n")
print(svc_grid.best_params_)
```

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
------------Best Estimator----------------

Pipeline(steps=[('smote', SMOTETomek(random_state=42)),
                ('scaler',
                 ColumnTransformer(transformers=[('scaling',
StandardScaler(),
```

```
                                                  [0, 1, 2, 3, 4, 5,
6, 7, 8, 9,
                                                   10, 11, 12, 13, 14,
15, 16,
                                                   17, 18, 19, 20, 21,
22, 23,
                                                   24, 25, 26, 27, 28,
29, ...])])),
                ['model',
                 SVC(C=20.0, kernel='linear', probability=True,
                     random_state=42)]])
------------Best Estimator parameters----------------

{'model__C': 20.0, 'model__kernel': 'linear'}
```

```
metrics(svc_grid.best_estimator_,X_test,y_test,'SVC')
result_dataset
```

```
accuracy score : 0.7248677248677249
f1 score : 0.7952755905511811
auc score : 0.7582089552238807
log loss : 0.5725122775648757
classification report :
               precision    recall  f1-score   support

         0.0       0.52      0.65      0.58        55
         1.0       0.84      0.75      0.80       134

    accuracy                           0.72       189
   macro avg       0.68      0.70      0.69       189
weighted avg       0.75      0.72      0.73       189

*********** confusion matrx ***********
```



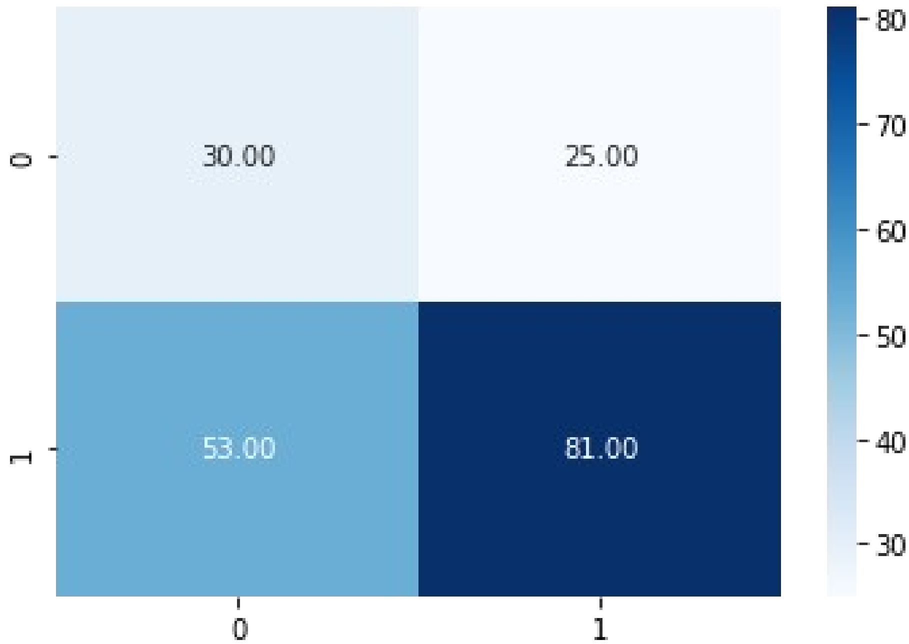

```
                       LR       SVC   DT   RF  GBDT
accuracy_score   0.746032  0.724868  0.0  0.0   0.0
f1_score         0.812500  0.795276  0.0  0.0   0.0
roc_auc_score    0.768928  0.758209  0.0  0.0   0.0
log_loss         0.689476  0.572512  0.0  0.0   0.0
```

Decision Tree Classification

```
dt_clf = DecisionTreeClassifier(random_state = 42)
param_grids = {'model__criterion':
['gini','entropy'],'model__max_depth' : [1,2,3,4,5,6,7,8]}
dt_pipe = imbpipeline(steps = [['smote', SMOTETomek(random_state=42)],
['scaler', scalar_for_grid],['model',dt_clf]])
stratified_kfold =
StratifiedKFold(n_splits=3,shuffle=True,random_state=42)
dt_grid = GridSearchCV(estimator=dt_pipe, param_grid=param_grids,cv =
stratified_kfold, scoring = 'accuracy',verbose = 1)
dt_grid.fit(X_train, y_train)
print("------------Best Estimator----------------\n")
print(dt_grid.best_estimator_)
print("------------Best Estimator parameters----------------\n")
print(dt_grid.best_params_)
```

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
------------Best Estimator----------------

Pipeline(steps=[('smote', SMOTETomek(random_state=42)),
                ('scaler',
                 ColumnTransformer(transformers=[('scaling',
StandardScaler(),
                                                  [0, 1, 2, 3, 4, 5,
```

```
6, 7, 8, 9,
                                                    10, 11, 12, 13, 14,
15, 16,
                                                    17, 18, 19, 20, 21,
22, 23,
                                                    24, 25, 26, 27, 28,
29, ...])])),
                ['model',
                 DecisionTreeClassifier(criterion='entropy',
max_depth=3,
                                        random_state=42)]])
------------Best Estimator parameters----------------

