Machine Learning with the Equipment Failure Prediction.

Overview

Now I will take the selected/pre-processed/imputed datasets and feed this into custom ML models by using cross validation.

Business Problem in machine learning terms:

Given the data points with their respective features, use classification to find out whether the data points belong to surface failure or downhole failure.

Metric to be used:

F2 Score : ((1 + (2)^2) * Precision * Recall) / (((2)^2 * Precision) + Recall)

Datasets:

<u>3 datasets :</u>

1)Median imputed Median dataset.
2)Median imputed Adasyn dataset.
3)Median imputed Smotetomek dataset.

I will feed these 3 datasets into Machine Learning Algorithm.

In [1]:
```
import pickle
import pandas as pd
import numpy as np
```

In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [3]:
```
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
```

In [4]:
```
from sklearn.metrics import fbeta_score
from sklearn.metrics import make_scorer
import warnings
warnings.simplefilter(action="ignore", category=UserWarning)
from sklearn.model_selection import GridSearchCV
```

In [5]:
```
import os
os.cpu_count()
```

Out[5]: 8

Get the data

In [ ]:
```
new_standard_train_median_all_features_df=pd.read_pickle("new_standard_train_median_all_features_df.pickle")
y_train_median_all_features=pd.read_pickle("y_train_median_all_features.pickle")
new_standard_test_median_all_features_df=pd.read_pickle("new_standard_test_median_all_features_df.pickle")
y_test_median_all_features=pd.read_pickle("y_test_median_all_features.pickle")

new_x_adasyn_df=pd.read_pickle("new_x_adasyn_df.pickle")
y_adasyn=pd.read_pickle("y_adasyn.pickle")
new_standard_test_adasyn_all_features_df=pd.read_pickle("new_standard_test_adasyn_all_features_df.pickle")
y_test_adasyn_all_features=pd.read_pickle("y_test_adasyn_all_features.pickle")

new_x_smotetomek_df=pd.read_pickle("new_x_smotetomek_df.pickle")
y_smotetomek=pd.read_pickle("y_smotetomek.pickle")
new_standard_test_smotetomek_all_features_df=pd.read_pickle("new_standard_test_smotetomek_all_features_df.pi
y_test_smotetomek_all_features=pd.read_pickle("y_test_smotetomek_all_features.pickle")
```

In [7]:
```
y_train_median_all_features.value_counts()
```

Out[7]:
```
0.0    47200
1.0      800
Name: target, dtype: int64
```

In [8]:
```
y_test_median_all_features.value_counts()
```

Out[8]:
```
0.0    11800
1.0      200
Name: target, dtype: int64
```

Splitting the train set into D1 and D2(50-50)

In [ ]:
```
from sklearn.model_selection import train_test_split
D1_median_x_train, D2_median_x_train, D1_median_y_train, D2_median_y_train = \
train_test_split(new_standard_train_median_all_features_df, y_train_median_all_features, test_size=0.50, str
```

In [ ]:
```
D1_median_y_train.value_counts()
```

Out[15]:
```
0.0    23600
1.0      400
Name: target, dtype: int64
```

In [ ]:
```
D2_median_y_train.value_counts()
```

Out[16]:
```
0.0    23600
1.0      400
Name: target, dtype: int64
```

In [ ]:
```
from sklearn.model_selection import train_test_split
D1_adasyn_x_train, D2_adasyn_x_train, D1_adasyn_y_train, D2_adasyn_y_train = train_test_split(new_x_adasyn_d
```

In [ ]:
```
D1_adasyn_y_train.value_counts()
```

Out[18]:
```
0.0    23600
1.0    17696
Name: target, dtype: int64
```

In [ ]:
```
D2_adasyn_y_train.value_counts()
```

Out[19]:
```
0.0    23600
1.0    17697
Name: target, dtype: int64
```

In [ ]:
```
from sklearn.model_selection import train_test_split
D1_smotetomek_x_train, D2_smotetomek_x_train, D1_smotetomek_y_train, D2_smotetomek_y_train = train_test_spli
```

In [ ]:
```
D1_smotetomek_y_train.value_counts()
```

Out[21]:
```
0.0    23599
1.0    17700
Name: target, dtype: int64
```

In [ ]:
```
D2_smotetomek_y_train.value_counts()
```

Out[22]:
```
0.0    23600
1.0    17699
Name: target, dtype: int64
```

In [12]:
```
from sklearn.model_selection import train_test_split
# ignoring SettingWithCopyWarning
pd.options.mode.chained_assignment = None
def reset_index(x,y):
    x["target"]=y
    x.reset_index(drop=True, inplace=True)
    y=x.pop("target")
    return x,y
```

In [ ]:
```
D1_median_x_train,D1_median_y_train=reset_index(D1_median_x_train,D1_median_y_train)
D2_median_x_train,D2_median_y_train=reset_index(D2_median_x_train,D2_median_y_train)
D1_adasyn_x_train,D1_adasyn_y_train=reset_index(D1_adasyn_x_train,D1_adasyn_y_train)
D2_adasyn_x_train,D2_adasyn_y_train=reset_index(D2_adasyn_x_train,D2_adasyn_y_train)
D1_smotetomek_x_train,D1_smotetomek_y_train=reset_index(D1_smotetomek_x_train,D1_smotetomek_y_train)
D2_smotetomek_x_train,D2_smotetomek_y_train=reset_index(D2_smotetomek_x_train,D2_smotetomek_y_train)
```

In [ ]:
```
D1_median_x_train.to_pickle("D1_median_x_train.pickle",protocol=4)
D2_median_x_train.to_pickle("D2_median_x_train.pickle",protocol=4)
D1_median_y_train.to_pickle("D1_median_y_train.pickle",protocol=4)
D2_median_y_train.to_pickle("D2_median_y_train.pickle",protocol=4)
```

In [ ]:
```
D1_adasyn_x_train.to_pickle("D1_adasyn_x_train.pickle",protocol=4)
D2_adasyn_x_train.to_pickle("D2_adasyn_x_train.pickle",protocol=4)
D1_adasyn_y_train.to_pickle("D1_adasyn_y_train.pickle",protocol=4)
D2_adasyn_y_train.to_pickle("D2_adasyn_y_train.pickle",protocol=4)
```

In [ ]:
```
D1_smotetomek_x_train.to_pickle("D1_smotetomek_x_train.pickle",protocol=4)
D2_smotetomek_x_train.to_pickle("D2_smotetomek_x_train.pickle",protocol=4)
D1_smotetomek_y_train.to_pickle("D1_smotetomek_y_train.pickle",protocol=4)
D2_smotetomek_y_train.to_pickle("D2_smotetomek_y_train.pickle",protocol=4)
```

Now from this D1 do sampling with replacement to create d1,d2,d3....dk(k samples)

I will create model_count samples(hyperparameter). Each sample containing randomly (with replacement), batch_size% of the data(hyperparameter).

First the median data.

In [13]:
```
def run_Decision_Tree(x_train, y_train):

    f_2_score = make_scorer(fbeta_score, pos_label=1.0 , beta=2)

    depth_range=[4,6,8,9,10,12,14,17]
    min_samples_range=[2,10,20,30,40,50]

    parameters = {'max_depth': depth_range,'min_samples_split':min_samples_range}

    DT=DecisionTreeClassifier(criterion='gini',class_weight="balanced")

    grid = GridSearchCV(DT, parameters, cv=5,n_jobs=-1, scoring=f_2_score)

    grid.fit(x_train, y_train)

    return grid
```

In [14]:
```
from sklearn.model_selection import train_test_split
def randomly_stratify(x,y,test_size):
    x_remaining,x_batch_D1,y_remaining,y_batch_D1= train_test_split(x,y,test_size=test_size, stratify=y)
    return x_batch_D1,y_batch_D1
```

