In [304]:

```
#Best Model - Simple SVM
#Best cross-validation score: 0.906
#Best Parameters {'C': 0.001, 'multi_class': 'crammer_singer', 'penalty': 'l1'}
#train score:  0.904
#test score:  1.0

#Three things i've tried that were not covered in class are-
#-->Randomized search CV with MLPClassifier model
#-->Saving and Uploading models using pickle
#-->GaussianProcessClassifier

#Comments
#--worked a lot on hyperparmeter tuning and selected the most appropriate so tha
#Referred https://scikit-learn.org/ for hyperparameters
#--used randomsampling of the train dataset to run models faster
#--I've split the train dataset into train and test so that i can verify how my
#  before predicting the test dataset provided and uploading on kaggle.
#--Made sure to resolve all the warnings

```

In [399]:

```
#loading test dataset into "test"
import pandas as pd

test= pd.read_csv(r'/Users/Desktop/MLPROJECT2/test.csv')
```

In [400]:

```
#Random Sampling of train data and loading into "data"
import random

f = "/Users/Desktop/MLPROJECT2/train.csv"
num_lines = sum(1 for l in open(f))
size = int(num_lines /10)
skip_idx = random.sample(range(1, num_lines), num_lines - size)
data = pd.read_csv(f, skiprows=skip_idx)
```

In [401]:

```
#to see the data to get an understanding
data.head()
```

Out[401]:

	Id	V1	V2	V3	V4	V5	V6	V7	
0	245376	1.812653	-0.476162	-0.338988	1.386750	-0.745965	-0.449870	-0.492226	0.0299
1	202483	-1.014219	0.522775	-0.337978	-1.957797	3.578395	3.266965	0.602857	0.6446
2	221275	-0.600317	0.915481	0.835746	-0.621856	1.284060	-0.070605	1.042698	-0.0725
3	155702	-0.524867	0.751315	2.278360	-0.295041	0.303778	0.164136	0.454016	-0.1106
4	76252	-16.772703	-14.426415	-5.606166	2.828980	-4.856624	1.538447	3.147259	-0.8755

5 rows × 31 columns

In [402]:

```
# Find out number or rows, columns and datatypes of all variables
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2483 entries, 0 to 2482
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Id      2483 non-null   int64  
 1   V1      2250 non-null   float64
 2   V2      2483 non-null   float64
 3   V3      2483 non-null   float64
 4   V4      2483 non-null   float64
 5   V5      2483 non-null   float64
 6   V6      2483 non-null   float64
 7   V7      2483 non-null   float64
 8   V8      2483 non-null   float64
 9   V9      2483 non-null   float64
 10  V10     2483 non-null   float64
 11  V11     2483 non-null   float64
 12  V12     2483 non-null   float64
 13  V13     2483 non-null   float64
 14  V14     2483 non-null   float64
 15  V15     2483 non-null   float64
 16  V16     2483 non-null   float64
 17  V17     2483 non-null   float64
 18  V18     2483 non-null   float64
 19  V19     2483 non-null   float64
 20  V20     2232 non-null   float64
 21  V21     2483 non-null   float64
 22  V22     2483 non-null   float64
 23  V23     2483 non-null   float64
 24  V24     2483 non-null   float64
 25  V25     2483 non-null   float64
 26  V26     2483 non-null   float64
 27  V27     2483 non-null   float64
 28  V28     2483 non-null   float64
 29  V29     2483 non-null   float64
 30  Target  2483 non-null   int64  
dtypes: float64(29), int64(2)
memory usage: 601.5 KB
```

In [403]:

```
#To find corr between variables and drop highly correlated values if any
data.corr()
```

Out[403]:

	Id	V1	V2	V3	V4	V5	V6	V7	
Id	1.000000	0.106002	-0.049306	-0.308952	-0.150191	0.184762	-0.062611	0.091785	-0
V1	0.106002	1.000000	0.045613	0.238650	-0.128216	0.149235	0.014607	0.058938	-0
V2	-0.049306	0.045613	1.000000	-0.048024	0.066016	-0.001432	-0.085869	-0.232418	0
V3	-0.308952	0.238650	-0.048024	1.000000	-0.146819	0.180421	0.008765	0.136767	-0
V4	-0.150191	-0.128216	0.066016	-0.146819	1.000000	-0.082313	-0.058040	-0.055540	0
V5	0.184762	0.149235	-0.001432	0.180421	-0.082313	1.000000	0.025338	0.052438	-0
V6	-0.062611	0.014607	-0.085869	0.008765	-0.058040	0.025338	1.000000	0.070925	-0
V7	0.091785	0.058938	-0.232418	0.136767	-0.055540	0.052438	0.070925	1.000000	-0
V8	-0.026564	-0.026575	0.053217	-0.083000	0.037203	-0.048890	-0.069330	-0.053344	1
V9	0.014134	0.014293	-0.115891	0.111096	-0.108221	0.028536	0.034843	0.149379	-0
V10	0.041763	0.132935	-0.162679	0.241818	-0.148472	0.119695	0.078622	0.224967	-0
V11	-0.219083	-0.120230	0.058261	-0.188455	0.146493	-0.111462	-0.033933	-0.130512	0
V12	0.119889	0.121350	-0.134130	0.262103	-0.219359	0.101383	0.093871	0.211309	-0
V13	-0.045369	-0.027993	0.011601	-0.048120	0.000784	-0.027940	0.000500	-0.004749	0
V14	-0.071924	0.118215	-0.111523	0.290048	-0.242955	0.142852	0.077512	0.207104	-0
V15	-0.170863	-0.032225	-0.044510	-0.042316	-0.003529	-0.027638	0.021803	0.017806	-0
V16	0.015783	0.075772	-0.115365	0.181669	-0.106318	0.123865	0.089122	0.209749	-0
V17	-0.004735	0.211444	-0.121221	0.265299	-0.161811	0.198831	0.074465	0.242020	-0
V18	0.108321	0.112264	-0.011928	0.146607	-0.093861	0.110579	-0.015746	0.107366	-0
V19	0.009758	-0.020820	0.028932	-0.051656	0.004942	-0.084806	0.003313	-0.026424	0
V20	-0.035656	0.158838	0.299548	0.044290	-0.053365	0.101502	-0.087152	-0.232085	0
V21	0.056391	0.015946	0.112762	0.015648	-0.037808	0.052674	-0.088302	-0.028338	0
V22	0.157150	0.013514	-0.011440	0.017954	-0.011248	0.008160	0.030719	0.014045	-0
V23	0.016082	0.161596	0.138701	0.066591	-0.074991	0.008635	0.002211	-0.003186	-0
V24	-0.014286	0.025835	-0.027349	-0.009417	-0.064187	0.004263	0.044203	0.026326	0
V25	-0.203261	0.038595	0.027026	0.001164	0.023302	-0.030131	-0.011469	-0.017525	0
V26	-0.036459	-0.051888	-0.023069	-0.010423	0.028182	0.030107	0.002508	-0.019956	0
V27	-0.018267	-0.127714	-0.091841	-0.105243	0.081911	0.007896	-0.018641	-0.064976	0
V28	-0.035357	0.161877	0.142665	0.037479	-0.022143	-0.024816	0.006421	-0.036404	0
V29	-0.011759	-0.278976	-0.544283	-0.227559	0.102688	-0.363458	0.228540	0.414344	-0
Target	-0.013973	-0.185015	0.195992	-0.374903	0.312903	-0.167710	-0.120355	-0.270939	0