{'model__criterion': 'entropy', 'model__max_depth': 3}

metrics(dt_grid.best_estimator_,X_test,y_test,'DT')
result_dataset

accuracy score : 0.5873015873015873
f1 score : 0.675
auc score : 0.6130936227951154
log loss : 1.2112066378109478
classification report :
               precision    recall  f1-score   support

         0.0       0.36      0.55      0.43        55
         1.0       0.76      0.60      0.68       134

    accuracy                           0.59       189
   macro avg       0.56      0.57      0.55       189
weighted avg       0.65      0.59      0.61       189

*********** confusion matrx ***********
```




```
                     LR       SVC        DT   RF  GBDT
accuracy_score  0.746032  0.724868  0.587302  0.0   0.0
f1_score        0.812500  0.795276  0.675000  0.0   0.0
roc_auc_score   0.768928  0.758209  0.613094  0.0   0.0
log_loss        0.689476  0.572512  1.211207  0.0   0.0
```

Random Forest Classification

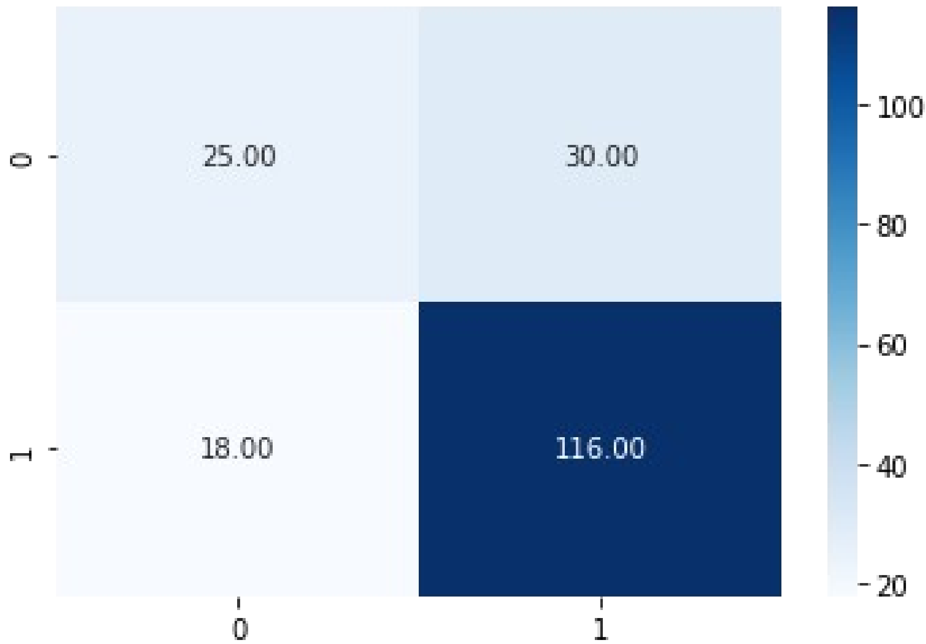
```
rf_clf = RandomForestClassifier()
param_grids = {'model__n_estimators' :
[90,95,100,105,110,115],'model__max_depth' : [9,10,13,17,19]}
rf_pipe = imbpipeline(steps = [['smote', SMOTETomek(random_state=42)],
['scaler', scalar_for_grid],['model',rf_clf]])
stratified_kfold =
StratifiedKFold(n_splits=3,shuffle=True,random_state=42)
rf_grid = GridSearchCV(estimator=rf_pipe, param_grid=param_grids,cv =
stratified_kfold, scoring = 'accuracy',verbose = 1)
rf_grid.fit(X_train, y_train)
print("------------Best Estimator----------------\n")
print(rf_grid.best_estimator_)
print("------------Best Estimator parameters----------------\n")
print(rf_grid.best_params_)
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
------------Best Estimator----------------

Pipeline(steps=[('smote', SMOTETomek(random_state=42)),
                ('scaler',
                 ColumnTransformer(transformers=[('scaling',
StandardScaler(),
                                                  [0, 1, 2, 3, 4, 5,
```

```
6, 7, 8, 9,
                                                    10, 11, 12, 13, 14,
15, 16,
                                                    17, 18, 19, 20, 21,
22, 23,
                                                    24, 25, 26, 27, 28,
29, ...])])),
                ['model',
                 RandomForestClassifier(max_depth=13,
n_estimators=110)]])
------------Best Estimator parameters----------------

{'model__max_depth': 13, 'model__n_estimators': 110}
```

```
metrics(rf_grid.best_estimator_,X_test,y_test,'RF')
result_dataset
```

```
accuracy score : 0.746031746031746
f1 score : 0.8285714285714286
auc score : 0.7459972862957938
log loss : 0.5663985650297113
classification report :
               precision    recall  f1-score   support

         0.0       0.58      0.45      0.51        55
         1.0       0.79      0.87      0.83       134

    accuracy                           0.75       189
   macro avg       0.69      0.66      0.67       189
weighted avg       0.73      0.75      0.74       189