In [15]:
```
def train_the_decision_trees(x_train,y_train,data_type,i,m_x,b_x):

    x_batch,y_batch=randomly_stratify(x_train,y_train,b_x)

    x_batch,y_batch=reset_index(x_batch,y_batch)

    b_x=int(b_x*100)

    grid=run_Decision_Tree(x_batch,y_batch)

    DT=DecisionTreeClassifier(criterion='gini',class_weight="balanced",max_depth=int(grid.best_params_["max_
                              min_samples_split=int(grid.best_params_["min_samples_split"]))

    #Fit with train data with the best hyperparameter.
    DT.fit(x_batch,y_batch)

    #Save the trained model
    print("Saving DT_model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type))
    pickle_object = open("DT_model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_typ
                         ".pickle","wb")
    pickle.dump(DT, pickle_object)
    pickle_object.close()

    #Save the parameters for this best model
    print("Saving best_params_for_model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(dat
          +"_"+str(grid.best_params_))
    pickle_object = open("best_params_for_model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"
                         str(data_type)+".pickle","wb")
    pickle.dump(grid.best_params_, pickle_object)
    pickle_object.close()

    #Save the best scores for this best model
    print("Saving best_scores_for_model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(dat
          +"_"+str(grid.best_score_))
    pickle_object = open("best_score_for_model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+
                         str(data_type)+".pickle","wb")
    pickle.dump(grid.best_score_, pickle_object)
    pickle_object.close()

    #Save the D1 dataset batch given into this model.
    print(f"Saving {b_x} percent data set batch used to train the model {i} out of {m_x}")
```

```
    print("Saving x_batch_D1_for_model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data
    print("Saving y_batch_D1_for_model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data
    x_batch.to_pickle("x_batch_D1_for_model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+\
                      str(data_type)+".pickle",protocol=4)
    y_batch.to_pickle("x_batch_D1_for_model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+\
                      str(data_type)+".pickle",protocol=4)

    print("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n
    print("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++")

    return grid.best_params_,grid.best_score_
```

In [31]:
```
model_count=[5,6,7,8,9,10]
batch_size=[0.3,0.4,0.5,0.6,0.7,0.8]
```

In [ ]:
```
%%time
data_type="median_data"
input_list_median_data=[]
param_list_median_data=[]
score_list_median_data=[]
for m_x in model_count:
    for b_x in batch_size:
        for i in range(1,m_x+1):
            param,score=train_the_decision_trees(D1_median_x_train,D1_median_y_train,data_type,i,m_x,b_x)
            input_list_median_data.append("model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_"+data_t
            param_list_median_data.append(param)
            score_list_median_data.append(score)
```

```
Saving DT_model_1_of_5_batch_size_30_percent_median_data
Saving best_params_for_model_1_of_5_batch_size_30_percent_median_data_{'max_depth': 6, 'min_samples_spli
t': 40}
Saving best_scores_for_model_1_of_5_batch_size_30_percent_median_data_0.7350085289925611
Saving 30 percent data set batch used to train the model 1 out of 5
Saving x_batch_D1_for_model_1_of_5_batch_size_30_percent_median_data
Saving y_batch_D1_for_model_1_of_5_batch_size_30_percent_median_data
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Saving DT_model_2_of_5_batch_size_30_percent_median_data
Saving best_params_for_model_2_of_5_batch_size_30_percent_median_data_{'max_depth': 4, 'min_samples_spli
t': 20}
Saving best_scores_for_model_2_of_5_batch_size_30_percent_median_data_0.719288538354936
Saving 30 percent data set batch used to train the model 2 out of 5
Saving x_batch_D1_for_model_2_of_5_batch_size_30_percent_median_data
Saving y_batch_D1_for_model_2_of_5_batch_size_30_percent_median_data
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.....................................................................................................
```

Take all these models batch wise meaning taking them and run D2 into them to get the result.

In [ ]:
```
np.save("input_list_median_data",input_list_median_data)
np.save("param_list_median_data",param_list_median_data)
np.save("score_list_median_data",score_list_median_data)
```

In [32]:
```
input_list_median_data=np.load("input_list_median_data.npy",allow_pickle=True)
param_list_median_data=np.load("param_list_median_data.npy",allow_pickle=True)
score_list_median_data=np.load("score_list_median_data.npy",allow_pickle=True)
```

Selecting only the top 10 dataset and model config to train my Meta model that gives me the highest cv score.

In [33]:
```
%%time
counter=0
average=[]
models_n_batches=[]
for m_x in model_count:
    for b_x in batch_size:
        b_x=int(b_x*100)
        xxx=0
        val=0
        print("model_count : ",m_x)
        print("batch_size : ",b_x)
        for i in range(0,m_x):
            xxx=xxx+score_list_median_data[counter]
            counter=counter+1
            val=val+1
        print("number of models "+str(m_x)+", batch_size in "+str(b_x),"average : ",xxx/val)
        models_n_batches.append("number of models "+str(m_x)+", batch_size in "+str(b_x))
        average.append(xxx/val)
        print("++++++++++++++++++++++++++++++++++")
```

```
model_count :  5
batch_size :  30
number of models 5, batch_size in 30 average :  0.7297382561886976
++++++++++++++++++++++++++++++++++
model_count :  5
batch_size :  40
number of models 5, batch_size in 40 average :  0.7539683879580985
++++++++++++++++++++++++++++++++++
model_count :  5
batch_size :  50
number of models 5, batch_size in 50 average :  0.7729435300526573
++++++++++++++++++++++++++++++++++
model_count :  5
batch_size :  60
number of models 5, batch_size in 60 average :  0.7852574729941472
++++++++++++++++++++++++++++++++++
model_count :  5
batch_size :  70
number of models 5, batch_size in 70 average :  0.8115638954998803
```

Selecting the top 10 train scores and model configurations.

In [34]:
```
np.sort(average)[::-1][:10]
```

Out[34]:
```
array([0.82011362, 0.8190454 , 0.81827266, 0.81796324, 0.8115639 ,
       0.81063606, 0.8097179 , 0.80939151, 0.8059277 , 0.80401612])
```

In [35]:
```
np.array(models_n_batches)[np.argsort(average)[::-1][:10]]
```

Out[35]:
```
array(['number of models 7, batch_size in 80',
       'number of models 6, batch_size in 80',
       'number of models 8, batch_size in 80',
       'number of models 10, batch_size in 80',
       'number of models 5, batch_size in 70',
       'number of models 6, batch_size in 70',
       'number of models 10, batch_size in 70',
       'number of models 5, batch_size in 80',
       'number of models 9, batch_size in 80',
       'number of models 7, batch_size in 70'], dtype='<U37')
```

Now I pass the D2 set to each of these model_count models(hyper parameter), now I will get that many predictions for D2, from each of these models, which in turn will become my new dataset.

In [36]:
```
model_count=[7,6,8,10,5,6,10,5,9,7]
batch_size=[80,80,80,80,70,70,70,80,80,70]
```

In [37]:
```
data_type="median_data"

for m_x,b_x in zip(model_count,batch_size):
    print("number of models "+str(m_x)+", batch_size in "+str(b_x))

    xxx=pd.DataFrame()

    for i in range(1,m_x+1):

        #Load the model.
        pickle_object = open("DT_model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data
                             'rb')
        model = pickle.load(pickle_object)
        pickle_object.close()

        #Make the prediction.
        y_pred=model.predict(D2_median_x_train)

        #Assign the prediction to a new column in the current dataframe
        xxx[str(i)]=y_pred
        del model

    xxx.to_pickle("data_model_count_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type))
```

```
number of models 7, batch_size in 80
number of models 6, batch_size in 80
number of models 8, batch_size in 80
number of models 10, batch_size in 80
number of models 5, batch_size in 70
number of models 6, batch_size in 70
number of models 10, batch_size in 70
number of models 5, batch_size in 80
number of models 9, batch_size in 80
number of models 7, batch_size in 70
```

Now, this new dataset that I have, and the D2_Y values that I have ,I will train new meta model, with 10 configurations. and select the top meta model.

