Project 2

May 11, 2020

1 Final Results

- Best Model based on Kaggle Public Scoreboard (0.83182) - Cost Sensitive - XGBoost
- Best parameters: {'class_weight': 'balanced', 'learning_rate': 0.01, 'n_estimators': 1140, 'scale_pos_weight': 1150}
- Best Mean cross-validation score: 0.86
- Train score is 0.9926769731489015

- Best Model based on CV scores in Stacking - Combination 3
- Best Mean Cross Validation Score is 0.9978266297321057
- Best Parameters are {'stack_method': 'auto'}
- Train score is 0.9535655058043118

- Note: scores depicted here on heading next to model names are Kaggle Public Scores

2 EDA

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
pd.pandas.set_option('display.max_columns', None)
%matplotlib inline

# for the model
from sklearn.model_selection import train_test_split
```

[2]:
```
data =  pd.read_csv(r'C:\Users\nabhs\OneDrive\BUAN - Semester 2\BUAN 6341 -␣
↪Applied Machine Learning\Project 2\mis6341-project2\train.csv')
test= pd.read_csv(r'C:\Users\nabhs\OneDrive\BUAN - Semester 2\BUAN 6341 -␣
↪Applied Machine Learning\Project 2\mis6341-project2\test.csv')
```

[3]:
```
pd.set_option('display.max_columns', 999)
data.head()
```

[3]:

	Id	V1	V2	V3	V4	V5	V6	\
0	138662	-0.711273	1.272483	1.681631	0.039897	-0.312381	-1.161083	

```
1  235999  1.990679 -0.128465 -1.700556  0.529839  0.136889 -1.404270
2  245376  1.812653 -0.476162 -0.338988  1.386750 -0.745965 -0.449870
3  202483 -1.014219  0.522775 -0.337978 -1.957797  3.578395  3.266965
4    9710 -0.967270 -0.053815  2.273463 -0.894434  0.728924  0.153524

         V7        V8        V9       V10       V11       V12       V13  \
0  0.853875 -0.173979 -0.438558 -0.053805  0.321036  0.814690  1.226450
1  0.593353 -0.456700  0.648260 -0.071353 -0.982339  0.229658 -0.592959
2 -0.492226  0.029904  1.129394  0.227346 -1.197477 -0.555129 -1.369130
3  0.602857  0.644645 -0.305879 -0.417906  0.010333 -0.299630 -0.379164
4 -0.038330  0.165252  1.172526 -0.879114  2.505831 -2.249565  0.791677

        V14       V15       V16       V17       V18       V19       V20  \
0 -0.152389  0.687480 -0.030825 -0.284477 -0.696867 -0.227419  0.298008
1  0.574159 -0.263545 -0.780886 -0.066039 -0.575168  0.227544 -0.205092
2  0.320282  1.002745  0.501765 -0.670053  0.478236 -0.820942 -0.202894
3  0.329193  0.957904 -0.417285 -0.705683 -0.063235 -0.381035  0.136061
4  1.546279 -0.133370  0.184501  0.307080 -0.081722 -1.359941  0.007512

        V21       V22       V23       V24       V25       V26       V27  \
0 -0.208181 -0.414631  0.057957  0.930321 -0.165598  0.038799  0.384295
1  0.018179  0.179084 -0.004248  0.039531  0.333765 -0.225201 -0.047285
2  0.252668  0.655719  0.090057 -0.136884 -0.172452 -0.556921  0.036490
3  0.147742  0.590740 -0.680551  0.745346  1.119496  0.013520  0.004530
4  0.106692  0.514430 -0.081546 -0.305042 -0.354883  0.896254 -0.182293

        V28    V29  Target
0  0.192762  21.87       0
1 -0.059682  49.99       0
2 -0.021575  79.00       0
3 -0.132643   6.00       0
4 -0.142810  15.95       0
```

[4]:
```
data.shape
```

[4]:
```
(24846, 31)
```

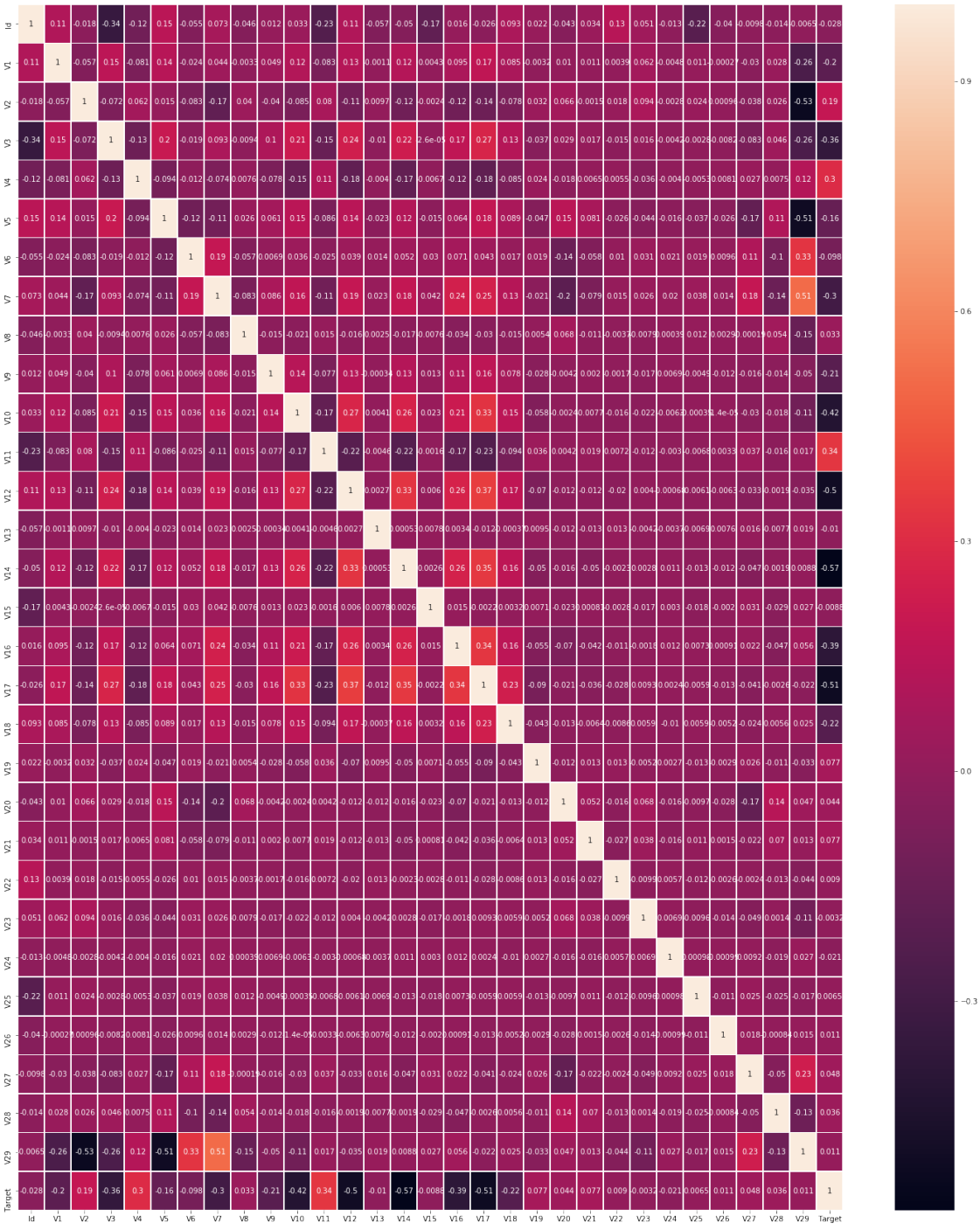
[5]:
```
test.shape
```

[5]:
```
(24846, 30)
```

2.1 Correlation Matrix

[134]:
```
fig, ax = plt.subplots(figsize=(25,30))         # Sample figsize in inches
sns.heatmap(data.corr(), annot=True, linewidths=.5, ax=ax)
```

[134]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x25759d6af08>
```


Id V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20 V21 V22 V23 V24 V25 V26 V27 V28 V29 Target
0.9
0.6
0.3
0.0
−0.3

2.2 Split in Test/Train

[6]:
```
X_train = data.drop(['Target','Id'], axis=1)
y_train = data['Target']

X_train.shape, y_train.shape
```

[6]:
```
((24846, 29), (24846,))
```

[7]:
```
X_test=test.drop(['Id'], axis=1)
X_test.shape
```

[7]:
```
(24846, 29)
```

2.3 Check Explanatory Variable Types

[8]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24846 entries, 0 to 24845
Data columns (total 31 columns):
Id        24846 non-null int64
V1        22459 non-null float64
V2        24846 non-null float64
V3        24846 non-null float64
V4        24846 non-null float64
V5        24846 non-null float64
V6        24846 non-null float64
V7        24846 non-null float64
V8        24846 non-null float64
V9        24846 non-null float64
V10       24846 non-null float64
V11       24846 non-null float64
V12       24846 non-null float64
V13       24846 non-null float64
V14       24846 non-null float64
V15       24846 non-null float64
V16       24846 non-null float64
V17       24846 non-null float64
V18       24846 non-null float64
V19       24846 non-null float64
V20       22317 non-null float64
V21       24846 non-null float64
V22       24846 non-null float64
V23       24846 non-null float64
V24       24846 non-null float64
V25       24846 non-null float64
```

```
V26       24846 non-null float64
V27       24846 non-null float64
V28       24846 non-null float64
V29       24846 non-null float64
Target    24846 non-null int64
dtypes: float64(29), int64(2)
memory usage: 5.9 MB
```

2.4 Categorize Variables acc. to Types

[9]:
```
# make list of variables  types

# numerical: discrete vs continuous
discrete = [var for var in data.columns if data[var].dtype!='O' and var!
 ↪='Target' and data[var].nunique()<10]
continuous = [var for var in data.columns if data[var].dtype!='O' and var!
 ↪='Target' and var not in discrete]

# categorical
categorical = [var for var in data.columns if data[var].dtype=='O']

print(f'There are {len(discrete)} discrete variables')
print(f'There are {len(continuous)} continuous variables')
print(f'There are {len(categorical)} categorical variables')
```

```
There are 0 discrete variables
There are 30 continuous variables
There are 0 categorical variables
```

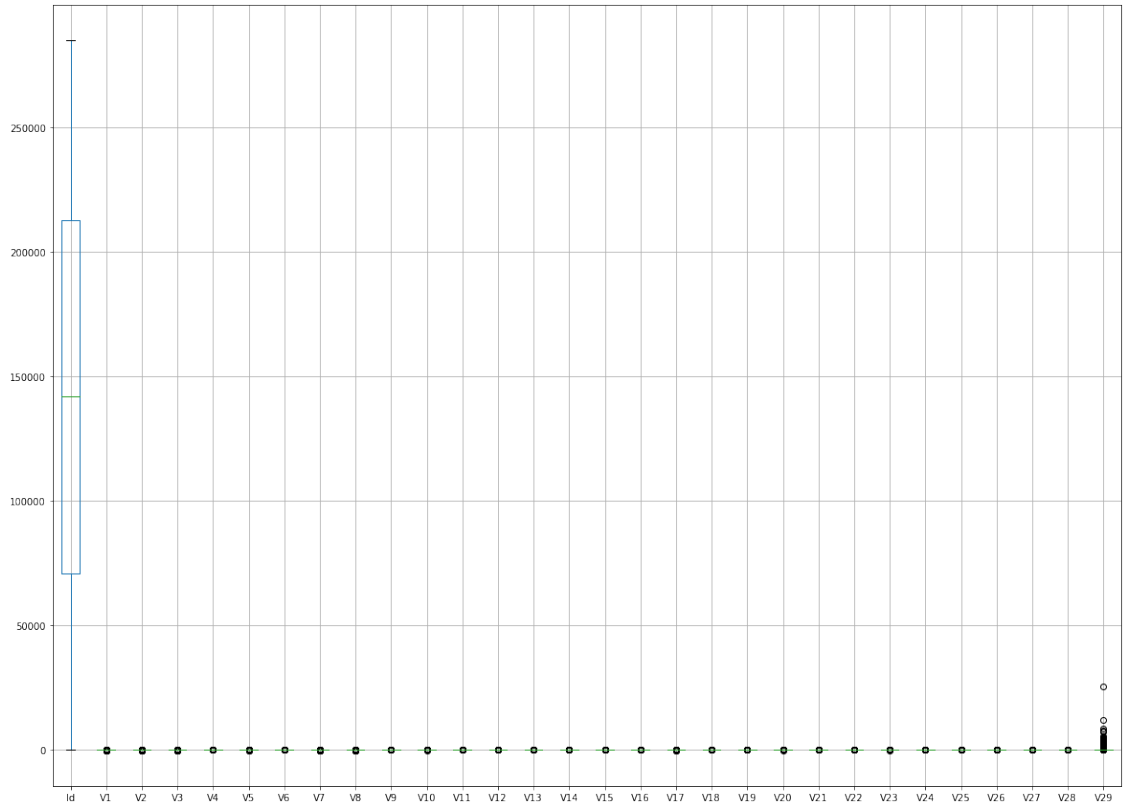
[10]:
```
len(data.columns)
```

[10]:
```
31
```

2.5 Outliers in Continuous

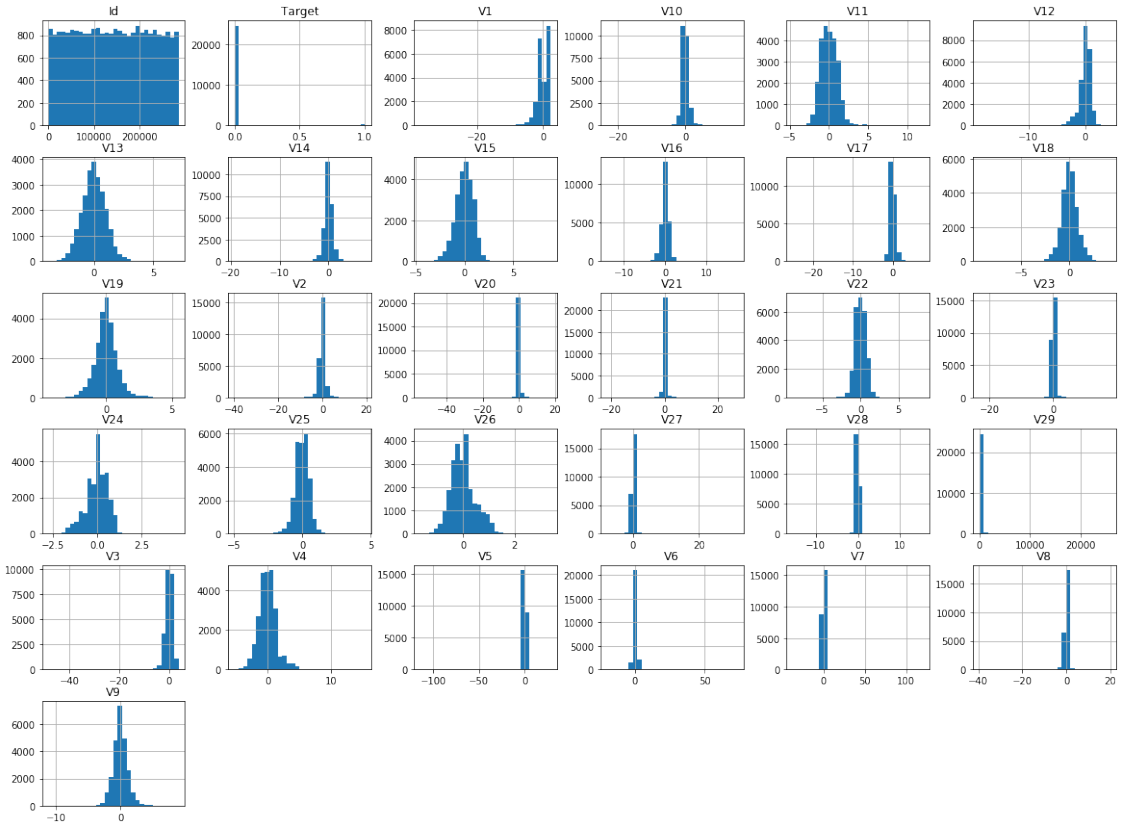
[11]:
```
data[continuous].boxplot(figsize=(20,15))
```

[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2573d6dbbc8>
```