31 rows × 31 columns

In [404]:

```
# Column with Null Values and it's count in Train dataset
null_values= data.isnull().sum()
for key,value in null_values.items():
    if value >0:
        print(key,":",value)
```

```
V1 : 233
V20 : 251
```

In [405]:

```
# Column with Null Values and it's count in test dataset
null_values= test.isnull().sum()
for key,value in null_values.items():
    if value >0:
        print(key,":",value)
```

```
V1 : 2499
V20 : 2504
```

In [406]:

```
#No of unique values in each column in train set
total_unique_values= data.nunique()
for key,value in total_unique_values.items():
    if value >0:
        print(key,":",value)
```

```
Id : 2483
V1 : 2250
V2 : 2482
V3 : 2482
V4 : 2482
V5 : 2482
V6 : 2482
V7 : 2482
V8 : 2482
V9 : 2482
V10 : 2482
V11 : 2482
V12 : 2482
V13 : 2482
V14 : 2482
V15 : 2482
V16 : 2482
V17 : 2482
V18 : 2482
V19 : 2482
V20 : 2231
V21 : 2482
V22 : 2482
V23 : 2482
V24 : 2482
V25 : 2482
V26 : 2482
V27 : 2482
V28 : 2482
V29 : 1452
Target : 2
```

In [407]:

```
#Split Train dataset into train and test 70-30
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data.drop(['Target'], axis=1
                                                    data['Target'],
                                                    test_size=0.3,
                                                    random_state=0)

X_train.shape, X_test.shape
```

Out[407]:

```
((1738, 30), (745, 30))
```

In [408]:

```
#number of columns in test to check if they match with train dataset
print(test.shape)
```

```
(24846, 30)
```

In [409]:

```
#feature engineering
from sklearn.pipeline import Pipeline
from feature_engine import missing_data_imputers as mdi
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler


    #Replacing Null values with the column median values
preprocess = Pipeline([
    ('imputer_num', mdi.MeanMedianImputer(imputation_method='median',
                                          variables=['V1','V20'])),
      # feature Scaling
   ('scaler', StandardScaler())


])
```

In [410]:

```
preprocess.fit(X_train,y_train)
```

Out[410]:

```
Pipeline(memory=None,
         steps=[('imputer_num',
                 MeanMedianImputer(imputation_method='median',
                                   variables=['V1', 'V20'])),
                ('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=Tr
ue))],
         verbose=False)
```

In [411]:

```
X_train=preprocess.transform(X_train)
X_test=preprocess.transform(X_test)
X_testt=preprocess.transform(test)
```

In [218]:

```
# Basic Algorithm
```

In [219]:

```
#f2 score
from sklearn.metrics import fbeta_score, make_scorer
ftwo_scorer = make_scorer(fbeta_score, beta=2)
```

In [278]:

```
#logistic regression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

logreg=LogisticRegression(max_iter=10000)

logreg_param= {
     'C': [0.01,0.1,1],
      'penalty':['l2','l1'],
     'solver' :['newton-cg','lbfgs', 'sag'],
     'multi_class':['auto']
}
logreg_grid = GridSearchCV(logreg,logreg_param,cv=5,n_jobs=-1,scoring=ftwo_score
logreg_grid.fit(X_train,y_train)
print(f'Best Mean Cross Validation Score is {logreg_grid.best_score_}')
print(f'Best Mean Cross Validation Score is {logreg_grid.best_params_}')
print(f'Train score is {logreg_grid.score(X_train,y_train)}')
print(f'Test score is {logreg_grid.score(X_test,y_test)}')

```

```
Best Mean Cross Validation Score is 0.7911340852130325
Best Mean Cross Validation Score is {'C': 1, 'multi_class': 'auto', 'p
enalty': 'l2', 'solver': 'newton-cg'}
Train score is 0.913978494623656
Test score is 0.9615384615384615
```

In [265]:

```
#Decision Tree

from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from math import sqrt
from sklearn.tree import DecisionTreeClassifier

dtree = DecisionTreeClassifier(random_state=0)

param_dtree = {'max_depth': [1,2,3,5],
               'splitter':['random','best'],
               'max_features':['auto','log2','sqrt'],
               'criterion':['gini','entropy'],

              }


grid_dtree = GridSearchCV(dtree, param_dtree, cv=5,n_jobs=-1,scoring=ftwo_scorer
grid_dtree.fit(X_train, y_train)

print("Best Mean Cross-validation score: {:.3f}".format(grid_dtree.best_score_))
print('Decision Tree parameters: ', grid_dtree.best_params_)
print('Train score: ', grid_dtree.score(X_train, y_train))
print("Test Score:", grid_dtree.score(X_test,y_test))

X_test_preds = grid_dtree.predict(X_testt)



pd.DataFrame({'Id': test.Id, 'Target':X_test_preds }).to_csv('solution_base15.cs
print("Done :D")


```

```
Best Mean Cross-validation score: 0.745
Decision Tree parameters:  {'criterion': 'gini', 'max_depth': 2, 'max_
features': 'auto', 'splitter': 'best'}
Train score:  0.9042553191489363
Test Score: 1.0
Done :D
```