In [16]:
```
def train_the_meta_model(x_data,y_data,data_type,m_x,b_x):

    grid=run_Decision_Tree(x_data,y_data)

    Meta_Model=DecisionTreeClassifier(criterion='gini',class_weight="balanced",max_depth=int(grid.best_param
                              min_samples_split=int(grid.best_params_["min_samples_split"]))

    #Fit with train data with the best hyperparameter.
    Meta_Model.fit(x_data,y_data)

    #Save the trained model
    print("Saving Meta_model_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type))
    pickle_object = open("Meta_model_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type)+\
                         ".pickle","wb")
    pickle.dump(Meta_Model, pickle_object)
    pickle_object.close()

    #Save the parameters for this best model
    print("Saving best_params_for_Meta_model_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type)\
          +"_"+str(grid.best_params_))
    pickle_object = open("best_params_for_Meta_model_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data
                         ".pickle","wb")
    pickle.dump(grid.best_params_, pickle_object)
    pickle_object.close()

    #Save the best scores for this best model
    print("Saving best_scores_for_Meta_model_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type)\
          +"_"+str(grid.best_score_))
    pickle_object = open("best_scores_for_Meta_model_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data
                         ".pickle","wb")
    pickle.dump(grid.best_score_, pickle_object)
    pickle_object.close()

    print("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n
    print("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++")

    return grid.best_params_,grid.best_score_
```

In [39]:
```
Meta_Model_input_list_median_data=[]
Meta_Model_param_list_median_data=[]
Meta_Model_score_list_median_data=[]
data_type="median_data"

for m_x,b_x in zip(model_count,batch_size):
    print("number of models "+str(m_x)+", batch_size in "+str(b_x))
    x_data=pd.read_pickle("data_model_count_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type))
    y_data=D2_median_y_train.copy()
    param,score=train_the_meta_model(x_data,y_data,data_type,m_x,b_x)

    Meta_Model_input_list_median_data.append("Meta_model_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(
    Meta_Model_param_list_median_data.append(param)
    Meta_Model_score_list_median_data.append(score)

np.save("Meta_Model_input_list_median_data.npy",Meta_Model_input_list_median_data)
np.save("Meta_Model_param_list_median_data.npy",Meta_Model_param_list_median_data)
np.save("Meta_Model_score_list_median_data.npy",Meta_Model_score_list_median_data)
Meta_Model_input_list_median_data=np.load("Meta_Model_input_list_median_data.npy",allow_pickle=True)
Meta_Model_param_list_median_data=np.load("Meta_Model_param_list_median_data.npy",allow_pickle=True)
Meta_Model_score_list_median_data=np.load("Meta_Model_score_list_median_data.npy",allow_pickle=True)
```

```
number of models 7, batch_size in 80
Saving Meta_model_7_batch_size_80_percent_median_data
Saving best_params_for_Meta_model_7_batch_size_80_percent_median_data_{'max_depth': 6, 'min_samples_spli
t': 50}
Saving best_scores_for_Meta_model_7_batch_size_80_percent_median_data_0.8264799438094151
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
number of models 6, batch_size in 80
Saving Meta_model_6_batch_size_80_percent_median_data
Saving best_params_for_Meta_model_6_batch_size_80_percent_median_data_{'max_depth': 6, 'min_samples_spli
t': 30}
Saving best_scores_for_Meta_model_6_batch_size_80_percent_median_data_0.8029906559274667
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
number of models 8, batch_size in 80
Saving Meta_model_8_batch_size_80_percent_median_data
```

```
Saving best_params_for_Meta_model_8_batch_size_80_percent_median_data_{'max_depth': 8, 'min_samples_spli
```

Selecting the top Meta Model with the best CV scores and model configurations on which the main test data.

In [40]:
```
np.sort(Meta_Model_score_list_median_data)[::-1][0]
```

Out[40]:
```
0.8437671260841612
```

In [41]:
```
Meta_Model_input_list_median_data[np.argsort(Meta_Model_score_list_median_data)[::-1][0]]
```

Out[41]:
```
'Meta_model_9_batch_size_80_percent_median_data'
```

In [42]:
```
Meta_Model_param_list_median_data[np.argsort(Meta_Model_score_list_median_data)[::-1][0]]
```

Out[42]:
```
{'max_depth': 9, 'min_samples_split': 20}
```

In [ ]:
```
model_count=[9]
batch_size=[80]
for m_x,b_x in zip(model_count,batch_size):
    print("number of models "+str(m_x)+", batch_size in "+str(b_x))
```

```
number of models 9, batch_size in 80
```

Passing the test set that I have, to each of the base models with configurations with top 10 scores so that I get "model_count" predictions.

In [ ]:
```
model_count=[7,6,8,10,5,6,10,5,9,7]
batch_size=[80,80,80,80,70,70,70,80,80,70]
```

In [ ]:
```
data_type="median_data"
for m_x,b_x in zip(model_count,batch_size):
    print("number of models "+str(m_x)+", batch_size in "+str(b_x))

    xxx=pd.DataFrame()

    for i in range(1,m_x+1):

        #Load the model.
        pickle_object = open("DT_model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+\
                             str(data_type)+".pickle",'rb')
        model = pickle.load(pickle_object)
        pickle_object.close()

        #Make the prediction.
        y_pred=model.predict(new_standard_test_median_all_features_df)

        #Assign the prediction to a new column in the current dataframe
        xxx[str(i)]=y_pred
        del model

    xxx.to_pickle("test_data_model_count_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type))
```

```
number of models 7, batch_size in 80
number of models 6, batch_size in 80
number of models 8, batch_size in 80
number of models 10, batch_size in 80
number of models 5, batch_size in 70
number of models 6, batch_size in 70
number of models 10, batch_size in 70
number of models 5, batch_size in 80
number of models 9, batch_size in 80
number of models 7, batch_size in 70
```

Now I pass this new dataset and pass it to the meta_model to get the final prediction.

In [17]:
```
# Utility function to print the confusion matrix
def confusion_matrix(Y_true, Y_pred):

    true = pd.Series(Y_true)
    predicted = pd.Series(Y_pred)

    return pd.crosstab(true, predicted, rownames=['True'], colnames=['Predicted'])
```

In [ ]:
```
model_count=[9]
batch_size=[80]
```

In [ ]:
```
data_type="median_data"

for m_x,b_x in zip(model_count,batch_size):
    print("number of models "+str(m_x)+", batch_size in "+str(b_x))

    #Load the model.
    pickle_object = open("Meta_model_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type)+".pickle"
                         'rb')
    model = pickle.load(pickle_object)
    pickle_object.close()

    #Load the data.
    x_test=pd.read_pickle("test_data_model_count_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_typ
    y_test=y_test_median_all_features.copy()

    #Make the prediction.
    y_pred=model.predict(x_test)

    #Now try and figure out the F2 Score.
    f2_score=fbeta_score(y_test, y_pred, pos_label=1.0, beta=2)
    print("f2_score : ",f2_score)
    print("confusion_matrix : \n",confusion_matrix(y_test, y_pred))
```

```
number of models 9, batch_size in 80
f2_score :  0.8175559380378657
confusion_matrix :
 Predicted    0.0  1.0
True
0.0        11628  172
1.0           10  190
```

Repeat the whole process with Adasyn Data now.

In [80]:
```
model_count=[5,6,7,8,9,10]
batch_size=[0.3,0.4,0.5,0.6,0.7,0.8]
```

In [ ]:
```
%%time
data_type="adasyn_data"
input_list_adasyn_data=[]
param_list_adasyn_data=[]
score_list_adasyn_data=[]
for m_x in model_count:
    for b_x in batch_size:
        for i in range(1,m_x+1):
            param,score=train_the_decision_trees(D1_adasyn_x_train,D1_adasyn_y_train,data_type,i,m_x,b_x)
            input_list_adasyn_data.append("model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_"+data_t
            param_list_adasyn_data.append(param)
            score_list_adasyn_data.append(score)
```

```
Saving DT_model_1_of_5_batch_size_30_percent_adasyn_data
Saving best_params_for_model_1_of_5_batch_size_30_percent_adasyn_data_{'max_depth': 12, 'min_samples_spli
t': 2}
Saving best_scores_for_model_1_of_5_batch_size_30_percent_adasyn_data_0.9901077677817346
Saving 30 percent data set batch used to train the model 1 out of 5
Saving x_batch_D1_for_model_1_of_5_batch_size_30_percent_adasyn_data
Saving y_batch_D1_for_model_1_of_5_batch_size_30_percent_adasyn_data
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Saving DT_model_2_of_5_batch_size_30_percent_adasyn_data
Saving best_params_for_model_2_of_5_batch_size_30_percent_adasyn_data_{'max_depth': 9, 'min_samples_spli
t': 2}
Saving best_scores_for_model_2_of_5_batch_size_30_percent_adasyn_data_0.9891015763715572
Saving 30 percent data set batch used to train the model 2 out of 5
Saving x_batch_D1_for_model_2_of_5_batch_size_30_percent_adasyn_data
Saving y_batch_D1_for_model_2_of_5_batch_size_30_percent_adasyn_data
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.....
```

Take all these models batch wise meaning taking them and run D2 into them to get the result.