2.6 Check for Normality

[12]:
```
# numerical
data.select_dtypes(exclude='O').hist(bins=30, figsize=(20,15))
plt.show()
```



2.7 Check for Null Values

[13]:
```
X_train.isnull().mean()
```

[13]:
```
V1      0.096072
V2      0.000000
V3      0.000000
V4      0.000000
V5      0.000000
V6      0.000000
V7      0.000000
V8      0.000000
V9      0.000000
V10     0.000000
V11     0.000000
V12     0.000000
V13     0.000000
V14     0.000000
V15     0.000000
V16     0.000000
V17     0.000000
```

```
V18    0.000000
V19    0.000000
V20    0.101787
V21    0.000000
V22    0.000000
V23    0.000000
V24    0.000000
V25    0.000000
V26    0.000000
V27    0.000000
V28    0.000000
V29    0.000000
dtype: float64
```

2.8 Pre-Processing

[14]:
```
# import relevant modules for feature engineering
from sklearn.pipeline import Pipeline
from feature_engine import missing_data_imputers as mdi
from feature_engine import categorical_encoders as ce
from sklearn.preprocessing import StandardScaler
```

[15]:
```
continuous=list(set(list(continuous))-set('Id'))
```

[16]:
```
continuous.remove('Id')
```

[17]:
```
continuous
```

[17]:
```
['V6',
 'V2',
 'V13',
 'V17',
 'V21',
 'V11',
 'V20',
 'V27',
 'V8',
 'V22',
 'V24',
 'V26',
 'V10',
 'V12',
 'V16',
 'V1',
 'V7',
 'V25',
 'V15',
```

```
 'V9',
 'V19',
 'V4',
 'V23',
 'V29',
 'V18',
 'V3',
 'V5',
 'V14',
 'V28']
```

[18]:
```
project2_preprocess = Pipeline([
    ('imputer_num', mdi.MeanMedianImputer(imputation_method='median',
                                          variables=continuous)),
  # feature Scaling
    ('scaler', StandardScaler())
])
```

[19]:
```
project2_preprocess.fit(X_train,y_train)
```

[19]:
```
Pipeline(memory=None,
         steps=[('imputer_num',
                 MeanMedianImputer(imputation_method='median',
                                   variables=['V6', 'V2', 'V13', 'V17', 'V21',
                                              'V11', 'V20', 'V27', 'V8', 'V22',
                                              'V24', 'V26', 'V10', 'V12', 'V16',
                                              'V1', 'V7', 'V25', 'V15', 'V9',
                                              'V19', 'V4', 'V23', 'V29', 'V18',
                                              'V3', 'V5', 'V14', 'V28'])),
                ('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True))],
         verbose=False)
```

[20]:
```
# Apply Transformations
X_train=project2_preprocess.transform(X_train)
X_test=project2_preprocess.transform(X_test)
```

3 DO NOT CHANGE STEPS BEFORE THIS POINT

3.1 Apply Basic Models

[21]:
```
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
```

```
from sklearn.dummy import DummyClassifier
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc
from sklearn.metrics import make_scorer
from matplotlib import pyplot
from sklearn.model_selection import GridSearchCV
```

[22]:
```
from sklearn.model_selection import cross_val_score
from imblearn.metrics import geometric_mean_score
from sklearn.metrics import make_scorer, fbeta_score
```

[23]:
```
f2score = make_scorer(fbeta_score,beta=2)
```

3.1.1 Logistic Regression - .80909

[24]:
```
from sklearn.linear_model import LogisticRegression

param_logit = {'C': [0.001, 0.01, 0.1, 1, 10,1000],
               'solver':['lbfgs'],
              'penalty':['l2'],
              'max_iter':range(1000,1500,10)}
print("Parameter grid:\n{}".format(param_logit))

grid_logit = GridSearchCV(LogisticRegression(random_state=42), param_logit,␣
 ↪cv=5, return_train_score=True,scoring = f2score)
grid_logit.fit(X_train, y_train)
```

```
Parameter grid:
{'C': [0.001, 0.01, 0.1, 1, 10, 1000], 'solver': ['lbfgs'], 'penalty': ['l2'],
'max_iter': range(1000, 1500, 10)}
```

[24]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='auto',
                                          n_jobs=None, penalty='l2',
                                          random_state=42, solver='lbfgs',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 1000],
                         'max_iter': range(1000, 1500, 10), 'penalty': ['l2'],
                         'solver': ['lbfgs']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[333]:
```
print("Best parameters: {}".format(grid_logit.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_logit.best_score_))
print('Train score: {:.4f}'.format(grid_logit.score(X_train, y_train)))
```

```
Best parameters: {'C': 10, 'max_iter': 1000, 'penalty': 'l2', 'solver': 'lbfgs'}
Best cross-validation score: 0.80
Train score: 0.8177
```

3.1.2 Decision Tree

[26]:
```
from sklearn.tree import DecisionTreeClassifier
```

[27]:
```
dtree = DecisionTreeClassifier(random_state=42)

#define a list of parameters
param_dtree = {'max_depth': range(1,20)}

#apply grid search
grid_dtree = GridSearchCV(dtree, param_dtree, cv=5, return_train_score =␣
 ↪True,scoring = f2score)
grid_dtree.fit(X_train, y_train)
```

[27]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=42,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': range(1, 20)}, pre_dispatch='2*n_jobs',
             refit=True, return_train_score=True,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

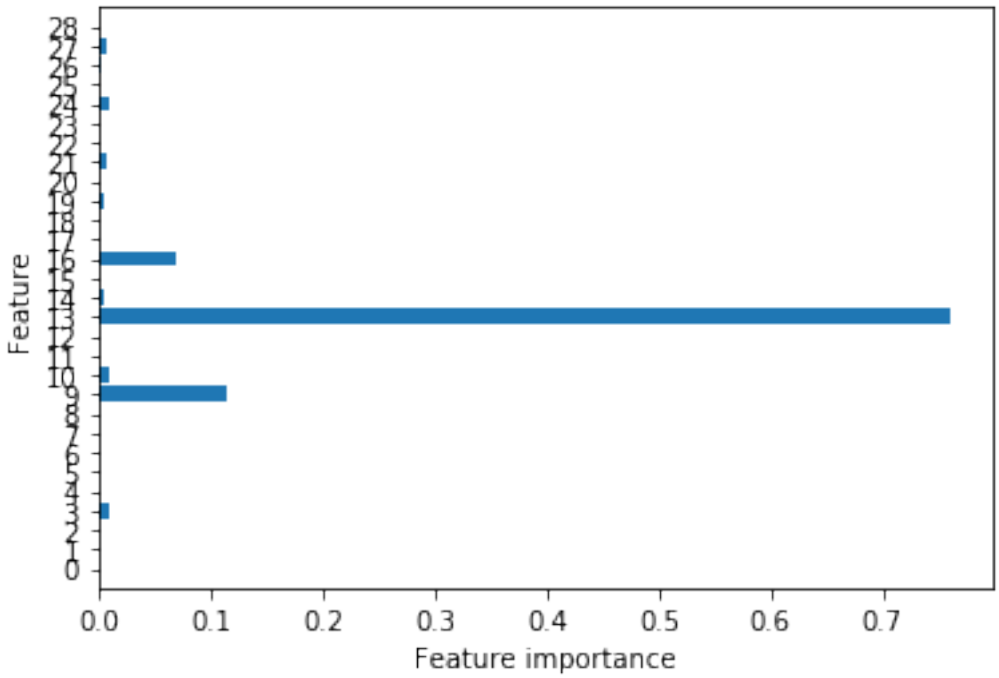
[335]:
```
#find best parameters
print('Decision Tree parameters: ', grid_dtree.best_params_)
# Mean Cross Validation Score
print("Best Mean Cross-validation score: {:.2f}".format(grid_dtree.best_score_))
```

```
print('Decision Tree Train score: {:.4f}'.format(grid_dtree.score(X_train,␣
 ↪y_train)))
```

```
Decision Tree parameters:  {'max_depth': 5}
Best Mean Cross-validation score: 0.81
Decision Tree Train score: 0.8742
```

[137]:
```
%matplotlib inline
def plot_feature_importances_cancer(model):
    n_features = X_train.shape[1]
    plt.barh(range(n_features), model.best_estimator_.feature_importances_,␣
 ↪align='center')
    plt.yticks(np.arange(n_features), X_train.dtype.names)
    plt.xlabel("Feature importance")
    plt.ylabel("Feature")
    plt.ylim(-1, n_features)

plot_feature_importances_cancer(grid_dtree)
```



3.1.3 KNN - .82289

[360]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[361]:
```
# Train a KNN model, report the coefficients, the best parameters, and model␣
 ↪performance
# hint: find the optimal k

knn = KNeighborsClassifier()

# define a list of parameters
param_knn = {'n_neighbors': range(1,10)}

#apply grid search
grid_knn = GridSearchCV(knn, param_knn, cv=5,␣
 ↪return_train_score=True,scoring=f2score)
grid_knn.fit(X_train, y_train)
```

[361]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=None,
             param_grid={'n_neighbors': range(1, 10)}, pre_dispatch='2*n_jobs',
             refit=True, return_train_score=True,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[363]:
```
#find best parameters
print("KNN parameters: {}".format(grid_knn.best_params_))
# Mean Cross Validation Score
print("Best Mean Cross-validation score: {:.2f}".format(grid_knn.best_score_))
print('Train score: {:.4f}'.format(grid_knn.score(X_train, y_train)))
```

```
KNN parameters: {'n_neighbors': 1}
Best Mean Cross-validation score: 0.83
Train score: 1.0000
```

3.1.4 Kernel SVC

[32]:
```
from sklearn.svm import SVC
param_svc = {'C':range(1,50,5),
             'kernel':['rbf','sigmoid'],
             'gamma' :['auto']}
```

[33]:
```
grid_svc = GridSearchCV(SVC(), param_svc, cv=5,␣
 ↪return_train_score=True,scoring=f2score)
grid_svc.fit(X_train, y_train)
```

[33]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': range(1, 50, 5), 'gamma': ['auto'],
                         'kernel': ['rbf', 'sigmoid']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[336]:
```
print("Best parameters: {}".format(grid_svc.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_svc.best_score_))
print('Train score: {:.4f}'.format(grid_svc.score(X_train, y_train)))
```

```
Best parameters: {'C': 21, 'gamma': 'auto', 'kernel': 'rbf'}
Best cross-validation score: 0.73
Train score: 0.9636
```