In [261]:

```
#knn
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()

param_knn = {
              'n_neighbors': [3],
              'weights': ['uniform','distance'],
              'algorithm' :['auto'],
              'p':[2,3]

            }

#apply grid search

grid_knn = GridSearchCV(knn, param_knn, cv=5,n_jobs=-1,scoring=ftwo_scorer)
grid_knn.fit(X_train, y_train)

# Mean Cross Validation Score
print("Best Mean Cross-validation score: {:.3f}".format(grid_knn.best_score_))
print('KNN parameters: ', grid_knn.best_params_)
print('Train score: ', grid_knn.score(X_train, y_train))
print("KNN Test Performance: ", grid_knn.score(X_test,y_test))


```

```
Best Mean Cross-validation score: 0.817
KNN parameters:  {'algorithm': 'auto', 'n_neighbors': 3, 'p': 2, 'weig
hts': 'uniform'}
Train score:  0.8152173913043478
KNN Test Performance:  1.0
```

In [263]:

```
#Simple SVM

import matplotlib.gridspec as gridspec
import itertools
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC, LinearSVC
from sklearn.model_selection import GridSearchCV

#svc = SVC()

svc = LinearSVC(max_iter=100000)
svc.fit(X_train, y_train)

#define a list of parameters
param_svc_kernel = {'C': [0.001,0.01,0.1,1],
                    'penalty':['l1','l2'],
                    'multi_class':['ovr','crammer_singer']

                   }

#apply grid search
grid_svc = GridSearchCV(svc,param_svc_kernel, cv=5,n_jobs=-1,scoring=ftwo_scorer
grid_svc.fit(X_train, y_train)

print('Best cross-validation score:', grid_svc.best_score_)
print(grid_svc.best_params_)
print('train score: ', grid_svc.score(X_train, y_train))
print('test score: ', grid_svc.score(X_test, y_test))



X_test_preds = grid_svc.predict(X_testt)
pd.DataFrame({'Id': test.Id, 'Target':X_test_preds }).to_csv('solution_base3.csv
print("Done :D")


```

```
Best cross-validation score: 0.906265664160401
{'C': 0.001, 'multi_class': 'crammer_singer', 'penalty': 'l1'}
train score:  0.9042553191489363
test score:  1.0
Done :D
```

In [312]:

```
#SVM with kernel='rbf'
import matplotlib.gridspec as gridspec
import itertools
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC, LinearSVC
from sklearn.model_selection import GridSearchCV
import numpy as np


svc = SVC()
svc.fit(X_train, y_train)
svc_kernel = SVC(kernel = 'rbf')

#define a list of parameters
C_range = 10. ** np.arange(-3, 8)
gamma_range = 10. ** np.arange(-5, 4)

param_svc_kernel = {'C': C_range,
                    'gamma':gamma_range}

#apply grid search
grid_svc_rbf = GridSearchCV(svc_kernel, param_svc_kernel, cv=5, n_jobs=-1,scorin
grid_svc_rbf.fit(X_train, y_train)
print('Best cross-validation score:', grid_svc_rbf.best_score_)
print(grid_svc_rbf.best_params_)
print('train score: ', grid_svc_rbf.score(X_train, y_train))
print('test score: ', grid_svc_rbf.score(X_test, y_test))


X_test_preds = grid_svc_rbf.predict(X_testt)
pd.DataFrame({'Id': test.Id, 'Target':X_test_preds }).to_csv('solution_base.csv
print("Done :D")



```

```
Best cross-validation score: 0.906265664160401
{'C': 10.0, 'gamma': 0.001}
train score:  0.913978494623656
test score:  1.0
Done :D
```

In [266]:

```
#Random Forest

from sklearn.ensemble import RandomForestClassifier
rfc =RandomForestClassifier(random_state=0)
rfc_param = {
     'n_estimators': [100],
     'max_features': ['auto','sqrt','log2'],
     'max_depth' : [1,2,3,5],
     'criterion' :['gini', 'entropy'],
     'min_samples_split' :[2],
     'min_samples_leaf':[3],

}
rfc_grid = GridSearchCV(rfc, rfc_param,cv=5,n_jobs=-1,scoring=ftwo_scorer)
rfc_grid.fit(X_train,y_train)
print(f'Best Mean Cross Validation Score is {rfc_grid.best_score_}')
print(f'Best param {rfc_grid.best_params_}')
print(f'Train score is {rfc_grid.score(X_train,y_train)}')
print(f'Test score is {rfc_grid.score(X_test,y_test)}')

```

```
Best Mean Cross Validation Score is 0.6660179861418251
Best param {'criterion': 'gini', 'max_depth': 2, 'max_features': 'aut
o', 'min_samples_leaf': 3, 'min_samples_split': 2, 'n_estimators': 10
0}
Train score is 0.8152173913043478
Test score is 0.8333333333333334
Done :D
```

In [226]:

```
#Extra trees
from  sklearn.ensemble import ExtraTreesClassifier
etc= ExtraTreesClassifier(random_state=0)
etc_param = {
    'n_estimators': [100],
    'max_features': ['auto','sqrt','log2'],
    'max_depth' : [1,2,3,5],
    'criterion' :['gini','entropy'],
    'min_samples_split':[2,3],


}
etc_grid = GridSearchCV(etc, etc_param,cv=5,n_jobs=-1,scoring=ftwo_scorer)
etc_grid.fit(X_train,y_train)

print(f'Best Mean Cross Validation Score is {etc_grid.best_score_}')
print(f'Best param {etc_grid.best_params_}')
print(f'Train score is {etc_grid.score(X_train,y_train)}')
print(f'Test score is {etc_grid.score(X_test,y_test)}')

```

```
Best Mean Cross Validation Score is 0.6255417956656346
Best param {'criterion': 'gini', 'max_depth': 5, 'max_features': 'aut
o', 'min_samples_split': 2, 'n_estimators': 100}
Train score is 0.8695652173913043
Test score is 0.6521739130434783
Done :D
```