In [ ]:
```
np.save("input_list_adasyn_data",input_list_adasyn_data)
np.save("param_list_adasyn_data",param_list_adasyn_data)
np.save("score_list_adasyn_data",score_list_adasyn_data)
```

In [81]:
```
input_list_adasyn_data=np.load("input_list_adasyn_data.npy",allow_pickle=True)
param_list_adasyn_data=np.load("param_list_adasyn_data.npy",allow_pickle=True)
score_list_adasyn_data=np.load("score_list_adasyn_data.npy",allow_pickle=True)
```

Selecting only the top 10 dataset and model config to train my Meta model that gives me the highest cv score.

In [82]:
```
%%time
counter=0
average=[]
models_n_batches=[]
for m_x in model_count:
    for b_x in batch_size:
        b_x=int(b_x*100)
        xxx=0
        val=0
        print("model_count : ",m_x)
        print("batch_size : ",b_x)
        for i in range(0,m_x):
            xxx=xxx+score_list_adasyn_data[counter]
            counter=counter+1
            val=val+1
        print("number of models "+str(m_x)+", batch_size in "+str(b_x),"average : ",xxx/val)
        models_n_batches.append("number of models "+str(m_x)+", batch_size in "+str(b_x))
        average.append(xxx/val)
        print("++++++++++++++++++++++++++++++++++")
```

```
model_count :  5
batch_size :  30
number of models 5, batch_size in 30 average :  0.98901524049686
++++++++++++++++++++++++++++++++++
model_count :  5
batch_size :  40
number of models 5, batch_size in 40 average :  0.9907827544048079
++++++++++++++++++++++++++++++++++
model_count :  5
batch_size :  50
number of models 5, batch_size in 50 average :  0.9915046500647744
++++++++++++++++++++++++++++++++++
model_count :  5
batch_size :  60
number of models 5, batch_size in 60 average :  0.9926468973914557
++++++++++++++++++++++++++++++++++
model_count :  5
batch_size :  70
number of models 5, batch_size in 70 average :  0.9932550849572422
```

Selecting the top 10 train scores and model configurations.

In [83]:
```
np.sort(average)[::-1][:10]
```

Out[83]:
```
array([0.99373098, 0.99370371, 0.993627  , 0.99358132, 0.99340582,
       0.99340018, 0.99325508, 0.99323417, 0.99298218, 0.99295487])
```

In [84]:
```
np.array(models_n_batches)[np.argsort(average)[::-1][:10]]
```

Out[84]:
```
array(['number of models 6, batch_size in 80',
       'number of models 5, batch_size in 80',
       'number of models 8, batch_size in 80',
       'number of models 9, batch_size in 80',
       'number of models 10, batch_size in 80',
       'number of models 7, batch_size in 80',
       'number of models 5, batch_size in 70',
       'number of models 9, batch_size in 70',
       'number of models 8, batch_size in 70',
       'number of models 6, batch_size in 70'], dtype='<U37')
```

Now I pass the D2 set to each of these model_count models(hyper parameter), now I will get that many predictions for D2, from each of these models, which in turn will become my new dataset.

In [85]:
```
model_count=[6,5,8,9,10,7,5,9,8,6]
batch_size=[80,80,80,80,80,80,70,70,70,70]
```

In [86]:
```
data_type="adasyn_data"

for m_x,b_x in zip(model_count,batch_size):
    print("number of models "+str(m_x)+", batch_size in "+str(b_x))

    xxx=pd.DataFrame()

    for i in range(1,m_x+1):

        #Load the model.
        pickle_object = open("DT_model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data
                             'rb')
        model = pickle.load(pickle_object)
        pickle_object.close()

        #Make the prediction.
        y_pred=model.predict(D2_adasyn_x_train)

        #Assign the prediction to a new column in the current dataframe
        xxx[str(i)]=y_pred
        del model

    xxx.to_pickle("data_model_count_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type))
```

```
number of models 6, batch_size in 80
number of models 5, batch_size in 80
number of models 8, batch_size in 80
number of models 9, batch_size in 80
number of models 10, batch_size in 80
number of models 7, batch_size in 80
number of models 5, batch_size in 70
number of models 9, batch_size in 70
number of models 8, batch_size in 70
number of models 6, batch_size in 70
```

Now, this new dataset that I have, and the D2_Y values that I have ,I will train new meta model, with 10 configurations. and select the top meta model.

In [87]:
```
Meta_Model_input_list_adasyn_data=[]
Meta_Model_param_list_adasyn_data=[]
Meta_Model_score_list_adasyn_data=[]
data_type="adasyn_data"

for m_x,b_x in zip(model_count,batch_size):
    print("number of models "+str(m_x)+", batch_size in "+str(b_x))
    x_data=pd.read_pickle("data_model_count_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type))
    y_data=D2_adasyn_y_train.copy()
    param,score=train_the_meta_model(x_data,y_data,data_type,m_x,b_x)

    Meta_Model_input_list_adasyn_data.append("Meta_model_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(
    Meta_Model_param_list_adasyn_data.append(param)
    Meta_Model_score_list_adasyn_data.append(score)

np.save("Meta_Model_input_list_adasyn_data.npy",Meta_Model_input_list_adasyn_data)
np.save("Meta_Model_param_list_adasyn_data.npy",Meta_Model_param_list_adasyn_data)
np.save("Meta_Model_score_list_adasyn_data.npy",Meta_Model_score_list_adasyn_data)
Meta_Model_input_list_adasyn_data=np.load("Meta_Model_input_list_adasyn_data.npy",allow_pickle=True)
Meta_Model_param_list_adasyn_data=np.load("Meta_Model_param_list_adasyn_data.npy",allow_pickle=True)
Meta_Model_score_list_adasyn_data=np.load("Meta_Model_score_list_adasyn_data.npy",allow_pickle=True)
```

```
number of models 6, batch_size in 80
Saving Meta_model_6_batch_size_80_percent_adasyn_data
Saving best_params_for_Meta_model_6_batch_size_80_percent_adasyn_data_{'max_depth': 6, 'min_samples_spli
t': 10}
Saving best_scores_for_Meta_model_6_batch_size_80_percent_adasyn_data_0.9975046159273775
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
number of models 5, batch_size in 80
Saving Meta_model_5_batch_size_80_percent_adasyn_data
Saving best_params_for_Meta_model_5_batch_size_80_percent_adasyn_data_{'max_depth': 6, 'min_samples_spli
t': 10}
Saving best_scores_for_Meta_model_5_batch_size_80_percent_adasyn_data_0.996884563842344
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
number of models 8, batch_size in 80
Saving Meta_model_8_batch_size_80_percent_adasyn_data
```

```
Saving best_params_for_Meta_model_8_batch_size_80_percent_adasyn_data_{'max_depth': 10, 'min_samples_spli
```

Selecting the top Meta Model with the best CV scores and model configurations on which the main test data.