3.1.5 Linear SVC - .81501

[35]:
```
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
linear_svm = LinearSVC(max_iter=3000,dual=False).fit(X_train, y_train)
print("Coefficient shape: ", linear_svm.coef_.shape)
print("Intercept shape: ", linear_svm.intercept_.shape)
```

```
Coefficient shape:  (1, 29)
Intercept shape:  (1,)
```

[36]:
```
cv_scores_linear = cross_val_score(linear_svm, X_train, y_train,scoring=f2score)
```

Results

[337]:
```
print("Cross-validation scores: {}".format(cv_scores_linear))
print('Train score: {:.4f}'.format(linear_svm.score(X_train, y_train)))
```

```
Cross-validation scores: [0.78389831 0.84051724 0.75757576 0.80508475
0.84745763]
Train score: 0.9976
```

3.1.6 Random Forest - .81352

[57]:
```
from sklearn.ensemble import RandomForestClassifier
```

[107]:
```
#random forest
from sklearn.ensemble import RandomForestClassifier
rfc =RandomForestClassifier(random_state=0)
rfc_param = {
    'n_estimators': range(2,15,2),
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : range(2,15,3),
    'criterion' :['gini', 'entropy']
}

grid_rf = GridSearchCV(rfc, rfc_param,cv=5, return_train_score=True, scoring␣
 ↪=f2score)
grid_rf.fit(X_train,y_train)
```

[107]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,
                                              oob_score=False, random_state=0,
                                              verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(2, 15, 3),
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'n_estimators': range(2, 15, 2)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[108]:
```
print(f'Best Mean Cross Validation Score is {grid_rf.best_params_}')
print(f'Best Mean Cross Validation Score is {grid_rf.best_score_}')
print(f'Train score is {grid_rf.score(X_train,y_train)}')
```

```
Best Mean Cross Validation Score is {'criterion': 'gini', 'max_depth': 5,
```

```
'max_features': 'auto', 'n_estimators': 12}
Best Mean Cross Validation Score is 0.8500952309334316
Train score is 0.8614864864864864
```

3.1.7 Extra-Trees - .80586

[40]:
```
from  sklearn.ensemble import ExtraTreesClassifier
```

[41]:
```
etc= ExtraTreesClassifier(random_state=42)
etc_param = {
    'n_estimators': range(50,100,10),
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : range(2,15,2),
    'criterion' :['gini', 'entropy']
}
etc_grid = GridSearchCV(etc, etc_param,cv=5, return_train_score=True,scoring =␣
 ↪f2score)
etc_grid.fit(X_train,y_train)
```

[41]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0,
                                            class_weight=None, criterion='gini',
                                            max_depth=None, max_features='auto',
                                            max_leaf_nodes=None,
                                            max_samples=None,
                                            min_impurity_decrease=0.0,
                                            min_impurity_split=None,
                                            min_samples_leaf=1,
                                            min_samples_split=2,
                                            min_weight_fraction_leaf=0.0,
                                            n_estimators=100, n_jobs=None,
                                            oob_score=False, random_state=42,
                                            verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(2, 15, 2),
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'n_estimators': range(50, 100, 10)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[338]:
```
print(f'Best Mean Cross Validation Score is {etc_grid.best_params_}')
print(f'Best Mean Cross Validation Score is {etc_grid.best_score_}')
print(f'Train score is {etc_grid.score(X_train,y_train)}')
```

```
Best Mean Cross Validation Score is {'criterion': 'gini', 'max_depth': 14,
```

```
'max_features': 'auto', 'n_estimators': 50}
Best Mean Cross Validation Score is 0.8327935241589982
Train score is 0.8608990670059373
```

3.1.8 Gradient Boost - .79684

[43]:
```
from  sklearn.ensemble import GradientBoostingClassifier
```

[44]:
```
gbc= GradientBoostingClassifier(random_state=42)
gbc_param = {
              'max_depth' : range(2,15,5),
              'n_estimators' : range(50,80,10),
              'learning_rate' : [0.01,1.0,2],
             }
gbc_grid = GridSearchCV(gbc, gbc_param,cv=5, return_train_score=True,scoring =␣
 ↪f2score)
gbc_grid.fit(X_train,y_train)
```

[44]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=GradientBoostingClassifier(ccp_alpha=0.0,
                                                  criterion='friedman_mse',
                                                  init=None, learning_rate=0.1,
                                                  loss='deviance', max_depth=3,
                                                  max_features=None,
                                                  max_leaf_nodes=None,
                                                  min_impurity_decrease=0.0,
                                                  min_impurity_split=None,
                                                  min_samples_leaf=1,
                                                  min_samples_split=2,
                                                  min_weight_fraction_leaf=0.0,
                                                  n_estimators=100,
                                                  n_iter_no_c…
                                                  presort='deprecated',
                                                  random_state=42,
                                                  subsample=1.0, tol=0.0001,
                                                  validation_fraction=0.1,
                                                  verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.01, 1.0, 2],
                         'max_depth': range(2, 15, 5),
                         'n_estimators': range(50, 80, 10)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[340]:
```
print(f'Best Mean Cross Validation Score is {gbc_grid.best_score_}')
print(f'Best Mean Cross Validation Score is {gbc_grid.best_params_}')
```

```
print(f'Train score is {gbc_grid.score(X_train,y_train)}')
```

```
Best Mean Cross Validation Score is 0.7998976537815454
Best Mean Cross Validation Score is {'learning_rate': 2, 'max_depth': 12,
'n_estimators': 50}
Train score is 1.0
```

3.1.9 XGBoost - .82116

[46]:
```
from xgboost import XGBClassifier
```

[47]:
```
from  xgboost import XGBClassifier
xgbc= XGBClassifier(random_state=42,early_stopping_rounds=2,objective= 'binary:
 ↪logistic')
xgbc_param = {
              'max_depth' : range(2,15,2),
              'n_estimators' : range(50,80,10),
              'learning_rate' : [0.1,0.01],
               'min_child_weight' : [1,3,5,7],
                'subsample':[0.6,0.7,0.8,0.9,1]
             }
xgbc_grid = GridSearchCV(xgbc, xgbc_param,cv=5, return_train_score=True,scoring␣
 ↪= f2score)
xgbc_grid.fit(X_train,y_train)
```

[47]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None,
                                     early_stopping_rounds=2, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=None, max_delta_step=None,
                                     max_depth=None, min_child_weight=None,
                                     missing=nan, monotone_…
                                     subsample=None, tree_method=None,
                                     validate_parameters=False,
                                     verbosity=None),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.1, 0.01],
                         'max_depth': range(2, 15, 2),
                         'min_child_weight': [1, 3, 5, 7],
                         'n_estimators': range(50, 80, 10),
                         'subsample': [0.6, 0.7, 0.8, 0.9, 1]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[341]:
```
print(f'Best Mean Cross Validation Score is {xgbc_grid.best_score_}')
print(f'Best Mean Cross Validation Score is {xgbc_grid.best_params_}')
print(f'Train score is {xgbc_grid.score(X_train,y_train)}')
```

```
Best Mean Cross Validation Score is 0.8536961171789608
Best Mean Cross Validation Score is {'learning_rate': 0.1, 'max_depth': 4,
'min_child_weight': 3, 'n_estimators': 50, 'subsample': 0.7}
Train score is 0.867003367003367
```

3.2 Cost Sensitive Models

3.2.1 Logistic Regression - .80909

[49]:
```
logreg = LogisticRegression(random_state=42)
param_grid = {'class_weight':['balanced','balanced_subsample'],
              'C': [0.001, 0.01, 0.1, 1, 10,1000],
                'solver':['lbfgs'],
               'penalty':['l2'],
               'max_iter':range(1000,1500,10)}# [{0:100,1:1}, {0:10,1:1}, {0:1,1:
 ↪1}, {0:1,1:10}, {0:1,1:100}]}

#apply grid search
grid_CSlogreg= GridSearchCV(logreg, param_grid, cv=5, n_jobs=-1,␣
 ↪scoring=f2score)
grid_CSlogreg.fit(X_train,y_train)
```

[49]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='auto',
                                          n_jobs=None, penalty='l2',
                                          random_state=42, solver='lbfgs',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 1000],
                         'class_weight': ['balanced', 'balanced_subsample'],
                         'max_iter': range(1000, 1500, 10), 'penalty': ['l2'],
                         'solver': ['lbfgs']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[105]:
```
print("Best parameters: {}".format(grid_CSlogreg.best_params_))
```

```
print("Best Mean cross-validation score: {:.2f}".format(grid_CSlogreg.
 ↪best_score_))
print('Train score: {:.4f}'.format(grid_logit.score(X_train, y_train)))
```

```
Best parameters: {'C': 10, 'class_weight': 'balanced_subsample', 'max_iter':
1000, 'penalty': 'l2', 'solver': 'lbfgs'}
Best Mean cross-validation score: 0.80
Train score: 0.8177
```

3.2.2 Descision Tree - .74823

[51]:
```
dtree = DecisionTreeClassifier()
param_grid = {'class_weight': ['balanced','balanced_subsample']}#[{0:100,1:1},␣
 ↪{0:10,1:1}, {0:1,1:1}, {0:1,1:10}, {0:1,1:100}]}
#apply grid search
grid_CSdtree= GridSearchCV(dtree, param_grid, cv=5, n_jobs=2, scoring=f2score)
grid_CSdtree.fit(X_train,y_train)
```

[51]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=2,
             param_grid={'class_weight': ['balanced', 'balanced_subsample']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[342]:
```
print("Best parameters: {}".format(grid_CSdtree.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(grid_CSdtree.
 ↪best_score_))
print(f'Train score is {grid_CSdtree.score(X_train,y_train)}')
```

```
Best parameters: {'class_weight': 'balanced'}
Best Mean cross-validation score: 0.78
Train score is 1.0
```

3.2.3 SVM

[53]:
```
# svc = SVC()
# param_grid = {'class_weight': ['balanced','balanced_subsample']}#[{0:100,1:
 ↪1}, {0:10,1:1}, {0:1,1:1}, {0:1,1:10}, {0:1,1:100}]}

# #apply grid search
# grid_CSsvc= GridSearchCV(svc, param_grid, cv=5, n_jobs=2, scoring=f2score)
# grid_CSsvc.fit(X_train,y_train)
```

Results

[54]:
```
# print("Best parameters: {}".format(grid_CSsvc.best_params_))
# print("Best Mean cross-validation score: {:.2f}".format(grid_CSsvc.
 ↪best_score_))
```

3.2.4 Random Forest - .80586

[55]:
```
from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import BaggingClassifier
```

[58]:
```
# define model
SCRandom = RandomForestClassifier(random_state=1)
SCRandom_param = {'n_estimators':range(2,10,2),
                  'class_weight':['balanced','balanced_subsample']}

# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
#apply grid search
grid_CSRF= GridSearchCV(SCRandom, SCRandom_param, cv=cv, n_jobs=-1,␣
 ↪scoring=f2score)
grid_CSRF.fit(X_train,y_train)
```

[58]:
```
GridSearchCV(cv=RepeatedStratifiedKFold(n_repeats=3, n_splits=10,
random_state=1),
             error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
```

```
                                                min_samples_split=2,
                                                min_weight_fraction_leaf=0.0,
                                                n_estimators=100, n_jobs=None,
                                                oob_score=False, random_state=1,
                                                verbose=0, warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'class_weight': ['balanced', 'balanced_subsample'],
                         'n_estimators': range(2, 10, 2)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[343]:
```
print("Best parameters: {}".format(grid_CSRF.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(grid_CSRF.best_score_))
print(f'Train score is {grid_CSRF.score(X_train,y_train)}')
```

```
Best parameters: {'class_weight': 'balanced', 'n_estimators': 8}
Best Mean cross-validation score: 0.79
Train score is 0.9727947238252268
```

3.2.5 XGBoost - .83182

[400]:
```
# SCxgboost = XGBClassifier()
# param_SCgrid = {'scale_pos_weight': [1, 10, 25, 50, 75, 99, 100, 1000]}

# #apply grid search
# grid_xgboost= GridSearchCV(SCxgboost, param_SCgrid, cv=5, n_jobs=-1,␣
 ↪scoring=f2score)
# grid_xgboost.fit(X_train,y_train)


SCxgboost = XGBClassifier()
param_SCgrid = {'scale_pos_weight': range(1000,1200,50),
                'class_weight':['balanced','balanced_subsample']}

#apply grid search
grid_xgboost= GridSearchCV(SCxgboost, param_SCgrid, cv=5, n_jobs=-1,␣
 ↪scoring=f2score)
grid_xgboost.fit(X_train,y_train)

# SCxgboost = XGBClassifier(early_stopping_rounds=5)
# param_SCgrid = {'scale_pos_weight': range(1000,1200,50),
#                 'class_weight':['balanced','balanced_subsample'],
#                 'n_estimators':range(1000,1200,20),
#                 'learning_rate':[0.05,0.01,0.1]}