In [227]:

```
#Gradient boost
from  sklearn.ensemble import GradientBoostingClassifier
gbc= GradientBoostingClassifier(random_state=0)
gbc_param = {
               'max_depth' : [1],
               'n_estimators' : [200],
               'learning_rate' : [0.1],
               'subsample':[1],
               'min_samples_split':[2],
               'min_samples_leaf':[4],
               'max_features': ['auto','sqrt','log2'],
               'loss': ['deviance','exponential'],


             }
gbc_grid = GridSearchCV(gbc, gbc_param,cv=5,n_jobs=-1,scoring=ftwo_scorer)
gbc_grid.fit(X_train,y_train)
print(f'Best Mean Cross Validation Score is {gbc_grid.best_score_}')
print(f'Best param {gbc_grid.best_params_}')
print(f'Train score is {gbc_grid.score(X_train,y_train)}')
print(f'Test score is {gbc_grid.score(X_test,y_test)}')


```

```
Best Mean Cross Validation Score is 0.8236842105263158
Best param {'learning_rate': 0.1, 'loss': 'exponential', 'max_depth':
1, 'max_features': 'auto', 'min_samples_leaf': 4, 'min_samples_split':
2, 'n_estimators': 200, 'subsample': 1}
Train score is 0.913978494623656
Test score is 0.9615384615384615
Done :D
```

In [268]:

```
#XGBoost

from xgboost import XGBClassifier
xgbc= XGBClassifier(random_state=0,early_stopping_rounds=4,objective= 'binary:lo
xgbc_param = {
              'max_depth' : [1,2,3,5],
              'n_estimators' : [150],
              'learning_rate' : [0.01,0.1,0.5],
              'min_child_weight' : [3],
             # 'subsample':[0.8]
             }
xgbc_grid = GridSearchCV(xgbc, xgbc_param,cv=5,n_jobs=-1,scoring=ftwo_scorer)
xgbc_grid.fit(X_train,y_train)
print(f'Best Mean Cross Validation Score is {xgbc_grid.best_score_}')
print(f'Best param {xgbc_grid.best_params_}')
print(f'Train score is {xgbc_grid.score(X_train,y_train)}')
print(f'Test score is {xgbc_grid.score(X_test,y_test)}')

```

```
Best Mean Cross Validation Score is 0.8094820384294069
Best param {'learning_rate': 0.01, 'max_depth': 1, 'min_child_weight':
3, 'n_estimators': 150}
Train score is 0.8241758241758242
Test score is 0.8333333333333334
Done :D
```

In [393]:

```
#Dummy Classifier

from sklearn.dummy import DummyClassifier

dummy= DummyClassifier()

dummy_param= {
   "strategy":['prior','uniform','stratified','most_frequent','constant']
}


dummy_grid = GridSearchCV(dummy, dummy_param,cv=5,n_jobs=-1,scoring=ftwo_scorer)
dummy_grid.fit(X_train,y_train)
print(f'Best Mean Cross Validation Score is {dummy_grid.best_score_}')
print(f'Best param {dummy_grid.best_params_}')
print(f'Train score is {dummy_grid.score(X_train,y_train)}')
print(f'Test score is {dummy_grid.score(X_test,y_test)}')

```

```
Best Mean Cross Validation Score is 0.06134344811097836
Best param {'strategy': 'uniform'}
Train score is 0.05241090146750525
Test score is 0.04705882352941176
```

In [229]:

```
#Cost Sensitive Algorithms
```

In [412]:

```
#logistic regression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold

logreg=LogisticRegression(max_iter=1000)

logreg_param= {
    'C': [0.01],
    'penalty':['l1', 'l2'],
    'class_weight' :['balanced','none'],
    'solver' :['lbfgs'],
    'max_iter' :[200],
    'multi_class':['auto'],
    'class_weight': [{0:1,1:10}]
}

cv = RepeatedStratifiedKFold(n_splits=7, n_repeats=3, random_state=1)

clogreg_grid = GridSearchCV(logreg, logreg_param,cv=cv,n_jobs=-1,scoring=ftwo_sc
clogreg_grid.fit(X_train,y_train)
print(f'Best Mean Cross Validation Score is {clogreg_grid.best_score_}')
print(f'Best param {clogreg_grid.best_params_}')
print(f'Train score is {clogreg_grid.score(X_train,y_train)}')
print(f'Test score is {clogreg_grid.score(X_test,y_test)}')



X_train_preds = clogreg_grid.predict(X_train)
X_test_preds = clogreg_grid.predict(X_test)

print('train rmse: {}'.format(sqrt(mean_squared_error(y_train, X_train_preds))))
print('test rmse: {}'.format(sqrt(mean_squared_error(y_test, X_test_preds))))


```

```
Best Mean Cross Validation Score is 0.6744916804901324
Best param {'C': 0.01, 'class_weight': {0: 1, 1: 10}, 'max_iter': 200,
'multi_class': 'auto', 'penalty': 'l2', 'solver': 'lbfgs'}
Train score is 0.7575757575757575
Test score is 0.9090909090909091
train rmse: 0.07955572841757301
test rmse: 0.08192319205190404
```

In [413]:

```
#Decision Tree

from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from math import sqrt
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=0)


#define a list of parameters
param_dtree = {'max_depth': [3],
               'class_weight': [{0:100,1:1}, {0:10,1:1}, {0:1,1:1}, {0:1,1:10},
                'splitter':['random','best'],
                'max_features':['auto','sqrt','log2']
               }

cgrid_dtree = GridSearchCV(dtree, param_dtree, cv=5,n_jobs=-1,scoring=ftwo_score
cgrid_dtree.fit(X_train, y_train)

print("Best Mean Cross-validation score: {:.3f}".format(cgrid_dtree.best_score_
print('Decision Tree parameters: ', cgrid_dtree.best_params_)
print('Train score: ', cgrid_dtree.score(X_train, y_train))
print("Test Score:", cgrid_dtree.score(X_test,y_test))

X_train_preds = cgrid_dtree.predict(X_train)
X_test_preds = cgrid_dtree.predict(X_test)

print('train rmse: {}'.format(sqrt(mean_squared_error(y_train, X_train_preds))))
print('test rmse: {}'.format(sqrt(mean_squared_error(y_test, X_test_preds))))