In [88]:
```
np.sort(Meta_Model_score_list_adasyn_data)[::-1][0]
```

Out[88]:
```
0.9978210942805115
```

In [89]:
```
Meta_Model_param_list_adasyn_data[np.argsort(Meta_Model_score_list_adasyn_data)[::-1][0]]
```

Out[89]:
```
{'max_depth': 6, 'min_samples_split': 2}
```

In [90]:
```
Meta_Model_input_list_adasyn_data[np.argsort(Meta_Model_score_list_adasyn_data)[::-1][0]]
```

Out[90]:
```
'Meta_model_10_batch_size_80_percent_adasyn_data'
```

In [91]:
```
model_count=[10]
batch_size=[80]
for m_x,b_x in zip(model_count,batch_size):
    print("number of models "+str(m_x)+", batch_size in "+str(b_x))
```

```
number of models 10, batch_size in 80
```

Passing the test set that I have, to each of the base models with configurations with top 10 scores so that I get "model_count" predictions.

In [92]:
```
model_count=[6,5,8,9,10,7,5,9,8,6]
batch_size=[80,80,80,80,80,80,70,70,70,70]
```

In [93]:
```
data_type="adasyn_data"
for m_x,b_x in zip(model_count,batch_size):
    print("number of models "+str(m_x)+", batch_size in "+str(b_x))

    xxx=pd.DataFrame()

    for i in range(1,m_x+1):

        #Load the model.
        pickle_object = open("DT_model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+\
                             str(data_type)+".pickle",'rb')
        model = pickle.load(pickle_object)
        pickle_object.close()

        #Make the prediction.
        y_pred=model.predict(new_standard_test_adasyn_all_features_df)

        #Assign the prediction to a new column in the current dataframe
        xxx[str(i)]=y_pred
        del model

    xxx.to_pickle("test_data_model_count_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type))
```

```
number of models 6, batch_size in 80
number of models 5, batch_size in 80
number of models 8, batch_size in 80
number of models 9, batch_size in 80
number of models 10, batch_size in 80
number of models 7, batch_size in 80
number of models 5, batch_size in 70
number of models 9, batch_size in 70
number of models 8, batch_size in 70
number of models 6, batch_size in 70
```

Now I pass this new dataset and pass it to the meta_model to get the final prediction.

In [94]:
```
model_count=[10]
batch_size=[80]
```

In [95]:
```
data_type="adasyn_data"

for m_x,b_x in zip(model_count,batch_size):
    print("number of models "+str(m_x)+", batch_size in "+str(b_x))

    #Load the model.
    pickle_object = open("Meta_model_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type)+".pickle"
                         'rb')
    model = pickle.load(pickle_object)
    pickle_object.close()

    #Load the data.
    x_test=pd.read_pickle("test_data_model_count_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_typ
    y_test=y_test_adasyn_all_features.copy()

    #Make the prediction.
    y_pred=model.predict(x_test)

    #Now try and figure out the F2 Score.
    f2_score=fbeta_score(y_test, y_pred, pos_label=1.0, beta=2)
    print("f2_score : ",f2_score)
    print("confusion_matrix : \n",confusion_matrix(y_test, y_pred))
```

```
number of models 10, batch_size in 80
f2_score :  0.8595988538681949
confusion_matrix :
 Predicted    0.0  1.0
True
0.0        11733   67
1.0           20  180
```

Repeat the whole process with smotetomek Data now.

In [19]:
```
model_count=[5,6,7,8,9,10]
batch_size=[0.3,0.4,0.5,0.6,0.7,0.8]
```

In [ ]:
```
%%time
data_type="smotetomek_data"
input_list_smotetomek_data=[]
param_list_smotetomek_data=[]
score_list_smotetomek_data=[]
for m_x in model_count:
    for b_x in batch_size:
        for i in range(1,m_x+1):
            param,score=train_the_decision_trees(D1_smotetomek_x_train,D1_smotetomek_y_train,data_type,i,m_x
            input_list_smotetomek_data.append("model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_"+da
            param_list_smotetomek_data.append(param)
            score_list_smotetomek_data.append(score)
```

```
Saving DT_model_1_of_5_batch_size_30_percent_smotetomek_data
Saving best_params_for_model_1_of_5_batch_size_30_percent_smotetomek_data_{'max_depth': 10, 'min_samples_
split': 2}
Saving best_scores_for_model_1_of_5_batch_size_30_percent_smotetomek_data_0.9880863209256343
Saving 30 percent data set batch used to train the model 1 out of 5
Saving x_batch_D1_for_model_1_of_5_batch_size_30_percent_smotetomek_data
Saving y_batch_D1_for_model_1_of_5_batch_size_30_percent_smotetomek_data
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Saving DT_model_2_of_5_batch_size_30_percent_smotetomek_data
Saving best_params_for_model_2_of_5_batch_size_30_percent_smotetomek_data_{'max_depth': 6, 'min_samples_s
plit': 10}
Saving best_scores_for_model_2_of_5_batch_size_30_percent_smotetomek_data_0.9903127787538797
Saving 30 percent data set batch used to train the model 2 out of 5
Saving x_batch_D1_for_model_2_of_5_batch_size_30_percent_smotetomek_data
Saving y_batch_D1_for_model_2_of_5_batch_size_30_percent_smotetomek_data
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

. . . . . .
```

Take all these models batch wise meaning taking them and run D2 into them to get the result.

In [ ]:
```
np.save("input_list_smotetomek_data",input_list_smotetomek_data)
np.save("param_list_smotetomek_data",param_list_smotetomek_data)
np.save("score_list_smotetomek_data",score_list_smotetomek_data)
```

In [20]:
```
input_list_smotetomek_data=np.load("input_list_smotetomek_data.npy",allow_pickle=True)
param_list_smotetomek_data=np.load("param_list_smotetomek_data.npy",allow_pickle=True)
score_list_smotetomek_data=np.load("score_list_smotetomek_data.npy",allow_pickle=True)
```

Selecting only the top 10 dataset and model config to train my Meta model that gives me the highest cv score.

In [21]:
```
%%time
counter=0
average=[]
models_n_batches=[]
for m_x in model_count:
    for b_x in batch_size:
        b_x=int(b_x*100)
        xxx=0
        val=0
        print("model_count : ",m_x)
        print("batch_size : ",b_x)
        for i in range(0,m_x):
            xxx=xxx+score_list_smotetomek_data[counter]
            counter=counter+1
            val=val+1
        print("number of models "+str(m_x)+", batch_size in "+str(b_x),"average : ",xxx/val)
        models_n_batches.append("number of models "+str(m_x)+", batch_size in "+str(b_x))
        average.append(xxx/val)
        print("++++++++++++++++++++++++++++++++++")
```

```
model_count :  5
batch_size :  30
number of models 5, batch_size in 30 average :  0.9898735063453581
++++++++++++++++++++++++++++++++++
model_count :  5
batch_size :  40
number of models 5, batch_size in 40 average :  0.9917268449922301
++++++++++++++++++++++++++++++++++
model_count :  5
batch_size :  50
number of models 5, batch_size in 50 average :  0.9920221696733952
++++++++++++++++++++++++++++++++++
model_count :  5
batch_size :  60
number of models 5, batch_size in 60 average :  0.9927832781439918
++++++++++++++++++++++++++++++++++
model_count :  5
batch_size :  70
number of models 5, batch_size in 70 average :  0.9937409517326514
```

Selecting the top 10 train scores and model configurations.

In [22]:
```
np.sort(average)[::-1][:10]
```

Out[22]:
```
array([0.99428008, 0.99427171, 0.99421226, 0.994187  , 0.99409098,
       0.99397567, 0.99392012, 0.99387321, 0.99377578, 0.99377191])
```

In [23]:
```
np.array(models_n_batches)[np.argsort(average)[::-1][:10]]
```

Out[23]:
```
array(['number of models 7, batch_size in 80',
       'number of models 5, batch_size in 80',
       'number of models 6, batch_size in 80',
       'number of models 8, batch_size in 80',
       'number of models 9, batch_size in 80',
       'number of models 10, batch_size in 80',
       'number of models 8, batch_size in 70',
       'number of models 7, batch_size in 70',
       'number of models 9, batch_size in 70',
       'number of models 10, batch_size in 70'], dtype='<U37')
```

Now I pass the D2 set to each of these model_count models(hyper parameter), now I will get that many predictions for D2, from each of these models, which in turn will become my new dataset.