# #apply grid search
```

```
# grid_xgboost= GridSearchCV(SCxgboost, param_SCgrid, cv=5, n_jobs=-1,␣
 ↪scoring=f2score)
# grid_xgboost.fit(X_train,y_train)
```

[400]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=None, max_delta_step=None,
                                     max_depth=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     n_estim…
                                     reg_lambda=None, scale_pos_weight=None,
                                     subsample=None, tree_method=None,
                                     validate_parameters=False,
                                     verbosity=None),
             iid='deprecated', n_jobs=-1,
             param_grid={'class_weight': ['balanced', 'balanced_subsample'],
                         'scale_pos_weight': range(1000, 1200, 50)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[401]:
```
print("Best parameters: {}".format(grid_xgboost.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(grid_xgboost.
 ↪best_score_))
print(f'Train score is {grid_xgboost.score(X_train,y_train)}')
```

```
Best parameters: {'class_weight': 'balanced', 'scale_pos_weight': 1000}
Best Mean cross-validation score: 0.86
Train score is 1.0
```

3.2.6 Extra Trees - .80586

[62]:
```
#extratrees
SCExtraTree = ExtraTreesClassifier(random_state=42)
SCExtraTree_param ={'n_estimators':range(2,50,5),
                    'class_weight':['balanced','balanced_subsample']}

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=42)

grid_SCExTree= GridSearchCV(SCExtraTree, SCExtraTree_param, cv=cv, n_jobs=-1,␣
 ↪scoring=f2score)
grid_SCExTree.fit(X_train,y_train)
```

[62]:
```
GridSearchCV(cv=RepeatedStratifiedKFold(n_repeats=3, n_splits=10,
random_state=42),
             error_score=nan,
             estimator=ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0,
                                            class_weight=None, criterion='gini',
                                            max_depth=None, max_features='auto',
                                            max_leaf_nodes=None,
                                            max_samples=None,
                                            min_impurity_decrease=0.0,
                                            min_impurity_split=None,
                                            min_samples_leaf=1,
                                            min_samples_split=2,
                                            min_weight_fraction_leaf=0.0,
                                            n_estimators=100, n_jobs=None,
                                            oob_score=False, random_state=42,
                                            verbose=0, warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'class_weight': ['balanced', 'balanced_subsample'],
                         'n_estimators': range(2, 50, 5)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[346]:
```
print("Best parameters: {}".format(grid_SCExTree.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(grid_SCExTree.
 ↪best_score_))
print(f'Train score is {grid_SCExTree.score(X_train,y_train)}')
```

```
Best parameters: {'class_weight': 'balanced', 'n_estimators': 27}
Best Mean cross-validation score: 0.84
Train score is 1.0
```

3.2.7 Bagging Decision Tree - Under Sampling

[66]:
```
from imblearn.under_sampling import RandomUnderSampler
from sklearn.ensemble import BaggingClassifier
from imblearn.pipeline import Pipeline
from imblearn.ensemble import BalancedBaggingClassifier
```

[67]:
```
bag_dtree1 = Pipeline(steps=[('random',RandomUnderSampler(random_state=42)),
                             ␣
 ↪('bagging',BalancedBaggingClassifier(base_estimator=
                                                                 ␣
 ↪DecisionTreeClassifier(class_weight='balanced'),
                                                                 ␣
 ↪random_state=42))])

bag_dtree1_param = {'bagging__n_estimators':[100]}

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=42)

grid_SCBagging = GridSearchCV(bag_dtree1, bag_dtree1_param,cv=cv,␣
 ↪scoring=f2score,return_train_score=True)
grid_SCBagging.fit(X_train,y_train)
```

[67]:
```
GridSearchCV(cv=RepeatedStratifiedKFold(n_repeats=3, n_splits=10,
random_state=42),
             error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('random',
                                        RandomUnderSampler(random_state=42,
                                                           replacement=False,
sampling_strategy='auto')),
                                       ('bagging',
BalancedBaggingClassifier(base_estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                          class_weight='balanced',
                          criterion='g…
max_features=1.0,
max_samples=1.0,
n_estimators=10,
                                                                 n_jobs=None,
oob_score=False,
random_state=42,
replacement=False,
sampling_strategy='auto',
                                                                 verbose=0,
warm_start=False))],
                                verbose=False),
             iid='deprecated', n_jobs=None,
```

```
                param_grid={'bagging__n_estimators': [100]},
                pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[347]:
```
print("Best parameters: {}".format(grid_SCBagging.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(grid_SCBagging.
 ↪best_score_))
print(f'Train score is {grid_SCBagging.score(X_train,y_train)}')
```

```
Best parameters: {'bagging__n_estimators': 100}
Best Mean cross-validation score: 0.47
Train score is 0.3774752475247524
```

3.3 Data Sampling

3.3.1 Over Sampling

[69]:
```
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import SVMSMOTE
from imblearn.over_sampling import ADASYN
from imblearn.pipeline import Pipeline
```

[70]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier

from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.dummy import DummyClassifier
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc
from sklearn.metrics import make_scorer
from matplotlib import pyplot
from sklearn.model_selection import GridSearchCV
```

Logistic Regression

[71]:
```
# GridSearch with oversampling
```

```
pipe_logit_smote =␣
 ↪Pipeline([('smote',SMOTE()),('Logistic',LogisticRegression(random_state=42))])

param_grid = {
    # try different feature engineering parameters
    'smote__k_neighbors': [1,2,3],
    'Logistic__C': [0.001, 0.01],
    'Logistic__solver':['lbfgs'],
    'Logistic__penalty':['l2'],
    'Logistic__max_iter':[1200]#range(1000,1500,10)
}

#apply grid search
grid_logitsmote= GridSearchCV(pipe_logit_smote, param_grid, cv=5, n_jobs=2,␣
 ↪scoring=f2score)
grid_logitsmote.fit(X_train,y_train)
```

[71]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('smote',
                                        SMOTE(k_neighbors=5, n_jobs=None,
                                              random_state=None,
                                              sampling_strategy='auto')),
                                       ('Logistic',
                                        LogisticRegression(C=1.0,
                                                           class_weight=None,
                                                           dual=False,
                                                           fit_intercept=True,
                                                           intercept_scaling=1,
                                                           l1_ratio=None,
                                                           max_iter=100,
                                                           multi_class='auto',
                                                           n_jobs=None,
                                                           penalty='l2',
                                                           random_state=42,
                                                           solver='lbfgs',
                                                           tol=0.0001,
                                                           verbose=0,
                                                           warm_start=False))],
                                verbose=False),
             iid='deprecated', n_jobs=2,
             param_grid={'Logistic__C': [0.001, 0.01],
                         'Logistic__max_iter': [1200],
                         'Logistic__penalty': ['l2'],
                         'Logistic__solver': ['lbfgs'],
                         'smote__k_neighbors': [1, 2, 3]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
```

```
                scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[348]:
```
print("Best parameters: {}".format(grid_logitsmote.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(grid_logitsmote.
 ↪best_score_))
print(f'Train score is {grid_logitsmote.score(X_train,y_train)}')
```

```
Best parameters: {'Logistic__C': 0.001, 'Logistic__max_iter': 1200,
'Logistic__penalty': 'l2', 'Logistic__solver': 'lbfgs', 'smote__k_neighbors': 2}
Best Mean cross-validation score: 0.70
Train score is 0.7057340894770006
```

Decision Tree

[73]:
```
#decisiontreee
pipe_ds_dtree =␣
 ↪Pipeline([('smote',SMOTE()),('dtree',DecisionTreeClassifier(random_state=42))])
param_ds_dtree = {'smote__k_neighbors': range(2,8,2),
                  'dtree__max_depth': range(1,20,5)}

DOS_dtree= GridSearchCV(pipe_ds_dtree,param_ds_dtree, cv=5, n_jobs=2,␣
 ↪scoring=f2score)
DOS_dtree.fit(X_train, y_train)
```

[73]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('smote',
                                        SMOTE(k_neighbors=5, n_jobs=None,
                                              random_state=None,
                                              sampling_strategy='auto')),
                                       ('dtree',
                                        DecisionTreeClassifier(ccp_alpha=0.0,
class_weight=None,
                                                               criterion='gini',
                                                               max_depth=None,
max_features=None,
max_leaf_nodes=None,
min_impurity_decrease=0.0,
min_impurity_split=None,
min_samples_leaf=1,
min_samples_split=2,
min_weight_fraction_leaf=0.0,
presort='deprecated',
                                                               random_state=42,
splitter='best'))],
                                verbose=False),
```

```
                iid='deprecated', n_jobs=2,
                param_grid={'dtree__max_depth': range(1, 20, 5),
                            'smote__k_neighbors': range(2, 8, 2)},
                pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[349]:
```
print("Best parameters: {}".format(DOS_dtree.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(DOS_dtree.best_score_))
print(f'Train score is {DOS_dtree.score(X_train,y_train)}')
```

```
Best parameters: {'dtree__max_depth': 16, 'smote__k_neighbors': 4}
Best Mean cross-validation score: 0.74
Train score is 0.959874114870181
```

KNN - .80279

[406]:
```
#knn
DOS_knn = Pipeline([('smote',SMOTE()),('knn',KNeighborsClassifier())])
param_DOS_knn = {'smote__k_neighbors': range(2,8,2),
                 'knn__n_neighbors': range(1,10,2)}

DOSgrid_knn= GridSearchCV(DOS_knn,param_DOS_knn, cv=5, n_jobs=-1,␣
 ↪scoring=f2score)
DOSgrid_knn.fit(X_train, y_train)
```

[406]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('smote',
                                        SMOTE(k_neighbors=5, n_jobs=None,
                                              random_state=None,
                                              sampling_strategy='auto')),
                                       ('knn',
                                        KNeighborsClassifier(algorithm='auto',
                                                             leaf_size=30,
                                                             metric='minkowski',
                                                             metric_params=None,
                                                             n_jobs=None,
                                                             n_neighbors=5, p=2,
weights='uniform'))],
                                verbose=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'knn__n_neighbors': range(1, 10, 2),
                         'smote__k_neighbors': range(2, 8, 2)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[408]:
```
print("Best parameters: {}".format(DOSgrid_knn.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(DOSgrid_knn.
 ↪best_score_))
print(f'Train score is {DOSgrid_knn.score(X_train,y_train)}')
```

```
Best parameters: {'knn__n_neighbors': 1, 'smote__k_neighbors': 2}
Best Mean cross-validation score: 0.82
Train score is 1.0
```

[411]:
```
test_data_labels = DOSgrid_knn.predict(X_test)

# Create predictions to be submitted!
pd.DataFrame({'Id': test.Id, 'Target': test_data_labels}).
 ↪to_csv('Project2_solution_base.csv', index =False)
print("Done :D")
```

```
Done :D
```

SVM

[77]:
```
# #svm
# DOS_svm =␣
 ↪Pipeline([('smote',SMOTE()),('svm',SVC(random_state=42,probability=True))])
# param_DOS_svm = {'smote__k_neighbors': [1,2,3,4,5],
#                  'svm__C':[1,20,30],
#               'svm__kernel':['rbf','sigmoid'],
#               'svm__gamma' :['auto']}

# DOSgrid_svm= GridSearchCV(DOS_svm,param_DOS_svm, cv=5, n_jobs=-1,␣
 ↪scoring=f2score)
# DOSgrid_svm.fit(X_train, y_train)
```

[78]:
```
# print("Best parameters: {}".format(DOSgrid_svm.best_params_))
# print("Best Mean cross-validation score: {:.2f}".format(DOSgrid_svm.
 ↪best_score_))
```

Random Forest - .81395

[79]:
```
#randomforest
DOS_rf =␣
 ↪Pipeline([('smote',SMOTE()),('rf',RandomForestClassifier(random_state=42))])
param_DOS_rf = {'smote__k_neighbors': [1,2,3,4,5],
                   'rf__n_estimators': range(20,80,10),
                    'rf__max_features': ['auto', 'sqrt', 'log2'],
                    'rf__max_depth' : range(2,8,4),
                    'rf__criterion' :['gini', 'entropy']}
```

```
DOSgrid_rf= GridSearchCV(DOS_rf,param_DOS_rf, cv=5, n_jobs=-1, scoring=f2score)
DOSgrid_rf.fit(X_train, y_train)
```

[79]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('smote',
                                        SMOTE(k_neighbors=5, n_jobs=None,
                                              random_state=None,
                                              sampling_strategy='auto')),
                                       ('rf',
                                        RandomForestClassifier(bootstrap=True,
                                                               ccp_alpha=0.0,
class_weight=None,
                                                               criterion='gini',
                                                               max_depth=None,
max_features='auto',
max_leaf_nodes=None,
                                                               max_samples=None,
min_impurity_decrease=0.0,
                                                               m…
warm_start=False))],
                                verbose=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'rf__criterion': ['gini', 'entropy'],
                         'rf__max_depth': range(2, 8, 4),
                         'rf__max_features': ['auto', 'sqrt', 'log2'],
                         'rf__n_estimators': range(20, 80, 10),
                         'smote__k_neighbors': [1, 2, 3, 4, 5]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[350]:
```
print("Best parameters: {}".format(DOSgrid_rf.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(DOSgrid_rf.best_score_))
print(f'Train score is {DOSgrid_rf.score(X_train,y_train)}')
```

```
Best parameters: {'rf__criterion': 'gini', 'rf__max_depth': 6,
'rf__max_features': 'auto', 'rf__n_estimators': 60, 'smote__k_neighbors': 1}
Best Mean cross-validation score: 0.86
Train score is 0.9226430298146655
```