```

```
Best Mean Cross-validation score: 0.673
Decision Tree parameters:  {'class_weight': {0: 1, 1: 1}, 'max_depth':
3, 'max_features': 'log2', 'splitter': 'best'}
Train score:  0.7142857142857142
Test Score: 0.6382978723404255
train rmse: 0.0719608639321375
test rmse: 0.08192319205190404
```

In [414]:

```
#Simple SVM
import matplotlib.gridspec as gridspec
import itertools
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC, LinearSVC
from sklearn.model_selection import GridSearchCV

#svc = SVC()

svc = LinearSVC(max_iter=100000)
svc.fit(X_train, y_train)



#define a list of parameters
param_svc_kernel = {'C': [0.01],
                    'class_weight': [{0:100,1:1}, {0:10,1:1}, {0:1,1:1}, {0:1,1:
                    'penalty':['l1','l2'],
                    'multi_class':['ovr','crammer_singer']
                   }

#apply grid search

cgrid_svc = GridSearchCV(svc,param_svc_kernel, cv=5,n_jobs=-1,scoring=ftwo_score
cgrid_svc.fit(X_train, y_train)
print('Best cross-validation score:', cgrid_svc.best_score_)
print(cgrid_svc.best_params_)
print('train score: ', cgrid_svc.score(X_train, y_train))
print('test score: ', cgrid_svc.score(X_test, y_test))

X_train_preds = cgrid_svc.predict(X_train)
X_test_preds = cgrid_svc.predict(X_test)

print('train rmse: {}'.format(sqrt(mean_squared_error(y_train, X_train_preds))))
print('test rmse: {}'.format(sqrt(mean_squared_error(y_test, X_test_preds))))


```

```
Best cross-validation score: 0.6836496836496837
{'C': 0.01, 'class_weight': {0: 1, 1: 1}, 'multi_class': 'crammer_sing
er', 'penalty': 'l1'}
train score:  0.703125
test score:  0.980392156862745
train rmse: 0.07955572841757301
test rmse: 0.03663716527236559
```

In [415]:

```
#Random Forest
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

crf = RandomForestClassifier(n_estimators=150)
crf_param={

                'max_depth' : [2,3,5],
                'criterion' : ["gini","entropy"],
                'max_features': ["auto","sqrt","log2"],
                'class_weight':["balanced"]

}
crf_grid = GridSearchCV(crf, crf_param,cv=5,n_jobs=-1, scoring=ftwo_scorer)

crf_grid.fit(X_train,y_train)

print(f'Best Mean Cross Validation Score is {crf_grid.best_score_}')
print(f'Best param {crf_grid.best_params_}')
print(f'Train score is {crf_grid.score(X_train,y_train)}')
print(f'Test score is {crf_grid.score(X_test,y_test)}')

X_train_preds = crf_grid.predict(X_train)
X_test_preds = crf_grid.predict(X_test)

print('train rmse: {}'.format(sqrt(mean_squared_error(y_train, X_train_preds))))
print('test rmse: {}'.format(sqrt(mean_squared_error(y_test, X_test_preds))))


```

```
Best Mean Cross Validation Score is 0.6991216512955644
Best param {'class_weight': 'balanced', 'criterion': 'entropy', 'max_d
epth': 3, 'max_features': 'log2'}
Train score is 0.8076923076923078
Test score is 0.8823529411764706
train rmse: 0.06346351669793114
test rmse: 0.06345743169703524
```

In [416]:

```
#XGBoost
from xgboost import XGBClassifier
xgbc= XGBClassifier(random_state=0,early_stopping_rounds=2,objective= 'binary:lo

param_grid = {
 # 'scale_pos_weight': [1, 10],
              'max_depth' : [1,2,3,5],
              'n_estimators' : [150],
              'learning_rate' : [0.1],
              'min_child_weight' : [1,2],
              'subsample':[0.5],
              'class_weight': [{0:100,1:1}, {0:10,1:1}, {0:1,1:1}, {0:1,1:10}, 

}

cxgbc_grid = GridSearchCV(xgbc, param_grid,cv=10,n_jobs=-1, scoring=ftwo_scorer)
cxgbc_grid.fit(X_train,y_train)
print(f'Best Mean Cross Validation Score is {cxgbc_grid.best_score_}')
print(f'Best param {cxgbc_grid.best_params_}')
print(f'Train score is {cxgbc_grid.score(X_train,y_train)}')
print(f'Test score is {cxgbc_grid.score(X_test,y_test)}')


X_train_preds = cxgbc_grid.predict(X_train)
X_test_preds = cxgbc_grid.predict(X_test)

print('train rmse: {}'.format(sqrt(mean_squared_error(y_train, X_train_preds))))
print('test rmse: {}'.format(sqrt(mean_squared_error(y_test, X_test_preds))))


```

```
Best Mean Cross Validation Score is 0.6864468864468865
Best param {'class_weight': {0: 100, 1: 1}, 'learning_rate': 0.1, 'max
_depth': 1, 'min_child_weight': 2, 'n_estimators': 150, 'subsample':
0.5}
Train score is 0.7421875000000001
Test score is 0.9615384615384615
train rmse: 0.0719608639321375
test rmse: 0.05181277601508398
```

In [417]:

```
#Extratrees
from  sklearn.ensemble import ExtraTreesClassifier


etc= ExtraTreesClassifier(random_state=0,class_weight='balanced')
etc_param = {
    'n_estimators': [100],
    'max_features': ['auto','sqrt','log2'],
    'max_depth' : [1,2,3,5],
    'criterion' :['gini','entropy'],
    'min_samples_split':[2,3],

}
cetc_grid = GridSearchCV(etc, etc_param,cv=5,n_jobs=-1,scoring=ftwo_scorer)
cetc_grid.fit(X_train,y_train)

print(f'Best Mean Cross Validation Score is {cetc_grid.best_score_}')
print(f'Best param {cetc_grid.best_params_}')
print(f'Train score is {cetc_grid.score(X_train,y_train)}')
print(f'Test score is {cetc_grid.score(X_test,y_test)}')

X_train_preds = cetc_grid.predict(X_train)
X_test_preds = cetc_grid.predict(X_test)

print('train rmse: {}'.format(sqrt(mean_squared_error(y_train, X_train_preds))))
print('test rmse: {}'.format(sqrt(mean_squared_error(y_test, X_test_preds))))