In [24]:
```
model_count=[7,5,6,8,9,10,8,7,9,10]
batch_size=[80,80,80,80,80,80,70,70,70,70]
```

In [25]:
```
data_type="smotetomek_data"

for m_x,b_x in zip(model_count,batch_size):
    print("number of models "+str(m_x)+", batch_size in "+str(b_x))

    xxx=pd.DataFrame()

    for i in range(1,m_x+1):

        #Load the model.
        pickle_object = open("DT_model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data
                             'rb')
        model = pickle.load(pickle_object)
        pickle_object.close()

        #Make the prediction.
        y_pred=model.predict(D2_smotetomek_x_train)

        #Assign the prediction to a new column in the current dataframe
        xxx[str(i)]=y_pred
        del model

    xxx.to_pickle("data_model_count_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type))
```

```
number of models 7, batch_size in 80
number of models 5, batch_size in 80
number of models 6, batch_size in 80
number of models 8, batch_size in 80
number of models 9, batch_size in 80
number of models 10, batch_size in 80
number of models 8, batch_size in 70
number of models 7, batch_size in 70
number of models 9, batch_size in 70
number of models 10, batch_size in 70
```

Now, this new dataset that I have, and the D2_Y values that I have ,I will train new meta model, with 10 configurations. and select the top meta model.

In [26]:
```
Meta_Model_input_list_smotetomek_data=[]
Meta_Model_param_list_smotetomek_data=[]
Meta_Model_score_list_smotetomek_data=[]
data_type="smotetomek_data"

for m_x,b_x in zip(model_count,batch_size):
    print("number of models "+str(m_x)+", batch_size in "+str(b_x))
    x_data=pd.read_pickle("data_model_count_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type))
    y_data=D2_smotetomek_y_train.copy()
    param,score=train_the_meta_model(x_data,y_data,data_type,m_x,b_x)

    Meta_Model_input_list_smotetomek_data.append("Meta_model_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+
    Meta_Model_param_list_smotetomek_data.append(param)
    Meta_Model_score_list_smotetomek_data.append(score)

np.save("Meta_Model_input_list_smotetomek_data.npy",Meta_Model_input_list_smotetomek_data)
np.save("Meta_Model_param_list_smotetomek_data.npy",Meta_Model_param_list_smotetomek_data)
np.save("Meta_Model_score_list_smotetomek_data.npy",Meta_Model_score_list_smotetomek_data)
Meta_Model_input_list_smotetomek_data=np.load("Meta_Model_input_list_smotetomek_data.npy",allow_pickle=True)
Meta_Model_param_list_smotetomek_data=np.load("Meta_Model_param_list_smotetomek_data.npy",allow_pickle=True)
Meta_Model_score_list_smotetomek_data=np.load("Meta_Model_score_list_smotetomek_data.npy",allow_pickle=True)
```

```
number of models 7, batch_size in 80
Saving Meta_model_7_batch_size_80_percent_smotetomek_data
Saving best_params_for_Meta_model_7_batch_size_80_percent_smotetomek_data_{'max_depth': 4, 'min_samples_s
plit': 50}
Saving best_scores_for_Meta_model_7_batch_size_80_percent_smotetomek_data_0.9975844143538568
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
number of models 5, batch_size in 80
Saving Meta_model_5_batch_size_80_percent_smotetomek_data
Saving best_params_for_Meta_model_5_batch_size_80_percent_smotetomek_data_{'max_depth': 4, 'min_samples_s
plit': 50}
Saving best_scores_for_Meta_model_5_batch_size_80_percent_smotetomek_data_0.9962059755851845
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
number of models 6, batch_size in 80
Saving Meta_model_6_batch_size_80_percent_smotetomek_data
```

```
Saving best_params_for_Meta_model_6_batch_size_80_percent_smotetomek_data_{'max_depth': 4, 'min_samples_s
```

Selecting the top Meta Model with the best CV scores and model configurations on which the main test data.

In [28]:
```
np.sort(Meta_Model_score_list_smotetomek_data)[::-1][0]
```

Out[28]:
```
0.9975844143538568
```

In [29]:
```
Meta_Model_input_list_smotetomek_data[np.argsort(Meta_Model_score_list_smotetomek_data)[::-1][0]]
```

Out[29]:
```
'Meta_model_7_batch_size_80_percent_smotetomek_data'
```

In [30]:
```
Meta_Model_param_list_smotetomek_data[np.argsort(Meta_Model_score_list_smotetomek_data)[::-1][0]]
```

Out[30]:
```
{'max_depth': 4, 'min_samples_split': 50}
```

In [ ]:
```
model_count=[7]
batch_size=[80]
for m_x,b_x in zip(model_count,batch_size):
    print("number of models "+str(m_x)+", batch_size in "+str(b_x))
```

```
number of models 7, batch_size in 80
```

Passing the test set that I have, to each of the base models with configurations with top 10 scores so that I get "model_count" predictions.

In [ ]:
```
model_count=[7,5,6,8,9,10,8,7,9,10]
batch_size=[80,80,80,80,80,80,70,70,70,70]
```

In [ ]:
```
data_type="smotetomek_data"
for m_x,b_x in zip(model_count,batch_size):
    print("number of models "+str(m_x)+", batch_size in "+str(b_x))

    xxx=pd.DataFrame()

    for i in range(1,m_x+1):

        #Load the model.
        pickle_object = open("DT_model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+\
                             str(data_type)+".pickle",'rb')
        model = pickle.load(pickle_object)
        pickle_object.close()

        #Make the prediction.
        y_pred=model.predict(new_standard_test_smotetomek_all_features_df)

        #Assign the prediction to a new column in the current dataframe
        xxx[str(i)]=y_pred
        del model

    xxx.to_pickle("test_data_model_count_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type))
```

```
number of models 7, batch_size in 80
number of models 5, batch_size in 80
number of models 6, batch_size in 80
number of models 8, batch_size in 80
number of models 9, batch_size in 80
number of models 10, batch_size in 80
number of models 8, batch_size in 70
number of models 7, batch_size in 70
number of models 9, batch_size in 70
number of models 10, batch_size in 70
```

Now I pass this new dataset and pass it to the meta_model to get the final prediction.

In [ ]:
```
model_count=[7]
batch_size=[80]
```

In [ ]:
```
data_type="smotetomek_data"

for m_x,b_x in zip(model_count,batch_size):
    print("number of models "+str(m_x)+", batch_size in "+str(b_x))

    #Load the model.
    pickle_object = open("Meta_model_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type)+".pickle"
                         'rb')
    model = pickle.load(pickle_object)
    pickle_object.close()

    #Load the data.
    x_test=pd.read_pickle("test_data_model_count_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_typ
    y_test=y_test_smotetomek_all_features.copy()

    #Make the prediction.
    y_pred=model.predict(x_test)

    #Now try and figure out the F2 Score.
    f2_score=fbeta_score(y_test, y_pred, pos_label=1.0, beta=2)
    print("f2_score : ",f2_score)
    print("confusion_matrix : \n",confusion_matrix(y_test, y_pred))
```

```
number of models 7, batch_size in 80
f2_score :  0.8737864077669903
confusion_matrix :
 Predicted    0.0  1.0
True
0.0        11750   50
1.0           20  180
```