*********** confusion matrx ***********
```




```
                      LR       SVC        DT        RF  GBDT
accuracy_score  0.746032  0.724868  0.587302  0.746032   0.0
f1_score        0.812500  0.795276  0.675000  0.828571   0.0
roc_auc_score   0.768928  0.758209  0.613094  0.745997   0.0
log_loss        0.689476  0.572512  1.211207  0.566399   0.0
```

Gradient Boosting Decision Trees

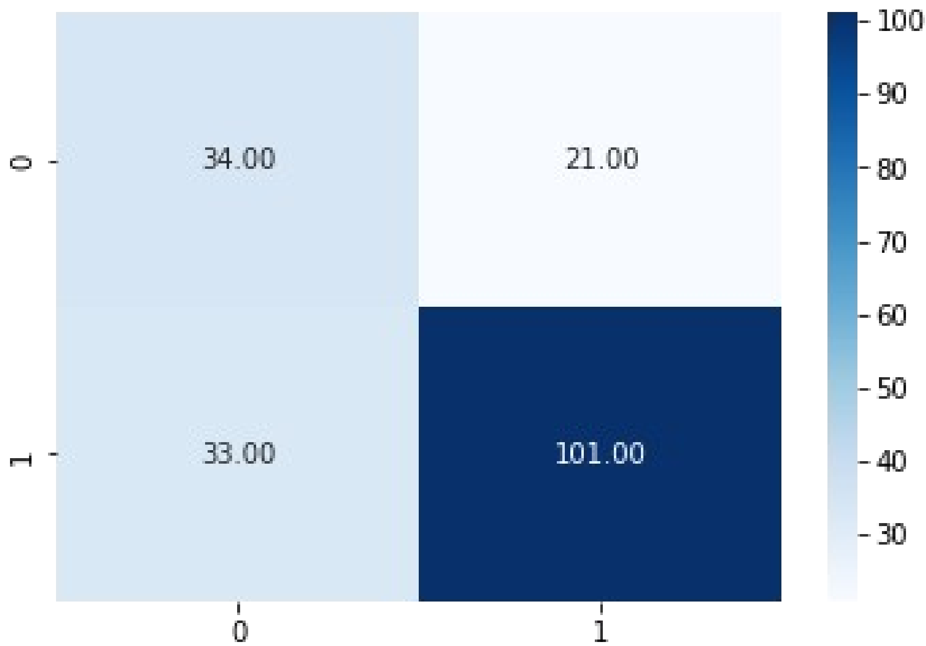
```
gbdt_clf = GradientBoostingClassifier(random_state = 42)
param_grids = {'model__loss' : ['deviance',
'exponential'],'model__learning_rate' :
[0.01,0.05,0.1],'model__n_estimators':[30,40,50,60,70,80]}
gbdt_pipe = imbpipeline(steps = [['smote',
SMOTETomek(random_state=42)],['scaler', scalar_for_grid],
['model',gbdt_clf]])
stratified_kfold =
StratifiedKFold(n_splits=3,shuffle=True,random_state=42)
gbdt_grid = GridSearchCV(estimator=gbdt_pipe,
param_grid=param_grids,cv = stratified_kfold, scoring =
'accuracy',verbose = 1)
gbdt_grid.fit(X_train, y_train)
print("------------Best Estimator----------------\n")
print(gbdt_grid.best_estimator_)
print("------------Best Estimator parameters----------------\n")
print(gbdt_grid.best_params_)
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
------------Best Estimator----------------

Pipeline(steps=[('smote', SMOTETomek(random_state=42)),
                ('scaler',
```

```
                 ColumnTransformer(transformers=[('scaling',
StandardScaler(),
                                                  [0, 1, 2, 3, 4, 5,
6, 7, 8, 9,
                                                   10, 11, 12, 13, 14,
15, 16,
                                                   17, 18, 19, 20, 21,
22, 23,
                                                   24, 25, 26, 27, 28,
29, ...])]))),
                ['model',
                 GradientBoostingClassifier(n_estimators=60,
random_state=42)]])
------------Best Estimator parameters----------------

{'model__learning_rate': 0.1, 'model__loss': 'deviance',
'model__n_estimators': 60}
```

```
metrics(gbdt_grid.best_estimator_,X_test,y_test,'GBDT')
result_dataset
```

```
accuracy score : 0.7142857142857143
f1 score : 0.7890625
auc score : 0.7341926729986431
log loss : 0.5646977224034936
classification report :
               precision    recall  f1-score   support

         0.0       0.51      0.62      0.56        55
         1.0       0.83      0.75      0.79       134

    accuracy                           0.71       189
   macro avg       0.67      0.69      0.67       189
weighted avg       0.73      0.71      0.72       189

*********** confusion matrx ***********
```



	LR	SVC	DT	RF	GBDT
accuracy_score	0.746032	0.724868	0.587302	0.746032	0.714286
f1_score	0.812500	0.795276	0.675000	0.828571	0.789062
roc_auc_score	0.768928	0.758209	0.613094	0.745997	0.734193
log_loss	0.689476	0.572512	1.211207	0.566399	0.564698