Easy Ensembler

[81]:
```
#easy ensemble classifier
from imblearn.ensemble import EasyEnsembleClassifier
DOS_easy = Pipeline([('smote',SMOTE()),('easy',EasyEnsembleClassifier())])
param_DOS_easy = {'smote__k_neighbors': range(2,10,2)}
```

```
DOSgrid_easy= GridSearchCV(DOS_easy,param_DOS_easy, cv=5, n_jobs=-1,␣
 ↪scoring=f2score)
DOSgrid_easy.fit(X_train, y_train)
```

[81]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('smote',
                                        SMOTE(k_neighbors=5, n_jobs=None,
                                              random_state=None,
                                              sampling_strategy='auto')),
                                       ('easy',
EasyEnsembleClassifier(base_estimator=None,
                                                              n_estimators=10,
                                                              n_jobs=None,
random_state=None,
replacement=False,
sampling_strategy='auto',
                                                              verbose=0,
warm_start=False))],
                                verbose=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'smote__k_neighbors': range(2, 10, 2)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[351]:
```
print("Best parameters: {}".format(DOSgrid_easy.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(DOSgrid_easy.
 ↪best_score_))
print(f'Train score is {DOSgrid_easy.score(X_train,y_train)}')
```

```
Best parameters: {'smote__k_neighbors': 8}
Best Mean cross-validation score: 0.68
Train score is 0.6595622119815668
```

XGBoost - .79216

[83]:
```
from xgboost import XGBClassifier
#xgb
DOS_xgb = Pipeline([('smote',SMOTE()),('xgboost',XGBClassifier(random_state=42,
                                                                ␣
 ↪early_stopping_rounds=2,
#                                                                 ␣
 ↪n_estimators=100,
                                                                ␣
 ↪objective ='binary:logistic',
```

```
#                                                                              ␣
 ↪max_depth = 4,
                                                                               ␣
 ↪))])
param_DOS_xgb = {'smote__k_neighbors': [1,2,3,4,5],
                 'xgboost__max_depth' : range(2,8,4),
                 'xgboost__n_estimators' : range(50,70,20),
                 'xgboost__learning_rate' : [0.1],
                 'xgboost__min_child_weight' : [1,3,5,7],
                 'xgboost__subsample':[0.6,0.7,0.8,0.9,1]}

DOSgrid_xgb= GridSearchCV(DOS_xgb,param_DOS_xgb, cv=3, n_jobs=-1,␣
 ↪scoring=f2score)
DOSgrid_xgb.fit(X_train, y_train)
```

[83]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('smote',
                                        SMOTE(k_neighbors=5, n_jobs=None,
                                              random_state=None,
                                              sampling_strategy='auto')),
                                       ('xgboost',
                                        XGBClassifier(base_score=None,
                                                      booster=None,
                                                      colsample_bylevel=None,
                                                      colsample_bynode=None,
                                                      colsample_bytree=None,
                                                      early_stopping_rounds=2,
                                                      gamma=None, gpu_id=None,
                                                      importance_type='gain',
                                                      i…
             iid='deprecated', n_jobs=-1,
             param_grid={'smote__k_neighbors': [1, 2, 3, 4, 5],
                         'xgboost__learning_rate': [0.1],
                         'xgboost__max_depth': range(2, 8, 4),
                         'xgboost__min_child_weight': [1, 3, 5, 7],
                         'xgboost__n_estimators': range(50, 70, 20),
                         'xgboost__subsample': [0.6, 0.7, 0.8, 0.9, 1]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[352]:
```
print("Best parameters: {}".format(DOSgrid_xgb.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(DOSgrid_xgb.
 ↪best_score_))
print(f'Train score is {DOSgrid_xgb.score(X_train,y_train)}')
```

```
Best parameters: {'smote__k_neighbors': 1, 'xgboost__learning_rate': 0.1,
'xgboost__max_depth': 6, 'xgboost__min_child_weight': 5,
'xgboost__n_estimators': 50, 'xgboost__subsample': 0.8}
Best Mean cross-validation score: 0.84
Train score is 0.9345794392523363
```

Neural Network - MLP Classifier

[152]:
```
from sklearn.neural_network import MLPClassifier
```

[159]:
```
DOS_mlp = Pipeline([('smote',SMOTE()),
                    ('mlp',MLPClassifier(random_state=42))])

parameters = {'mlp__solver': ['lbfgs'],
              'mlp__max_iter': [1200],
              'mlp__alpha': 10.0 ** -np.arange(1, 10),
              'mlp__hidden_layer_sizes':np.arange(10, 15),
              'mlp__random_state':range(2,12,2)}
DOSMLPGrid = GridSearchCV(DUS_mlp, parameters,cv=5, n_jobs=-1,scoring=f2score)

DOSMLPGrid.fit(X_train, y_train)
```

[159]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('random',
                                        RandomUnderSampler(random_state=42,
                                                           replacement=False,
sampling_strategy='auto')),
                                       ('mlp',
                                        MLPClassifier(activation='relu',
                                                      alpha=0.0001,
                                                      batch_size='auto',
                                                      beta_1=0.9, beta_2=0.999,
                                                      early_stopping=False,
                                                      epsilon=1e-08,
                                                      hidden_layer_sizes=(100,),
                                                      learning_rate='constant',
                                                      learning…
             iid='deprecated', n_jobs=-1,
             param_grid={'mlp__alpha': array([1.e-01, 1.e-02, 1.e-03, 1.e-04,
1.e-05, 1.e-06, 1.e-07, 1.e-08,
       1.e-09]),
                         'mlp__hidden_layer_sizes': array([10, 11, 12, 13, 14]),
                         'mlp__max_iter': [1200],
                         'mlp__random_state': range(2, 12, 2),
                         'mlp__solver': ['lbfgs']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

[353]:
```
print("Best parameters: {}".format(DOSMLPGrid.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(DOSMLPGrid.best_score_))
print(f'Train score is {DOSMLPGrid.score(X_train,y_train)}')
```

```
Best parameters: {'mlp__alpha': 0.1, 'mlp__hidden_layer_sizes': 10,
'mlp__max_iter': 1200, 'mlp__random_state': 2, 'mlp__solver': 'lbfgs'}
Best Mean cross-validation score: 0.43
Train score is 0.4450930317402408
```

3.3.2 Under Sampling

Logistic Regression

[85]:
```
from imblearn.under_sampling import RandomUnderSampler
```

[86]:
```
# GridSearch with oversampling
pipe_logit_RS = Pipeline([('random',RandomUnderSampler(random_state=42)),
                          ('Logistic',LogisticRegression(random_state=42))])

param_grid = {
    # try different feature engineering parameters
    'Logistic__C': [0.001, 0.01, 0.1, 1, 10,1000],
    'Logistic__solver':['lbfgs'],
    'Logistic__penalty':['l2'],
    'Logistic__max_iter':[1200]#range(1000,1500,10)
}

#apply grid search
grid_logitRS= GridSearchCV(pipe_logit_smote, param_grid, cv=5, n_jobs=-1,␣
 ↪scoring=f2score)
grid_logitRS.fit(X_train,y_train)
```

[86]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('smote',
                                        SMOTE(k_neighbors=5, n_jobs=None,
                                              random_state=None,
                                              sampling_strategy='auto')),
                                       ('Logistic',
                                        LogisticRegression(C=1.0,
                                                           class_weight=None,
                                                           dual=False,
                                                           fit_intercept=True,
                                                           intercept_scaling=1,
                                                           l1_ratio=None,
                                                           max_iter=100,
                                                           multi_class='auto',
                                                           n_jobs=None,
```

```
                                                       penalty='l2',
                                                       random_state=42,
                                                       solver='lbfgs',
                                                       tol=0.0001,
                                                       verbose=0,
                                                       warm_start=False))],
                                verbose=False),
                   iid='deprecated', n_jobs=-1,
                   param_grid={'Logistic__C': [0.001, 0.01, 0.1, 1, 10, 1000],
                               'Logistic__max_iter': [1200],
                               'Logistic__penalty': ['l2'],
                               'Logistic__solver': ['lbfgs']},
                   pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                   scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[354]:
```
print("Best parameters: {}".format(grid_logitRS.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(grid_logitRS.
 ↪best_score_))
print(f'Train score is {grid_logitRS.score(X_train,y_train)}')
```

```
Best parameters: {'Logistic__C': 0.001, 'Logistic__max_iter': 1200,
'Logistic__penalty': 'l2', 'Logistic__solver': 'lbfgs'}
Best Mean cross-validation score: 0.69
Train score is 0.7070707070707071
```

Decision Tree

[88]:
```
#decisiontreee
pipe_dus_dtree = Pipeline([('random',RandomUnderSampler(random_state=42)),
                           ('dtree',DecisionTreeClassifier(random_state=42))])
param_dus_dtree = {'dtree__max_depth': range(1,20)}

DUS_dtree= GridSearchCV(pipe_dus_dtree,param_dus_dtree, cv=5, n_jobs=-1,␣
 ↪scoring=f2score)
DUS_dtree.fit(X_train, y_train)
```

[88]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('random',
                                        RandomUnderSampler(random_state=42,
                                                           replacement=False,
sampling_strategy='auto')),
                                       ('dtree',
                                        DecisionTreeClassifier(ccp_alpha=0.0,
class_weight=None,
                                                               criterion='gini',
```

```
                                                                max_depth=None,
                       max_features=None,
                       max_leaf_nodes=None,
                       min_impurity_decrease=0.0,
                       min_impurity_split=None,
                       min_samples_leaf=1,
                       min_samples_split=2,
                       min_weight_fraction_leaf=0.0,
                       presort='deprecated',
                                                                random_state=42,
                       splitter='best'))],
                                        verbose=False),
                    iid='deprecated', n_jobs=-1,
                    param_grid={'dtree__max_depth': range(1, 20)},
                    pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                    scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[355]:
```
print("Best parameters: {}".format(DUS_dtree.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(DUS_dtree.best_score_))
print(f'Train score is {DUS_dtree.score(X_train,y_train)}')
```

```
Best parameters: {'dtree__max_depth': 1}
Best Mean cross-validation score: 0.51
Train score is 0.5549654806160383
```

KNN

[404]:
```
#knn
DUS_knn = Pipeline([('random',RandomUnderSampler(random_state=42)),
                    ('knn',KNeighborsClassifier())])
param_DUS_knn = {'knn__n_neighbors': range(1,10)}

DUSgrid_knn= GridSearchCV(DUS_knn,param_DUS_knn, cv=5, n_jobs=-1,␣
 ↪scoring=f2score)
DUSgrid_knn.fit(X_train, y_train)
```

[404]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('random',
                                        RandomUnderSampler(random_state=42,
                                                           replacement=False,
                       sampling_strategy='auto')),
                                       ('knn',
                                        KNeighborsClassifier(algorithm='auto',
                                                             leaf_size=30,
                                                             metric='minkowski',
```

```
                                                      metric_params=None,
                                                      n_jobs=None,
                                                      n_neighbors=5, p=2,
weights='uniform'))],
                                    verbose=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'knn__n_neighbors': range(1, 10)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[405]:
```
print("Best parameters: {}".format(DUSgrid_knn.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(DUSgrid_knn.
 ↪best_score_))
```

```
Best parameters: {'knn__n_neighbors': 8}
Best Mean cross-validation score: 0.72
```

SVM

[92]:
```
# #svm
# DUS_svm = Pipeline([('random',RandomUnderSampler(random_state=42)),
#                     ('svm',SVC(random_state=42))])
# param_DUS_svm = {'svm__C':[1,20,30],
#                  'svm__kernel':['rbf','sigmoid'],
#                  'svm__gamma' :['auto']}

# DUSgrid_svm= GridSearchCV(DUS_svm,param_DUS_svm, cv=5, n_jobs=-1,␣
 ↪scoring=f2score)
# DUSgrid_svm.fit(X_train, y_train)
```

[93]:
```
# print("Best parameters: {}".format(DUSgrid_svm.best_params_))
# print("Best Mean cross-validation score: {:.2f}".format(DUSgrid_svm.
 ↪best_score_))
```

Random Forest - .76518

[143]:
```
#randomforest
DUS_rf = Pipeline([('random',RandomUnderSampler(random_state=42)),
                   ('rf',RandomForestClassifier(random_state=42))])
param_DUS_rf = {'rf__n_estimators': range(20,100,10),
                'rf__max_features': ['auto', 'sqrt', 'log2'],
                'rf__max_depth' : range(2,15,4),
                'rf__criterion' :['gini', 'entropy']}

DUSgrid_rf= GridSearchCV(DUS_rf,param_DUS_rf, cv=5, n_jobs=-1, scoring=f2score)
DUSgrid_rf.fit(X_train, y_train)
```