```

```
Best Mean Cross Validation Score is 0.7160528682267813
Best param {'criterion': 'gini', 'max_depth': 3, 'max_features': 'aut
o', 'min_samples_split': 2, 'n_estimators': 100}
Train score is 0.7835820895522388
Test score is 0.9433962264150945
train rmse: 0.07955572841757301
test rmse: 0.06345743169703524
```

In [307]:

```
#Bagging Decision Tree with undersampling
from imblearn.ensemble import BalancedBaggingClassifier
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score

b_dtree = BalancedBaggingClassifier()
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(b_dtree, X_train, y_train, scoring=ftwo_scorer, cv=cv,
print('Mean f2: %.3f' % scores.mean())

```

```
Mean f2: 0.691
```

In [423]:

```
scores1= cross_val_score(etc, X_train, y_train, scoring=ftwo_scorer, cv=5, n_jo
scores2= cross_val_score(b_dtree, X_train, y_train, scoring=ftwo_scorer, cv=5, 
scores3= cross_val_score(xgbc, X_train, y_train, scoring=ftwo_scorer, cv=5, n_j
scores4= cross_val_score(crf, X_train, y_train, scoring=ftwo_scorer, cv=5, n_jo
scores5= cross_val_score(svc, X_train, y_train, scoring=ftwo_scorer, cv=5, n_jo
scores6= cross_val_score(dtree, X_train, y_train, scoring=ftwo_scorer, cv=5, n_
scores7= cross_val_score(logreg, X_train, y_train, scoring=ftwo_scorer, cv=5, n_
print('mean scores of all cost sensitive algorithms')
print(f'etc {scores1.mean()}')
print(f'b_tree {scores2.mean()}')
print(f'xgbc {scores3.mean()}')
print(f'randomforest {scores4.mean()}')
print(f'svc {scores5.mean()}')
print(f'dtree {scores6.mean()}')
print(f'logreg {scores7.mean()}')

```

```
mean scores of all cost sensitive algorithms
etc 0.563338959890684
b_tree 0.5525844184139883
xgbc 0.637052685778323
randomforest 0.6914293436032567
svc 0.6556433904259992
dtree 0.6619252432155658
logreg 0.6223100570926658
```

In [237]:

```
#3. Data Sampling Algorithms
```

In [428]:

```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as p

pipe_roc_lg = p([('smote',SMOTE()),('lg',LogisticRegression())])
param_roc_lg = {'smote__k_neighbors': [2],
                'lg__C': [0.0001],
                 'lg__penalty':['l1', 'l2'],
                'lg__max_iter':[150],
                'lg__solver':['newton-cg'],
                 'lg__class_weight':["balanced"],
                'lg__multi_class':['auto', 'ovr', 'multinomial']
                }

ogrid_lg= GridSearchCV(pipe_roc_lg,param_roc_lg, cv=5, n_jobs=-1, scoring=ftwo_s
ogrid_lg.fit(X_train, y_train)

print("Best Mean cross-validation score: {:.3f}".format(ogrid_lg.best_score_))
print("Best parameters: {}".format(ogrid_lg.best_params_))
print(f'Train score is {ogrid_lg.score(X_train,y_train)}')
print(f'Test score is {ogrid_lg.score(X_test,y_test)}')


# let's get the predictions
X_test_preds = ogrid_lg.predict(X_testt)



pd.DataFrame({'Id': test.Id, 'Target':X_test_preds }).to_csv('solution_base1.csv
print("Done :D")


```

```
Best Mean cross-validation score: 0.699
Best parameters: {'lg__C': 0.0001, 'lg__class_weight': 'balanced', 'lg
__max_iter': 150, 'lg__multi_class': 'multinomial', 'lg__penalty': 'l
2', 'lg__solver': 'newton-cg', 'smote__k_neighbors': 2}
Train score is 0.7835820895522388
Test score is 0.9259259259259259
Done :D
```

In [343]:

```
#Decision Tree
pipe_roc_dtree = p([('smote',SMOTE()),('dtree',DecisionTreeClassifier(random_sta
param_roc_dtree = {'smote__k_neighbors': [2,3,5],
                   'dtree__max_depth': [2,3,5],
                'dtree__splitter':['random','best'],
                'dtree__max_features':['auto','log2','sqrt'],
                'dtree__criterion':['gini','entropy'],
            }

ogrid_dtree= GridSearchCV(pipe_roc_dtree,param_roc_dtree, cv=5, n_jobs=-1, scori
ogrid_dtree.fit(X_train, y_train)

print("Best Mean cross-validation score: {:.3f}".format(ogrid_dtree.best_score_
print("Best parameters: {}".format(ogrid_dtree.best_params_))
print(f'Train score is {ogrid_dtree.score(X_train,y_train)}')
print(f'Test score is {ogrid_dtree.score(X_test,y_test)}')

# let's get the predictions
X_test_preds = ogrid_dtree.predict(X_testt)



pd.DataFrame({'Id': test.Id, 'Target':X_test_preds }).to_csv('solution_base111.c
print("Done :D")

```

```
Best Mean cross-validation score: 0.868
Best parameters: {'dtree__criterion': 'entropy', 'dtree__max_depth':
3, 'dtree__max_features': 'log2', 'dtree__splitter': 'best', 'smote__k
_neighbors': 5}
Train score is 0.9042553191489363
Test score is 0.8928571428571429
Done :D
```

In [398]:

```
#KNN

pipe_roc_knn = p([('smote',SMOTE()),('knn',KNeighborsClassifier())])
param_roc_knn = {'smote__k_neighbors': [2,3,5],
                 'knn__weights':['uniform'],
                 'knn__algorithm' :['kd_tree', 'brute'],
                 'knn__p':[3]
                }

ogrid_knn= GridSearchCV(pipe_roc_knn,param_roc_knn, cv=5, n_jobs=-1, scoring=ftw
ogrid_knn.fit(X_train, y_train)

print("Best Mean cross-validation score: {:.3f}".format(ogrid_knn.best_score_))
print("Best parameters: {}".format(ogrid_knn.best_params_))
print(f'Train score is {ogrid_knn.score(X_train,y_train)}')
print(f'Test score is {ogrid_knn.score(X_test,y_test)}')