custom_ensemble function

In [18]:
```
def custom_ensemble(x_train,y_train,x_test, n_estimators):


    """x_train = Train data
       y_train = Labels for train data
       x_test = The test dataset
       n_estimators = number of base models.
    The function should do everything from dividing X_train to D1 and D2 to predictions from meta model.
    This function returns the predictions for X_test."""

    #Splitting the train set into D1 and D2(50-50)
    D1_x_train, D2_x_train, D1_y_train, D2_y_train = \
    train_test_split(x_train, y_train, test_size=0.50, stratify=y_train)

    #Now from this D1 do sampling with replacement to create d1,d2,d3....dk(k samples)

    #I will create model_count samples(hyperparameter). Each sample containing randomly (with replacement),
    #batch_size% of the data(hyperparameter).

    model_count=[n_estimators]
    batch_size=[0.3,0.4,0.5,0.6,0.7,0.8]

    data_type="function"
    input_list_data=[]
    param_list_data=[]
    score_list_data=[]
    for m_x in model_count:
        for b_x in batch_size:
            for i in range(1,m_x+1):
                param,score=train_the_decision_trees(D1_x_train,D1_y_train,data_type,i,m_x,b_x)
                input_list_data.append("DT_model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_"+data_t
                param_list_data.append(param)
                score_list_data.append(score)

    #Take all these models batch wise meaning taking them and run D2 into them to get the result.

    np.save("input_list_data",input_list_data)
    np.save("param_list_data",param_list_data)
    np.save("score_list_data",score_list_data)

    input_list_data=np.load("input_list_data.npy",allow_pickle=True)
```

```
param_list_data=np.load("param_list_data.npy",allow_pickle=True)
score_list_data=np.load("score_list_data.npy",allow_pickle=True)

#Selecting only the top 10 dataset and model config to train my Meta model that gives me the highest cv
counter=0
average=[]
models=[]
batches=[]
for m_x in model_count:
    for b_x in batch_size:
        b_x=int(b_x*100)
        xxx=0
        val=0
        print("model_count : ",m_x)
        print("batch_size : ",b_x)
        for i in range(0,m_x):
            xxx=xxx+score_list_data[counter]
            counter=counter+1
            val=val+1
        print("number of models "+str(m_x)+", batch_size in "+str(b_x),"average : ",xxx/val)
        models.append(int(m_x))
        batches.append(int(b_x))
        average.append(xxx/val)
        print("++++++++++++++++++++++++++++++++++")

#Selecting the top 10 train scores and model configurations.
print("Top 10 Averages :", np.sort(average)[::-1][:10])


model_count=list(np.array(models)[np.argsort(average)[::-1][:10]])
batch_size=list(np.array(batches)[np.argsort(average)[::-1][:10]])


#Now I pass the D2 set to each of these model_count models(hyper parameter),
#now I will get that many predictions for D2, from each of these models, which in turn will
#become my new dataset.
for m_x,b_x in zip(model_count,batch_size):
    print("number of models "+str(m_x)+", batch_size in "+str(b_x))

    xxx=pd.DataFrame()

    for i in range(1,m_x+1):
```

```
            #Load the model.
            pickle_object = open("DT_model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(
                                 'rb')
            model = pickle.load(pickle_object)
            pickle_object.close()

            #Make the prediction.
            y_pred=model.predict(D2_x_train)

            #Assign the prediction to a new column in the current dataframe
            xxx[str(i)]=y_pred
            del model

        xxx.to_pickle("data_model_count_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type))


    #Now, this new dataset that I have, and the D2_Y values that I have ,I  will train new meta model, with
    #10 configurations. and select the top meta model.
    Meta_Model_input_list_data=[]
    Meta_Model_param_list_data=[]
    Meta_Model_score_list_data=[]

    for m_x,b_x in zip(model_count,batch_size):
        print("number of models "+str(m_x)+", batch_size in "+str(b_x))
        x_data=pd.read_pickle("data_model_count_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type
        y_data=D2_y_train.copy()
        param,score=train_the_meta_model(x_data,y_data,data_type,m_x,b_x)

        Meta_Model_input_list_data.append("Meta_model_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(dat
        Meta_Model_param_list_data.append(param)
        Meta_Model_score_list_data.append(score)

    np.save("Meta_Model_input_list_data.npy",Meta_Model_input_list_data)
    np.save("Meta_Model_param_list_data.npy",Meta_Model_param_list_data)
    np.save("Meta_Model_score_list_data.npy",Meta_Model_score_list_data)

    Meta_Model_input_list_data=np.load("Meta_Model_input_list_data.npy",allow_pickle=True)
    Meta_Model_param_list_data=np.load("Meta_Model_param_list_data.npy",allow_pickle=True)
    Meta_Model_score_list_data=np.load("Meta_Model_score_list_data.npy",allow_pickle=True)

    #Selecting the top Meta Model with the best CV scores and model configurations on which the main test da
```

```
print("Best Meta Model score : ",np.sort(Meta_Model_score_list_data)[::-1][0])

print("Best Meta Model input : ",Meta_Model_input_list_data[np.argsort(Meta_Model_score_list_data)[::-1]

print("Best Meta Model params : ",Meta_Model_param_list_data[np.argsort(Meta_Model_score_list_data)[::-1

#Passing the test set that I have, to each of the base models with configurations with top 10 scores so

for m_x,b_x in zip(model_count,batch_size):
    print("number of models "+str(m_x)+", batch_size in "+str(b_x))

    xxx=pd.DataFrame()

    for i in range(1,m_x+1):

        #Load the model.
        pickle_object = open("DT_model_"+str(i)+"_of_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+\
                             str(data_type)+".pickle",'rb')
        model = pickle.load(pickle_object)
        pickle_object.close()

        #Make the prediction.
        y_pred=model.predict(x_test)

        #Assign the prediction to a new column in the current dataframe
        xxx[str(i)]=y_pred
        del model

    xxx.to_pickle("test_data_model_count_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type))

batch_size=[int(Meta_Model_input_list_data[np.argsort(Meta_Model_score_list_data)[::-1][0]].split("_")[5
model_count=[int(Meta_Model_input_list_data[np.argsort(Meta_Model_score_list_data)[::-1][0]].split("_")[

#Now I pass this new dataset and pass it to the meta_model to get the final prediction.

for m_x,b_x in zip(model_count,batch_size):
    print("number of models "+str(m_x)+", batch_size in "+str(b_x))

    #Load the model.
    pickle_object = open("Meta_model_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data_type)+".pic
                         'rb')
```

```
        model = pickle.load(pickle_object)
        pickle_object.close()

        #Load the data.
        x_test=pd.read_pickle("test_data_model_count_"+str(m_x)+"_batch_size_"+str(b_x)+"_percent_"+str(data

        #Make the prediction.
        y_pred=model.predict(x_test)

    return y_pred
```

In [19]:
```
y_pred = custom_ensemble(new_x_smotetomek_df,
                         y_smotetomek,
                         new_standard_test_smotetomek_all_features_df,
                         7)
```

```
Saving DT_model_1_of_7_batch_size_30_percent_function
Saving best_params_for_model_1_of_7_batch_size_30_percent_function_{'max_depth': 8, 'min_samples_split':
2}
Saving best_scores_for_model_1_of_7_batch_size_30_percent_function_0.9908195076103652
Saving 30 percent data set batch used to train the model 1 out of 7
Saving x_batch_D1_for_model_1_of_7_batch_size_30_percent_function
Saving y_batch_D1_for_model_1_of_7_batch_size_30_percent_function
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Saving DT_model_2_of_7_batch_size_30_percent_function
Saving best_params_for_model_2_of_7_batch_size_30_percent_function_{'max_depth': 6, 'min_samples_split':
20}
Saving best_scores_for_model_2_of_7_batch_size_30_percent_function_0.9886914726438663
Saving 30 percent data set batch used to train the model 2 out of 7
Saving x_batch_D1_for_model_2_of_7_batch_size_30_percent_function
Saving y_batch_D1_for_model_2_of_7_batch_size_30_percent_function
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

..........................................................................................
```