[143]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('random',
                                        RandomUnderSampler(random_state=42,
                                                           replacement=False,
sampling_strategy='auto')),
                                       ('rf',
                                        RandomForestClassifier(bootstrap=True,
                                                               ccp_alpha=0.0,
class_weight=None,
                                                               criterion='gini',
                                                               max_depth=None,
max_features='auto',
max_leaf_nodes=None,
                                                               max_samples=None,
min_impurity_decrease=0…
                                                               random_state=42,
                                                               verbose=0,
warm_start=False))],
                                verbose=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'rf__criterion': ['gini', 'entropy'],
                         'rf__max_depth': range(2, 15, 4),
                         'rf__max_features': ['auto', 'sqrt', 'log2'],
                         'rf__n_estimators': range(20, 100, 10)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[356]:
```
print("Best parameters: {}".format(DUSgrid_rf.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(DUSgrid_rf.best_score_))
print(f'Train score is {DUSgrid_rf.score(X_train,y_train)}')
```

```
Best parameters: {'rf__criterion': 'gini', 'rf__max_depth': 2,
'rf__max_features': 'auto', 'rf__n_estimators': 90}
Best Mean cross-validation score: 0.82
Train score is 0.8154670750382847
```

Easy Ensembler

[96]:
```
#easy ensemble classifier
from imblearn.ensemble import EasyEnsembleClassifier
DUS_easy = Pipeline([('random',RandomUnderSampler()),
                     ('easy',EasyEnsembleClassifier())])
param_DUS_easy = {'easy__n_estimators':range(2,50,5)}
```

```
DUSgrid_easy= GridSearchCV(DUS_easy,param_DUS_easy, cv=5, n_jobs=-1,␣
 ↪scoring=f2score)
DUSgrid_easy.fit(X_train, y_train)
```

[96]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('random',
                                        RandomUnderSampler(random_state=None,
                                                           replacement=False,
sampling_strategy='auto')),
                                       ('easy',
EasyEnsembleClassifier(base_estimator=None,
                                                               n_estimators=10,
                                                               n_jobs=None,
random_state=None,
replacement=False,
sampling_strategy='auto',
                                                               verbose=0,
warm_start=False))],
                                verbose=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'easy__n_estimators': range(2, 50, 5)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[357]:
```
print("Best parameters: {}".format(DUSgrid_easy.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(DUSgrid_easy.
 ↪best_score_))
print(f'Train score is {DUSgrid_easy.score(X_train,y_train)}')
```

```
Best parameters: {'easy__n_estimators': 37}
Best Mean cross-validation score: 0.46
Train score is 0.5091819699499165
```

XGBoost

[98]:
```
#xgb
DUS_xgb = Pipeline([('random',RandomUnderSampler(random_state=42)),
                    ('xgboost',XGBClassifier(random_state=42,
                                             early_stopping_rounds=2,
                                             objective ='binary:logistic',
                                              ))])
param_DUS_xgb = {'xgboost__max_depth' : range(2,8,2),
                 'xgboost__n_estimators' : range(50,80,20),
                 'xgboost__learning_rate' : [0.1],
                 'xgboost__min_child_weight' : [1,3,5,7],
```

```
                'xgboost__subsample':[0.6,0.7,0.8,0.9,1]}

DUSgrid_xgb= GridSearchCV(DUS_xgb,param_DUS_xgb, cv=5, n_jobs=-1,␣
 ↪scoring=f2score)
DUSgrid_xgb.fit(X_train, y_train)
```

[98]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('random',
                                        RandomUnderSampler(random_state=42,
                                                           replacement=False,
sampling_strategy='auto')),
                                       ('xgboost',
                                        XGBClassifier(base_score=None,
                                                      booster=None,
                                                      colsample_bylevel=None,
                                                      colsample_bynode=None,
                                                      colsample_bytree=None,
                                                      early_stopping_rounds=2,
                                                      gamma=None, gpu_id=None,
                                                      importance_type='gai…
                                                      verbosity=None))],
                                verbose=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'xgboost__learning_rate': [0.1],
                         'xgboost__max_depth': range(2, 8, 2),
                         'xgboost__min_child_weight': [1, 3, 5, 7],
                         'xgboost__n_estimators': range(50, 80, 20),
                         'xgboost__subsample': [0.6, 0.7, 0.8, 0.9, 1]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[358]:
```
print("Best parameters: {}".format(DUSgrid_xgb.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(DUSgrid_xgb.
 ↪best_score_))
print(f'Train score is {DUSgrid_xgb.score(X_train,y_train)}')
```

```
Best parameters: {'xgboost__learning_rate': 0.1, 'xgboost__max_depth': 4,
'xgboost__min_child_weight': 7, 'xgboost__n_estimators': 50,
'xgboost__subsample': 0.6}
Best Mean cross-validation score: 0.59
Train score is 0.5963791267305645
```

3.4 Neural Network - MLP Classifier

[146]:
```
from sklearn.neural_network import MLPClassifier
```

[157]:
```
DUS_mlp = Pipeline([('random',RandomUnderSampler(random_state=42)),
                    ('mlp',MLPClassifier(random_state=42))])

parameters = {'mlp__solver': ['lbfgs'],
              'mlp__max_iter': [1200],
              'mlp__alpha': 10.0 ** -np.arange(1, 10),
              'mlp__hidden_layer_sizes':np.arange(10, 15),
              'mlp__random_state':range(2,12,2)}
MLPGrid = GridSearchCV(DUS_mlp, parameters,cv=5, n_jobs=-1,scoring=f2score)

MLPGrid.fit(X_train, y_train)
```

[157]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('random',
                                        RandomUnderSampler(random_state=42,
                                                           replacement=False,
sampling_strategy='auto')),
                                       ('mlp',
                                        MLPClassifier(activation='relu',
                                                      alpha=0.0001,
                                                      batch_size='auto',
                                                      beta_1=0.9, beta_2=0.999,
                                                      early_stopping=False,
                                                      epsilon=1e-08,
                                                      hidden_layer_sizes=(100,),
                                                      learning_rate='constant',
                                                      learning…
             iid='deprecated', n_jobs=-1,
             param_grid={'mlp__alpha': array([1.e-01, 1.e-02, 1.e-03, 1.e-04,
1.e-05, 1.e-06, 1.e-07, 1.e-08,
       1.e-09]),
                         'mlp__hidden_layer_sizes': array([10, 11, 12, 13, 14]),
                         'mlp__max_iter': [1200],
                         'mlp__random_state': range(2, 12, 2),
                         'mlp__solver': ['lbfgs']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

[359]:
```
print("Best parameters: {}".format(MLPGrid.best_params_))
print("Best Mean cross-validation score: {:.2f}".format(MLPGrid.best_score_))
print(f'Train score is {MLPGrid.score(X_train,y_train)}')
```

```
Best parameters: {'mlp__alpha': 0.1, 'mlp__hidden_layer_sizes': 10,
```

```
'mlp__max_iter': 1200, 'mlp__random_state': 2, 'mlp__solver': 'lbfgs'}
Best Mean cross-validation score: 0.43
Train score is 0.4450930317402408
```

3.5 Stacking

4 Summary

[195]:
```
classifiers={
'Logistic':grid_logit,
'DecisionTree':grid_dtree,
'Kernel SVC':grid_svc,
'RF':grid_rf,
'Extra Trees':etc_grid,
'Gradient Boost':gbc_grid,
'XGBoost':xgbc_grid,
'CS Logit':grid_CSlogreg,
'CS DTree':grid_CSdtree,
# 'CS RF':SCRandom_param,
'CS XGBoost':grid_xgboost,
'CS Extra Tree':grid_SCExTree,
'CS Bagging D Tree':grid_SCBagging,
'Over logit':grid_logitsmote,
'Over DTree':DOS_dtree,
'Over RF':DOSgrid_rf,
'Over Easy':DOSgrid_easy,
'Over XGboost':DOSgrid_xgb,
'Over MLP':DOSMLPGrid,
'Under Logit':grid_logitRS,
'Under Decision Tree': DUS_dtree,
'Under RF':DUSgrid_rf,
'Under Easy':DUSgrid_easy,
'Under XGboost':DUSgrid_xgb,
'Under MLP':MLPGrid
}
```

[196]:
```
classifiers.keys()
```

[196]:
```
dict_keys(['Logistic', 'DecisionTree', 'Kernel SVC', 'RF', 'Extra Trees',
'Gradient Boost', 'XGBoost', 'CS Logit', 'CS DTree', 'CS XGBoost', 'CS Extra
Tree', 'CS Bagging D Tree', 'Over logit', 'Over DTree', 'Over RF', 'Over Easy',
'Over XGboost', 'Over MLP', 'Under Logit', 'Under Decision Tree', 'Under RF',
'Under Easy', 'Under XGboost', 'Under MLP'])
```

[197]:
```
results_mean_std = []
for  key, value in classifiers.items():
    mean = value.cv_results_['mean_test_score'][value.best_index_]
```

```
    std=value.cv_results_['std_test_score'][value.best_index_]

    results_mean_std.append({
        "model": key,
        "mean": mean,
        "std": std
    })
```

[198]:
```
# Create a Pandas DataFrame with the mean+std results
accuracy_df = pd.DataFrame(results_mean_std, columns=['model', 'mean', 'std'])
```

[199]:
```
# Show the accuracy dataframe

accuracy_df.sort_values(by=['mean'], inplace=True,ascending=False)
accuracy_df
```

[199]:

	model	mean	std
9	CS XGBoost	0.863311	0.046246
14	Over RF	0.861917	0.034663
6	XGBoost	0.853696	0.048157
3	RF	0.850095	0.055296
10	CS Extra Tree	0.842084	0.049814
16	Over XGboost	0.838657	0.050082
4	Extra Trees	0.832794	0.053444
20	Under RF	0.821151	0.029584
1	DecisionTree	0.814990	0.056385
0	Logistic	0.800555	0.027454
7	CS Logit	0.800555	0.027454
5	Gradient Boost	0.799898	0.050013
8	CS DTree	0.781130	0.036795
13	Over DTree	0.735339	0.047498
2	Kernel SVC	0.728533	0.068554
12	Over logit	0.696800	0.038193
18	Under Logit	0.689065	0.037545
15	Over Easy	0.675208	0.025761
22	Under XGboost	0.587185	0.026722
19	Under Decision Tree	0.514595	0.051219
11	CS Bagging D Tree	0.471119	0.053869
21	Under Easy	0.461867	0.032179
17	Over MLP	0.427471	0.045042
23	Under MLP	0.427471	0.045042

[200]:
```
# Create a prediction of all models on the test set
predictions_all = {}
for  key, value in classifiers.items():
    # Get best estimator
    best_model = value.best_estimator_
```

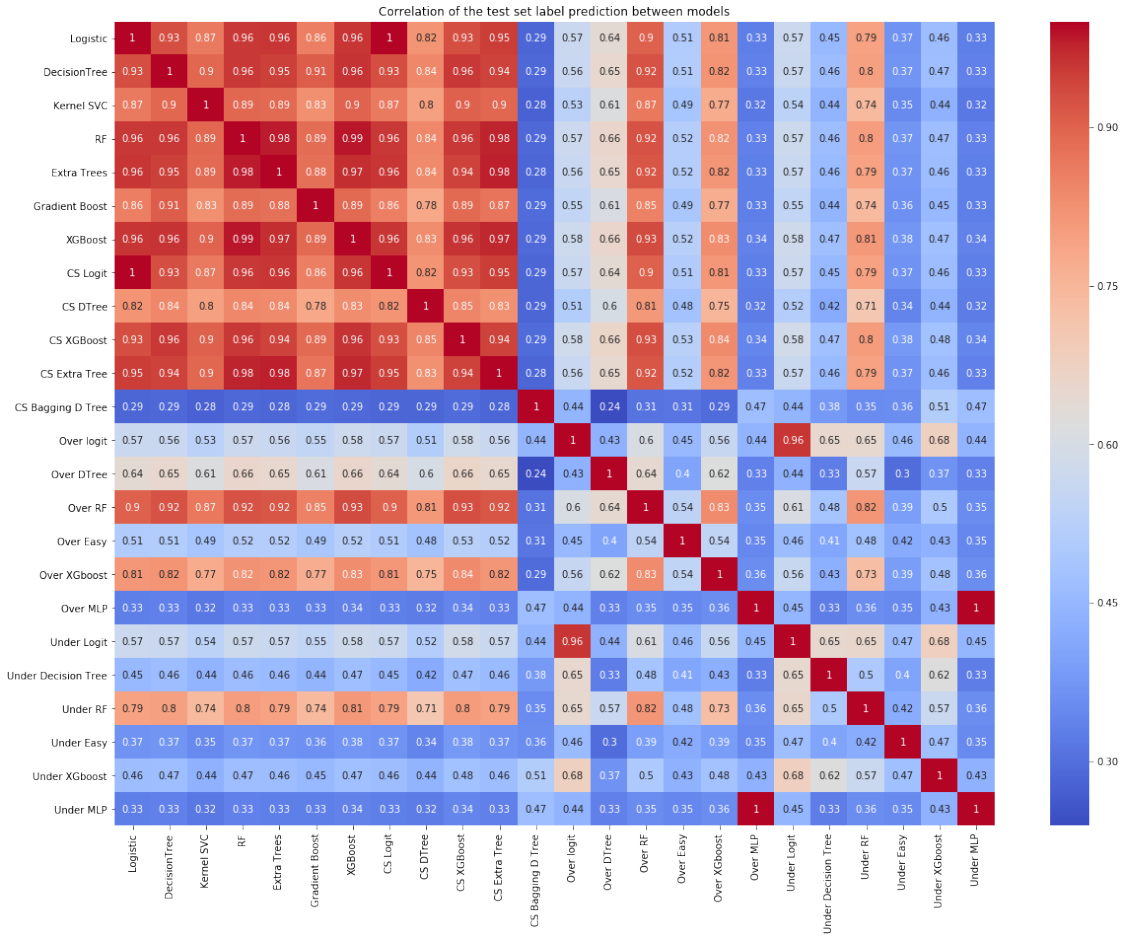
```
    # Predict test labels
    predictions = best_model.predict(X_test)

    # Save predictions to a list
    predictions_all[key] = predictions
```