```

```
Best Mean cross-validation score: 0.873
Best parameters: {'knn__algorithm': 'kd_tree', 'knn__p': 3, 'knn__weig
hts': 'uniform', 'smote__k_neighbors': 3}
Train score is 0.9693877551020408
Test score is 0.8928571428571429
```

In [241]:

```
#SVM rbf
from sklearn import svm

from sklearn.svm import SVC, LinearSVC

pipe_roc_svm = p([('smote',SMOTE()),('svm',svm.SVC(kernel='rbf'))])
param_roc_svm = {'smote__k_neighbors': [2,3,5],
                 'svm__C':[0.001,0.01,0.1],
                 'svm__gamma':['scale','auto'],
                 }

ogrid_svm_rbf= GridSearchCV(pipe_roc_svm,param_roc_svm, cv=5, n_jobs=-1, scorin
ogrid_svm_rbf.fit(X_train, y_train)

print("Best Mean cross-validation score: {:.3f}".format(ogrid_svm_rbf.best_score
print("Best parameters: {}".format(ogrid_svm_rbf.best_params_))
print(f'Train score is {ogrid_svm_rbf.score(X_train,y_train)}')
print(f'Test score is {ogrid_svm_rbf.score(X_test,y_test)}')
```

```
Best Mean cross-validation score: 0.874
Best parameters: {'smote__k_neighbors': 2, 'svm__C': 0.1, 'svm__gamm
a': 'scale'}
Train score is 0.913978494623656
Test score is 1.0
```

In [347]:

```
#Random Forest
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE

pipe_roc_rf = p([('smote',SMOTE()),('rf',RandomForestClassifier(random_state=0))
param_roc_rf = {

                 'smote__k_neighbors': [2],
                'rf__n_estimators' : [150],
                 'rf__max_depth' : [1],
                'rf__criterion' : ["gini","entropy"],
                'rf__min_samples_split' :[1,2],
                'rf__max_features': ["auto","sqrt","log2"]

               }

ogrid_rf= GridSearchCV(pipe_roc_rf,param_roc_rf, cv=7, n_jobs=-1, scoring=ftwo_s
ogrid_rf.fit(X_train, y_train)

print("Best Mean cross-validation score: {:.3f}".format(ogrid_rf.best_score_))
print("Best parameters: {}".format(ogrid_rf.best_params_))
print(f'Train score is {ogrid_rf.score(X_train,y_train)}')
print(f'Test score is {ogrid_rf.score(X_test,y_test)}')


# let's get the predictions
X_test_preds = ogrid_rf.predict(X_testt)



pd.DataFrame({'Id': test.Id, 'Target':X_test_preds }).to_csv('solution_base222.c
print("Done :D")


```

```
Best Mean cross-validation score: 0.862
Best parameters: {'rf__criterion': 'gini', 'rf__max_depth': 1, 'rf__ma
x_features': 'log2', 'rf__min_samples_split': 2, 'rf__n_estimators': 1
50, 'smote__k_neighbors': 2}
Train score is 0.8602150537634409
Test score is 1.0
Done :D
```

In [352]:

```
#easy ensemble classifier
from imblearn.ensemble import EasyEnsembleClassifier
pipe_roc_ee = p([('smote',SMOTE()),('ee',EasyEnsembleClassifier(random_state=0)
param_roc_ee = {'smote__k_neighbors': [1],
                  'ee__n_estimators':[50]
                }

ogrid_ee= GridSearchCV(pipe_roc_ee,param_roc_ee, cv=5, n_jobs=-1, scoring=ftwo_s
ogrid_ee.fit(X_train, y_train)

print("Best Mean cross-validation score: {:.3f}".format(ogrid_ee.best_score_))
print("Best parameters: {}".format(ogrid_ee.best_params_))
print(f'Train score is {ogrid_ee.score(X_train,y_train)}')
print(f'Test score is {ogrid_ee.score(X_test,y_test)}')

```

```
Best Mean cross-validation score: 0.841
Best parameters: {'ee__n_estimators': 50, 'smote__k_neighbors': 1}
Train score is 1.0
Test score is 1.0
```

In [309]:

```
#XGBoost
from  xgboost import XGBClassifier

pipe_roc_xgb = p([('smote',SMOTE()),('xgb',XGBClassifier(random_state=0,
                                                         earl
                                                         n_es


                                                          ))
param_roc_xgb = {
                  'smote__k_neighbors': [1,2,3,5],
                  'xgb__learning_rate' : [0.01],
                  'xgb__max_depth' : [2,3,5]
}

ogrid_xgb= GridSearchCV(pipe_roc_xgb,param_roc_xgb, cv=5, n_jobs=-1, scoring=ftw
ogrid_xgb.fit(X_train, y_train)

print("Best Mean cross-validation score: {:.3f}".format(ogrid_xgb.best_score_))
print("Best parameters: {}".format(ogrid_xgb.best_params_))

print(f'Train score is {ogrid_xgb.score(X_train,y_train)}')
print(f'Test score is {ogrid_xgb.score(X_test,y_test)}')

```

```
Best Mean cross-validation score: 0.821
Best parameters: {'smote__k_neighbors': 3, 'xgb__learning_rate': 0.01,
'xgb__max_depth': 3}
Train score is 0.9693877551020408
Test score is 0.9259259259259259
Done :D
```

In [251]:

```
#Neural Network(scikit learn MLPClassifier) with RandomizedSearchCV
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV


nn = MLPClassifier(random_state=0)

param_nn = {
               'activation': ['identity', 'logistic', 'tanh', 'relu'],
               'solver':['adam','lbfgs','sgd'],
               'alpha':[0.001,0.01],
               'max_iter':[10000],
              # 'learning_rate':['constant', 'invscaling', 'adaptive']
             }


#apply grid search
grid_nn = RandomizedSearchCV(nn, param_nn, cv=5, n_jobs=-1, scoring=ftwo_scorer)
grid_nn.fit(X_train, y_train)

# Mean Cross Validation Score
print("Best Mean Cross-validation score: {:.3f}".format(grid_nn.best_score_))
print('parameters: ', grid_nn.best_params_)
print('Train score: ', grid_nn.score(X_train, y_train))
print("Test score: ", grid_nn.score(X_test,y_test))


# let's get the predictions
X_test_preds = grid_nn.predict(X_testt)



pd.DataFrame({'Id': test.Id, 'Target':X_test_preds }).to_csv('solution_base5.csv
print("Done :D")
```

```
Best Mean Cross-validation score: 0.864
parameters:  {'solver': 'adam', 'max_iter': 10000, 'alpha': 0.001, 'ac
tivation': 'logistic'}
Train score:  0.913978494623656
Test score:  1.0
Done :D
```