Comparitive summary of all the models in the Pretty Table

In [24]:
```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Model","Data","Hyperparameters","CV F2 Score","Test F2 Score"]

x.add_row(["GBDT_XGB","Smotetomek_117_Features","{'n_estimators':500,'max_depth':8,'learning_rate':0.1}","0.
x.add_row(["GBDT_XGB","Median_127_Features","{'n_estimators':300,'max_depth':4,'learning_rate':0.3}","0.9395
x.add_row(["GBDT_AdaBoost","Smotetomek_117_Features","{'n_estimators':1000,'learning_rate':0.7}","0.99935","
x.add_row(["GBDT_XGB","Adasyn_144_Features","{'n_estimators':500,'max_depth':12,'learning_rate':0.05}","0.99
x.add_row(["GBDT_AdaBoost","Median_127_Features","{'n_estimators':1000,'learning_rate':0.3}","0.92683","0.94
x.add_row(["RF","Smotetomek_117_Features","{'max_depth':24,'n_estimators':100}","0.99902","0.92899"])
x.add_row(["RF","Median_127_Features","{'max_depth':18,'n_estimators':100}","0.90227","0.91633"])
x.add_row(["GBDT_AdaBoost","Adasyn_144_Features","{'n_estimators':200,'learning_rate':0.5}","0.99401","0.912
x.add_row(["RF","Adasyn_144_Features","{'max_depth':18,'n_estimators':50}","0.99476","0.91216"])
x.add_row(["MLP","Smotetomek_117_Features","200|0.1|300|0.2|400|0.3|500|0.4|600|0.5,relu","0.99969","0.91045
x.add_row(["MLP","Adasyn_144_Features","200|0.1|300|0.2|400|0.3|500|0.4|600|0.5,relu","0.99983","0.89552"])
x.add_row(["MLP","Median_127_Features","200|0.1|300|0.2|400|0.3|500|0.4|600|0.5,relu","0.91076","0.89458"])
x.add_row(["Custom","Smotetomek_Batch_size_80%_Features_7","{'max_depth':4,'min_samples_split':50}","0.99758
x.add_row(["Custom","Adasyn_Batch_size_80%_Features_10","{'max_depth':6,'min_samples_split':2}","0.99782","0
x.add_row(["DT","Adasyn_144_Features","{'max_depth':12,'min_samples_split':2}","0.97742","0.85754"])
x.add_row(["DT","Smotetomek_117_Features","{'max_depth':17,'min_samples_split':2}","0.99606","0.8559"])
x.add_row(["DT","Median_127_Features","{'max_depth':12,'min_samples_split':2}","0.85617","0.84722"])
x.add_row(["Custom","Median_Batch_size_80%_Features_9","{'max_depth':9,'min_samples_split':20}","0.84376","0
x.add_row(["KNN","Smotetomek_117_Features","{'n_neighbors':11}","0.99437","0.78215"])
x.add_row(["LR_Lib","Median_127_Features","{'C':0.1}","0.78023","0.75929"])
x.add_row(["LR_Lib","Smotetomek_117_Features","{'C':100}","0.96688","0.73955"])
x.add_row(["KNN","Adasyn_144_Features","{'n_neighbors':31}","0.98078","0.70144"])
x.add_row(["LR_SGD_Lib","Median_127_Features","{'alpha':1e-06}","0.69976","0.68452"])
x.add_row(["SVM_SGD","Adasyn_144_Features","{'alpha':0.001}","0.92844","0.68448"])
x.add_row(["SVM_SGD","Median_127_Features","{'alpha':0.0001}","0.63208","0.68448"])
x.add_row(["LR_Lib","Adasyn_144_Features","{'C':100}","0.93735","0.67131"])
x.add_row(["LR_SGD_Lib","Smotetomek_117_Features","{'alpha':0.0001}","0.96295","0.66421"])
x.add_row(["SVM_SGD","Smotetomek_117_Features","{'alpha':0.0001}","0.96382","0.64996"])
x.add_row(["KNN","Median_127_Features","{'n_neighbors':11}","0.59931","0.62633"])
x.add_row(["LR_SGD_Lib","Adasyn_144_Features","{'alpha':1e-05}","0.93024","0.56628"])
print(x)
```

```
+---------------+----------------------------------------+-----------------------------------------------
---------+-------------+--------------+
|     Model     |                  Data                  |                    Hyperparameters
```

```
| CV F2 Score | Test F2 Score |
+---------------+---------------------------------------+-----------------------------------------------
---------+-------------+--------------+
|    GBDT_XGB   |       Smotetomek_117_Features        |  {'n_estimators':500,'max_depth':8,'learning_rat
e':0.1}  |   0.99944   |    0.97208    |
|    GBDT_XGB   |         Median_127_Features          |  {'n_estimators':300,'max_depth':4,'learning_rat
e':0.3}  |   0.93954   |     0.9718    |
| GBDT_AdaBoost |       Smotetomek_117_Features        |       {'n_estimators':1000,'learning_rate':0.7}
|   0.99935   |    0.96439    |
|    GBDT_XGB   |         Adasyn_144_Features          | {'n_estimators':500,'max_depth':12,'learning_rat
e':0.05} |   0.99626   |    0.96326    |
| GBDT_AdaBoost |         Median_127_Features          |       {'n_estimators':1000,'learning_rate':0.3}
|   0.92683   |    0.94444    |
|       RF      |       Smotetomek_117_Features        |          {'max_depth':24,'n_estimators':100}
|   0.99902   |    0.92899    |
|       RF      |         Median_127_Features          |          {'max_depth':18,'n_estimators':100}
|   0.90227   |    0.91633    |
| GBDT_AdaBoost |         Adasyn_144_Features          |        {'n_estimators':200,'learning_rate':0.5}
|   0.99401   |    0.91251    |
|       RF      |         Adasyn_144_Features          |           {'max_depth':18,'n_estimators':50}
|   0.99476   |    0.91216    |
|      MLP      |       Smotetomek_117_Features        |       200|0.1|300|0.2|400|0.3|500|0.4|600|0.5,re
lu       |   0.99969   |    0.91045    |
|      MLP      |         Adasyn_144_Features          |       200|0.1|300|0.2|400|0.3|500|0.4|600|0.5,re
lu       |   0.99983   |    0.89552    |
|      MLP      |         Median_127_Features          |       200|0.1|300|0.2|400|0.3|500|0.4|600|0.5,re
lu       |   0.91076   |    0.89458    |
|     Custom    | Smotetomek_Batch_size_80%_Features_7 |         {'max_depth':4,'min_samples_split':50}
|   0.99758   |    0.87378    |
|     Custom    |  Adasyn_Batch_size_80%_Features_10   |         {'max_depth':6,'min_samples_split':2}
|   0.99782   |    0.85959    |
|       DT      |         Adasyn_144_Features          |         {'max_depth':12,'min_samples_split':2}
|   0.97742   |    0.85754    |
|       DT      |       Smotetomek_117_Features        |         {'max_depth':17,'min_samples_split':2}
|   0.99606   |     0.8559    |
|       DT      |         Median_127_Features          |         {'max_depth':12,'min_samples_split':2}
|   0.85617   |    0.84722    |
|     Custom    |   Median_Batch_size_80%_Features_9   |         {'max_depth':9,'min_samples_split':20}
|   0.84376   |    0.81755    |
|      KNN      |       Smotetomek_117_Features        |                  {'n_neighbors':11}
|   0.99437   |    0.78215    |
|     LR_Lib    |         Median_127_Features          |                      {'C':0.1}
```

```
|   0.78023   |    0.75929    |
|     LR_Lib    |       Smotetomek_117_Features        |                        {'C':100}
|   0.96688   |    0.73955    |
|      KNN      |         Adasyn_144_Features          |                    {'n_neighbors':31}
|   0.98078   |    0.70144    |
|   LR_SGD_Lib  |         Median_127_Features          |                     {'alpha':1e-06}
|   0.69976   |    0.68452    |
|    SVM_SGD    |         Adasyn_144_Features          |                     {'alpha':0.001}
|   0.92844   |    0.68448    |
|    SVM_SGD    |         Median_127_Features          |                     {'alpha':0.0001}
|   0.63208   |    0.68448    |
|     LR_Lib    |         Adasyn_144_Features          |                        {'C':100}
|   0.93735   |    0.67131    |
|   LR_SGD_Lib  |       Smotetomek_117_Features        |                     {'alpha':0.0001}
|   0.96295   |    0.66421    |
|    SVM_SGD    |       Smotetomek_117_Features        |                     {'alpha':0.0001}
|   0.96382   |    0.64996    |
|      KNN      |         Median_127_Features          |                    {'n_neighbors':11}
|   0.59931   |    0.62633    |
|   LR_SGD_Lib  |         Adasyn_144_Features          |                     {'alpha':1e-05}
|   0.93024   |    0.56628    |
+---------------+--------------------------------------+-----------------------------------------------
---------+-------------+---------------+
```

Conclusion : Here I am getting the best score with smotetomek median data with XGBoost library.
with cv score as 0.99944 and test score as : 0.97208
I also tried with other models and libraries.