[201]:
```
# Creat a DataFrame for the predictions
pred = pd.DataFrame(predictions_all)
```

[206]:
```
# Plot a heatmap of all correlations for easier visualization
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(20,15))
g = sns.heatmap(pred.corr(), annot=True, cmap='coolwarm', ax=ax)
g.set_title('Correlation of the test set label prediction between models')
```

[206]: Text(0.5, 1, 'Correlation of the test set label prediction between models')



[207]:
```
def get_redundant_pairs(df):
    '''Get diagonal and lower triangular pairs of correlation matrix'''
    pairs_to_drop = set()
    cols = df.columns
    for i in range(0, df.shape[1]):
        for j in range(0, i+1):
            pairs_to_drop.add((cols[i], cols[j]))
    return pairs_to_drop

def get_top_abs_correlations(df, n=5):
    au_corr = df.corr().abs().unstack()
    labels_to_drop = get_redundant_pairs(df)
    au_corr = au_corr.drop(labels=labels_to_drop).sort_values(ascending=True)
    return au_corr[0:n]
```

[209]:
```
print("Top Least Correlations")
print(get_top_abs_correlations(pred, 7))
```

```
Top Least Correlations
CS Bagging D Tree  Over DTree           0.238920
Kernel SVC         CS Bagging D Tree    0.276968
CS Extra Tree      CS Bagging D Tree    0.284768
Extra Trees        CS Bagging D Tree    0.284895
CS DTree           CS Bagging D Tree    0.285106
Logistic           CS Bagging D Tree    0.286314
CS Logit           CS Bagging D Tree    0.286314
dtype: float64
```

4.1 Voting top 5

[211]:
```
from sklearn.ensemble import VotingClassifier
```

[212]:
```
vclf1 = VotingClassifier(estimators=
                              [('CS XGBoost', grid_xgboost.best_estimator_),
                               ('Over RF', DOSgrid_rf.best_estimator_),
                               ('XGBoost', xgbc_grid.best_estimator_),
                               ('RF', grid_rf.best_estimator_),
                               ('CS Extra Tree', grid_SCExTree.best_estimator_),
                               ('Over XGboost', DOSgrid_xgb.best_estimator_),
                               ('Extra Trees', etc_grid.best_estimator_),
                              ] )
vclf1_param = {'voting' : ['hard','soft']}
vclf1_grid = GridSearchCV(vclf1, vclf1_param,cv=5,n_jobs=-1,␣
 ↪return_train_score=True,scoring=f2score)
vclf1_grid.fit(X_train,y_train)
```

[212]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=VotingClassifier(estimators=[('CS XGBoost',
XGBClassifier(base_score=0.5,
                                                                booster=None,
colsample_bylevel=1,
colsample_bynode=1,
colsample_bytree=1,
                                                                gamma=0,
                                                                gpu_id=-1,
importance_type='gain',
interaction_constraints=None,
learning_rate=0.300000012,
max_delta_step=0,
                                                                max_depth=6,
min_child_weight=1,
                                                                missing=nan,
                                                                monotone…
min_weight_fraction_leaf=0.0,
n_estimators=50,
n_jobs=None,
oob_score=False,
random_state=42,
verbose=0,
warm_start=False))],
                                        flatten_transform=True, n_jobs=None,
                                        voting='hard', weights=None),
             iid='deprecated', n_jobs=-1,
             param_grid={'voting': ['hard', 'soft']}, pre_dispatch='2*n_jobs',
             refit=True, return_train_score=True,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

4.1.1 Results

[213]:
```
print(f'Best Mean Cross Validation Score is {vclf1_grid.best_score_}')
print(f'Best Mean Cross Validation Score is {vclf1_grid.best_params_}')
print(f'Train score is {vclf1_grid.score(X_train,y_train)}')
```

```
Best Mean Cross Validation Score is 0.8599431114386332
Best Mean Cross Validation Score is {'voting': 'hard'}
Train score is 0.940347970173985
```

4.2 Stacking Top 5

https://towardsdatascience.com/stacking-made-easy-with-sklearn-e27a0793c92b

[217]:
```
from sklearn.ensemble import StackingClassifier
```

4.2.1 Combination 1 - .81352

[222]:
```
sclf1 = StackingClassifier(estimators=
                           [('CS XGBoost', grid_xgboost.best_estimator_),
                            ('Over RF', DOSgrid_rf.best_estimator_),
                            ('XGBoost', xgbc_grid.best_estimator_),
                            ('RF', grid_rf.best_estimator_),
                            ('CS Extra Tree', grid_SCExTree.best_estimator_),
                            ('Over XGboost', DOSgrid_xgb.best_estimator_),
                            ('Extra Trees', etc_grid.best_estimator_),
                           ] , final_estimator=LogisticRegression(
    multi_class="multinomial",solver="lbfgs", C=30))

sclf1_param = {
                'final_estimator__C': [0.001, 0.01, 0.1, 1, 10,1000],
                'final_estimator__penalty':['l2'],
                'stack_method':['auto', 'predict_proba']
             }
sclf1_grid = GridSearchCV(sclf1, sclf1_param,cv=5,␣
 ↪return_train_score=True,n_jobs=-1,scoring=f2score )
sclf1_grid.fit(X_train,y_train)
```

[222]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=StackingClassifier(cv=None,
                                          estimators=[('CS XGBoost',
XGBClassifier(base_score=0.5,
booster=None,
colsample_bylevel=1,
colsample_bynode=1,
colsample_bytree=1,
                                                                  gamma=0,
                                                                  gpu_id=-1,
importance_type='gain',
interaction_constraints=None,
learning_rate=0.300000012,
max_delta_step=0,
max_depth=6,
min_child_weight=1,
missing=na…
warm_start=False),
                                          n_jobs=None, passthrough=False,
                                          stack_method='auto', verbose=0),
             iid='deprecated', n_jobs=-1,
             param_grid={'final_estimator__C': [0.001, 0.01, 0.1, 1, 10, 1000],
                         'final_estimator__penalty': ['l2'],
                         'stack_method': ['auto', 'predict_proba']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
```

```
                scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

[223]:
```
print(f'Best Mean Cross Validation Score is {sclf1_grid.best_score_}')
print(f'Best Mean Cross Validation Score is {sclf1_grid.best_params_}')
print(f'Train score is {sclf1_grid.score(X_train,y_train)}')
```

```
Best Mean Cross Validation Score is 0.8478698044874516
Best Mean Cross Validation Score is {'final_estimator__C': 1,
'final_estimator__penalty': 'l2', 'stack_method': 'auto'}
Train score is 0.9835390946502058
```

4.2.2 Combination 2 -.81352

[225]:
```
sclf2 = StackingClassifier(estimators=
                           [('CS XGBoost', grid_xgboost.best_estimator_),
                            ('Over RF', DOSgrid_rf.best_estimator_),
                            ('XGBoost', xgbc_grid.best_estimator_),
                            ('RF', grid_rf.best_estimator_),
                            ('CS Extra Tree', grid_SCExTree.best_estimator_),
                            ('Over XGboost', DOSgrid_xgb.best_estimator_),
                            ('Extra Trees', etc_grid.best_estimator_),
                           ] , final_estimator=sclf1_grid.best_estimator_)

sclf2_param = {
#                 'final_estimator__C': [0.001, 0.01, 0.1, 1, 10,1000],
#                 'final_estimator__penalty':['l2'],
               'stack_method':['auto', 'predict_proba']
              }
sclf2_grid = GridSearchCV(sclf2, sclf2_param,cv=5,␣
 ↪return_train_score=True,scoring=f2score)
sclf2_grid.fit(X_train,y_train)
```

[225]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=StackingClassifier(cv=None,
                                          estimators=[('CS XGBoost',
XGBClassifier(base_score=0.5,
booster=None,
colsample_bylevel=1,
colsample_bynode=1,
colsample_bytree=1,
                                                                  gamma=0,
                                                                  gpu_id=-1,

importance_type='gain',
interaction_constraints=None,
learning_rate=0.300000012,
max_delta_step=0,
max_depth=6,
```

```
min_child_weight=1,
missing=na…
                                  tol=0.0001,
                                  verbose=0,
                                  warm_start=False),
n_jobs=None,
passthrough=False,
stack_method='auto',
verbose=0),
                                             n_jobs=None, passthrough=False,
                                             stack_method='auto', verbose=0),
             iid='deprecated', n_jobs=None,
             param_grid={'stack_method': ['auto', 'predict_proba']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

[226]:
```
print(f'Best Mean Cross Validation Score is {sclf2_grid.best_score_}')
print(f'Best Mean Cross Validation Score is {sclf2_grid.best_params_}')
print(f'Train score is {sclf2_grid.score(X_train,y_train)}')
```

```
Best Mean Cross Validation Score is 0.8595398428731761
Best Mean Cross Validation Score is {'stack_method': 'auto'}
Train score is 0.9535655058043118
```

4.2.3 Combination 3 - .81204

[227]:
```
sclf3 = StackingClassifier(estimators=
                           [('CS XGBoost', grid_xgboost.best_estimator_),
                            ('Over RF', DOSgrid_rf.best_estimator_),
                            ('XGBoost', xgbc_grid.best_estimator_),
                            ('RF', grid_rf.best_estimator_),
                            ('CS Extra Tree', grid_SCExTree.best_estimator_),
                            ('Over XGboost', DOSgrid_xgb.best_estimator_),
                            ('Extra Trees', etc_grid.best_estimator_),
                           ] , final_estimator=sclf2_grid.best_estimator_)

sclf3_param = {
#                 'final_estimator__C': [0.001, 0.01, 0.1, 1, 10,1000],
#                 'final_estimator__penalty':['l2'],
               'stack_method':['auto', 'predict_proba']
             }
sclf3_grid = GridSearchCV(sclf3, sclf3_param,cv=5, return_train_score=True )
sclf3_grid.fit(X_train,y_train)
```

[227]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=StackingClassifier(cv=None,
                                          estimators=[('CS XGBoost',
```

```
XGBClassifier(base_score=0.5,
booster=None,
colsample_bylevel=1,
colsample_bynode=1,
colsample_bytree=1,
                                                                    gamma=0,
                                                                    gpu_id=-1,
importance_type='gain',
interaction_constraints=None,
learning_rate=0.300000012,
max_delta_step=0,
max_depth=6,
min_child_weight=1,
missing=na…
warm_start=False),
                                  n_jobs=None,
                                  passthrough=False,
                                  stack_method='auto',
                                  verbose=0),
n_jobs=None,
passthrough=False,
stack_method='auto',
verbose=0),
                                             n_jobs=None, passthrough=False,
                                             stack_method='auto', verbose=0),
             iid='deprecated', n_jobs=None,
             param_grid={'stack_method': ['auto', 'predict_proba']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

[382]:
```
print(f'Best Mean Cross Validation Score is {sclf3_grid.best_score_}')
print(f'Best Parameters are {sclf3_grid.best_params_}')
print(f'Train score is {sclf2_grid.score(X_train,y_train)}')
```

```
Best Mean Cross Validation Score is 0.9978266297321057
Best Parameters are {'stack_method': 'auto'}
Train score is 0.9535655058043118
```

4.3 Other Methods

4.3.1 RandomizedSearchCV with Random Forest

[234]:
```
from sklearn.model_selection import RandomizedSearchCV
```

[235]:
```
#random forest
from sklearn.ensemble import RandomForestClassifier
rfc =RandomForestClassifier(random_state=0)
rfc_param = {
```

```
    'n_estimators': range(2,15,2),
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : range(2,15,3),
    'criterion' :['gini', 'entropy']
}

rsearchcv_rf = RandomizedSearchCV(rfc, rfc_param,cv=5, return_train_score=True,␣
 ↪scoring =f2score)
rsearchcv_rf.fit(X_train,y_train)
```

[235]:
```
RandomizedSearchCV(cv=5, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
min_weight_fraction_leaf=0.0,
                                                    n_estimators=100,
                                                    n_jobs…
                                                    random_state=0, verbose=0,
                                                    warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'criterion': ['gini', 'entropy'],
                                        'max_depth': range(2, 15, 3),
                                        'max_features': ['auto', 'sqrt',
                                                         'log2'],
                                        'n_estimators': range(2, 15, 2)},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True,
                   scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

Results

[236]:
```
print(f'Best Mean Cross Validation Score is {rsearchcv_rf.best_params_}')
print(f'Best Mean Cross Validation Score is {rsearchcv_rf.best_score_}')
print(f'Train score is {rsearchcv_rf.score(X_train,y_train)}')
```