In [382]:

```
#Stacking 1
#Stacking of Data Sampling Algorithms(randomforest, xgboost,easyensemble) with 

from sklearn.ensemble import StackingClassifier
sclf1 = StackingClassifier(estimators=
                                [('randomforest', ogrid_rf.best_estimator_),
                                 ('xgboost', ogrid_xgb.best_estimator_),
                                 ('easyemsemble', ogrid_ee.best_estimator_),
                                ],  final_estimator=LogisticRegression())
sclf1_param = {
                    'stack_method':['auto', 'predict_proba']

}
sclf1_grid = GridSearchCV(sclf1, sclf1_param,cv=5, n_jobs=-1, scoring=ftwo_score
sclf1_grid.fit(X_train,y_train)
print(f'Best Mean Cross Validation Score is {sclf1_grid.best_score_}')
print(f'Best param {sclf1_grid.best_params_}')
print(f'Train score is {sclf1_grid.score(X_train,y_train)}')
print(f'Test score is {sclf1_grid.score(X_test,y_test)}')
```

```
Best Mean Cross Validation Score is 0.6023391812865496
Best param {'stack_method': 'auto'}
Train score is 0.8695652173913043
Test score is 1.0
```

In [378]:

```
#Stacking 2
#Stacking of cost sensitive algorithms(logistic reg, decision tree, simple svm)

from sklearn.ensemble import StackingClassifier
sclf2 = StackingClassifier(estimators=
                                [('logreg', clogreg_grid.best_estimator_),
                                 ('dtree', cgrid_dtree.best_estimator_),
                                 ('simplesvm', cgrid_svc.best_estimator_),
                                ],  final_estimator=XGBClassifier())
sclf2_param = {
                'logreg__C':[0.01],
                'dtree__max_depth': [1,2,3],
                'simplesvm__penalty':['l1','l2'],

}
sclf2_grid = GridSearchCV(sclf2, sclf2_param,cv=5, n_jobs=-1, scoring=ftwo_score
sclf2_grid.fit(X_train,y_train)
print(f'Best Mean Cross Validation Score is {sclf2_grid.best_score_}')
print(f'Best param {sclf2_grid.best_params_}')
print(f'Train score is {sclf2_grid.score(X_train,y_train)}')
print(f'Test score is {sclf2_grid.score(X_test,y_test)}')


```

```
Best Mean Cross Validation Score is 0.906265664160401
Best param {'dtree__max_depth': 1, 'logreg__C': 0.01, 'simplesvm__pena
lty': 'l1'}
Train score is 0.9042553191489363
Test score is 1.0
```

In [389]:

```
#stacking 3
#stacking of basic Algorithms(knn, gradient boost and svm) with XGB as my final estimat



from sklearn.ensemble import StackingClassifier
sclf3 = StackingClassifier(estimators=
                           [('knn', grid_knn.best_estimator_),
                            ('gbc', gbc_grid.best_estimator_),
                            ('dtree', grid_dtree.best_estimator_),
                           ],  final_estimator=XGBClassifier(random_state=42,early_s
sclf3_param = {
        #  'final_estimator__C' : [0.01,0.1],
        #  'knn__n_neighbors': [2,3,5],
        #  'gbc__learning_rate' : [0.01,0.1]
                'stack_method':['predict_proba']


sclf3_grid = GridSearchCV(sclf3, sclf3_param,cv=5, n_jobs=-1, scoring=ftwo_scorer)
sclf3_grid.fit(X_train,y_train)
print(f'Best Mean Cross Validation Score is {sclf3_grid.best_score_}')
print(f'Best param {sclf3_grid.best_params_}')
print(f'Train score is {sclf3_grid.score(X_train,y_train)}')
print(f'Test score is {sclf3_grid.score(X_test,y_test)}')
```

```
Best Mean Cross Validation Score is 0.8236842105263158
Best param {'stack_method': 'predict_proba'}
Train score is 0.9895833333333334
Test score is 1.0
```

In [356]:

```
#Saving and uploading model using pickle for logistic regression model

import pickle

#logistic regression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

logreg=LogisticRegression(max_iter=1000)
logreg_param= {
    'C': [0.001,0.01,0.1,10,100],
    'penalty':['l1', 'l2']
}
logreg_grid = GridSearchCV(logreg,logreg_param,cv=5,n_jobs=-1,scoring=ftwo_score
logreg_grid.fit(X_train,y_train)

model = pickle.dumps(logreg_grid)
model_from_pickle = pickle.loads(model)
trainscore=model_from_pickle.score(X_train,y_train)

testscore=model_from_pickle.score(X_test,y_test)
print (f'train score {trainscore}')

```

```
train score 0.8695652173913043
```

In [255]:

```
#GaussianProcessClassifier
from sklearn.gaussian_process import GaussianProcessClassifier

gpc=GaussianProcessClassifier(random_state=0)

gpc_param= { 'n_restarts_optimizer':[2,3,4],
             'max_iter_predict':[150],
              'warm_start':[True]
}
#isoforest_grid = RandomizedSearchCV(isoforest, isoforest_param,cv=5,n_jobs=-1,s

gpc_grid = GridSearchCV(gpc, gpc_param,cv=5,n_jobs=-1,scoring=ftwo_scorer)
gpc_grid.fit(X_train,y_train)
print(f'Best Mean Cross Validation Score is {gpc_grid.best_score_}')
print(f'Best param {gpc_grid.best_params_}')
print(f'Train score is {gpc_grid.score(X_train,y_train)}')
print(f'Test score is {gpc_grid.score(X_test,y_test)}')
```

```
Best Mean Cross Validation Score is 0.8048767752715122
Best param {'max_iter_predict': 150, 'n_restarts_optimizer': 2, 'warm_
start': True}
Train score is 1.0
Test score is 0.9615384615384615
```

In [ ]:

```

```