```
Best Mean Cross Validation Score is {'n_estimators': 12, 'max_features': 'auto',
'max_depth': 8, 'criterion': 'gini'}
Best Mean Cross Validation Score is 0.8415861898736801
Train score is 0.871404399323181
```

4.3.2 CatBoost Classifier - .81352

[239]:
```
#pip install catboost
```

[243]:
```
from catboost import CatBoostClassifier
```

[308]:
```
catb = CatBoostClassifier()

catb_param = {'learning_rate': [0.03, 0.1],
              'depth': range(2,10,2),
              'l2_leaf_reg': range(2,10,2)}

catb_grid = GridSearchCV(catb, catb_param,cv=5,n_jobs=-1,␣
 ↪return_train_score=True, scoring =f2score)
catb_grid.fit(X_train,y_train)
```

```
0:      learn: 0.6074288        total: 74.3ms   remaining: 1m 14s
1:      learn: 0.5308762        total: 88ms     remaining: 43.9s
2:      learn: 0.4647003        total: 98.8ms   remaining: 32.8s
3:      learn: 0.4103385        total: 110ms    remaining: 27.4s
4:      learn: 0.3589659        total: 122ms    remaining: 24.3s
5:      learn: 0.3156362        total: 133ms    remaining: 22.1s
6:      learn: 0.2759750        total: 146ms    remaining: 20.6s
7:      learn: 0.2440715        total: 159ms    remaining: 19.7s
8:      learn: 0.2144962        total: 171ms    remaining: 18.8s
9:      learn: 0.1885638        total: 182ms    remaining: 18s
10:     learn: 0.1670417        total: 196ms    remaining: 17.6s
11:     learn: 0.1476058        total: 208ms    remaining: 17.1s
12:     learn: 0.1331938        total: 220ms    remaining: 16.7s
13:     learn: 0.1190247        total: 233ms    remaining: 16.4s
14:     learn: 0.1061889        total: 245ms    remaining: 16.1s
15:     learn: 0.0950364        total: 259ms    remaining: 15.9s
16:     learn: 0.0854880        total: 273ms    remaining: 15.8s
17:     learn: 0.0770394        total: 286ms    remaining: 15.6s
18:     learn: 0.0698292        total: 297ms    remaining: 15.3s
19:     learn: 0.0635772        total: 309ms    remaining: 15.1s
20:     learn: 0.0577759        total: 322ms    remaining: 15s
21:     learn: 0.0528676        total: 335ms    remaining: 14.9s
22:     learn: 0.0485986        total: 347ms    remaining: 14.8s
23:     learn: 0.0449189        total: 364ms    remaining: 14.8s
24:     learn: 0.0415947        total: 375ms    remaining: 14.6s
25:     learn: 0.0387336        total: 388ms    remaining: 14.5s
26:     learn: 0.0361942        total: 400ms    remaining: 14.4s
27:     learn: 0.0339457        total: 411ms    remaining: 14.3s
28:     learn: 0.0318975        total: 422ms    remaining: 14.1s
29:     learn: 0.0300825        total: 433ms    remaining: 14s
30:     learn: 0.0282923        total: 446ms    remaining: 13.9s
31:     learn: 0.0268204        total: 461ms    remaining: 13.9s
```

```
32:     learn: 0.0255583        total: 474ms    remaining: 13.9s
33:     learn: 0.0243798        total: 490ms    remaining: 13.9s
34:     learn: 0.0233485        total: 504ms    remaining: 13.9s
35:     learn: 0.0224139        total: 516ms    remaining: 13.8s
36:     learn: 0.0215928        total: 529ms    remaining: 13.8s
37:     learn: 0.0208351        total: 541ms    remaining: 13.7s
38:     learn: 0.0201242        total: 553ms    remaining: 13.6s
39:     learn: 0.0195033        total: 565ms    remaining: 13.6s
40:     learn: 0.0189460        total: 577ms    remaining: 13.5s
41:     learn: 0.0182528        total: 590ms    remaining: 13.4s
42:     learn: 0.0177152        total: 603ms    remaining: 13.4s
43:     learn: 0.0172876        total: 616ms    remaining: 13.4s
44:     learn: 0.0168570        total: 629ms    remaining: 13.4s
45:     learn: 0.0164760        total: 644ms    remaining: 13.4s
46:     learn: 0.0161703        total: 656ms    remaining: 13.3s
47:     learn: 0.0158765        total: 668ms    remaining: 13.2s
48:     learn: 0.0155867        total: 680ms    remaining: 13.2s
49:     learn: 0.0152391        total: 692ms    remaining: 13.1s
50:     learn: 0.0150067        total: 704ms    remaining: 13.1s
51:     learn: 0.0148255        total: 716ms    remaining: 13.1s
52:     learn: 0.0145782        total: 727ms    remaining: 13s
53:     learn: 0.0143297        total: 739ms    remaining: 12.9s
54:     learn: 0.0141875        total: 751ms    remaining: 12.9s
55:     learn: 0.0139818        total: 763ms    remaining: 12.9s
56:     learn: 0.0137251        total: 774ms    remaining: 12.8s
57:     learn: 0.0135430        total: 786ms    remaining: 12.8s
58:     learn: 0.0133772        total: 799ms    remaining: 12.7s
59:     learn: 0.0132640        total: 811ms    remaining: 12.7s
60:     learn: 0.0131530        total: 824ms    remaining: 12.7s
61:     learn: 0.0130523        total: 837ms    remaining: 12.7s
62:     learn: 0.0129569        total: 1.18s    remaining: 17.6s
63:     learn: 0.0128589        total: 1.19s    remaining: 17.5s
64:     learn: 0.0126957        total: 1.21s    remaining: 17.4s
65:     learn: 0.0125399        total: 1.22s    remaining: 17.2s
66:     learn: 0.0124000        total: 1.23s    remaining: 17.1s
67:     learn: 0.0122751        total: 1.24s    remaining: 17s
68:     learn: 0.0121453        total: 1.25s    remaining: 16.9s
69:     learn: 0.0120447        total: 1.27s    remaining: 16.8s
70:     learn: 0.0119707        total: 1.28s    remaining: 16.7s
71:     learn: 0.0119007        total: 1.29s    remaining: 16.6s
72:     learn: 0.0118277        total: 1.3s     remaining: 16.5s
73:     learn: 0.0117382        total: 1.31s    remaining: 16.4s
74:     learn: 0.0116948        total: 1.33s    remaining: 16.4s
75:     learn: 0.0116481        total: 1.34s    remaining: 16.3s
76:     learn: 0.0116015        total: 1.35s    remaining: 16.2s
77:     learn: 0.0115148        total: 1.36s    remaining: 16.1s
78:     learn: 0.0114770        total: 1.38s    remaining: 16.1s
79:     learn: 0.0114308        total: 1.39s    remaining: 16s
```

```
80:     learn: 0.0113522        total: 1.4s     remaining: 15.9s
81:     learn: 0.0112771        total: 1.42s    remaining: 15.9s
82:     learn: 0.0112396        total: 1.43s    remaining: 15.8s
83:     learn: 0.0111904        total: 1.44s    remaining: 15.7s
84:     learn: 0.0111437        total: 1.45s    remaining: 15.6s
85:     learn: 0.0110683        total: 1.46s    remaining: 15.5s
86:     learn: 0.0110223        total: 1.47s    remaining: 15.5s
87:     learn: 0.0109570        total: 1.49s    remaining: 15.4s
88:     learn: 0.0108920        total: 1.5s     remaining: 15.3s
89:     learn: 0.0108630        total: 1.51s    remaining: 15.3s
90:     learn: 0.0108245        total: 1.52s    remaining: 15.2s
91:     learn: 0.0107678        total: 1.53s    remaining: 15.1s
92:     learn: 0.0106960        total: 1.55s    remaining: 15.1s
93:     learn: 0.0106408        total: 1.56s    remaining: 15s
94:     learn: 0.0105965        total: 1.57s    remaining: 15s
95:     learn: 0.0105478        total: 1.58s    remaining: 14.9s
96:     learn: 0.0105234        total: 1.6s     remaining: 14.9s
97:     learn: 0.0104791        total: 1.61s    remaining: 14.8s
98:     learn: 0.0104317        total: 1.62s    remaining: 14.7s
99:     learn: 0.0104177        total: 1.63s    remaining: 14.7s
100:    learn: 0.0103786        total: 1.64s    remaining: 14.6s
101:    learn: 0.0103596        total: 1.65s    remaining: 14.6s
102:    learn: 0.0103284        total: 1.66s    remaining: 14.5s
103:    learn: 0.0103166        total: 1.68s    remaining: 14.4s
104:    learn: 0.0102865        total: 1.69s    remaining: 14.4s
105:    learn: 0.0102276        total: 1.7s     remaining: 14.3s
106:    learn: 0.0101953        total: 1.71s    remaining: 14.3s
107:    learn: 0.0101559        total: 1.73s    remaining: 14.3s
108:    learn: 0.0101250        total: 1.74s    remaining: 14.2s
109:    learn: 0.0100993        total: 1.75s    remaining: 14.2s
110:    learn: 0.0100575        total: 1.76s    remaining: 14.1s
111:    learn: 0.0100238        total: 1.77s    remaining: 14.1s
112:    learn: 0.0099926        total: 1.79s    remaining: 14s
113:    learn: 0.0099663        total: 1.8s     remaining: 14s
114:    learn: 0.0099471        total: 1.81s    remaining: 13.9s
115:    learn: 0.0099176        total: 1.82s    remaining: 13.9s
116:    learn: 0.0099030        total: 1.84s    remaining: 13.9s
117:    learn: 0.0098836        total: 1.85s    remaining: 13.8s
118:    learn: 0.0098587        total: 1.86s    remaining: 13.8s
119:    learn: 0.0098566        total: 1.87s    remaining: 13.7s
120:    learn: 0.0098288        total: 1.88s    remaining: 13.7s
121:    learn: 0.0098034        total: 1.89s    remaining: 13.6s
122:    learn: 0.0097571        total: 1.91s    remaining: 13.6s
123:    learn: 0.0097284        total: 1.92s    remaining: 13.6s
124:    learn: 0.0097179        total: 1.93s    remaining: 13.5s
125:    learn: 0.0097039        total: 1.94s    remaining: 13.5s
126:    learn: 0.0096849        total: 1.95s    remaining: 13.4s
127:    learn: 0.0096576        total: 1.97s    remaining: 13.4s
```

```
128:    learn: 0.0096399        total: 1.98s    remaining: 13.4s
129:    learn: 0.0096171        total: 1.99s    remaining: 13.3s
130:    learn: 0.0096149        total: 2s       remaining: 13.3s
131:    learn: 0.0095878        total: 2.02s    remaining: 13.3s
132:    learn: 0.0095664        total: 2.03s    remaining: 13.2s
133:    learn: 0.0095468        total: 2.04s    remaining: 13.2s
134:    learn: 0.0095134        total: 2.05s    remaining: 13.1s
135:    learn: 0.0094953        total: 2.06s    remaining: 13.1s
136:    learn: 0.0094830        total: 2.07s    remaining: 13.1s
137:    learn: 0.0094601        total: 2.08s    remaining: 13s
138:    learn: 0.0094482        total: 2.1s     remaining: 13s
139:    learn: 0.0094423        total: 2.11s    remaining: 13s
140:    learn: 0.0094401        total: 2.12s    remaining: 12.9s
141:    learn: 0.0094119        total: 2.13s    remaining: 12.9s
142:    learn: 0.0093963        total: 2.15s    remaining: 12.9s
143:    learn: 0.0093849        total: 2.16s    remaining: 12.8s
144:    learn: 0.0093830        total: 2.17s    remaining: 12.8s
145:    learn: 0.0093510        total: 2.18s    remaining: 12.8s
146:    learn: 0.0093376        total: 2.19s    remaining: 12.7s
147:    learn: 0.0093331        total: 2.2s     remaining: 12.7s
148:    learn: 0.0093199        total: 2.22s    remaining: 12.7s
149:    learn: 0.0093172        total: 2.23s    remaining: 12.6s
150:    learn: 0.0092960        total: 2.24s    remaining: 12.6s
151:    learn: 0.0092748        total: 2.25s    remaining: 12.6s
152:    learn: 0.0092636        total: 2.26s    remaining: 12.5s
153:    learn: 0.0092411        total: 2.27s    remaining: 12.5s
154:    learn: 0.0092309        total: 2.29s    remaining: 12.5s
155:    learn: 0.0092167        total: 2.3s     remaining: 12.4s
156:    learn: 0.0091942        total: 2.32s    remaining: 12.4s
157:    learn: 0.0091918        total: 2.33s    remaining: 12.4s
158:    learn: 0.0091694        total: 2.34s    remaining: 12.4s
159:    learn: 0.0091593        total: 2.35s    remaining: 12.3s
160:    learn: 0.0091515        total: 2.36s    remaining: 12.3s
161:    learn: 0.0091335        total: 2.37s    remaining: 12.3s
162:    learn: 0.0090960        total: 2.38s    remaining: 12.2s
163:    learn: 0.0090865        total: 2.4s     remaining: 12.2s
164:    learn: 0.0090712        total: 2.41s    remaining: 12.2s
165:    learn: 0.0090689        total: 2.42s    remaining: 12.2s
166:    learn: 0.0090331        total: 2.43s    remaining: 12.1s
167:    learn: 0.0090159        total: 2.44s    remaining: 12.1s
168:    learn: 0.0090040        total: 2.46s    remaining: 12.1s
169:    learn: 0.0089861        total: 2.47s    remaining: 12.1s
170:    learn: 0.0089605        total: 2.48s    remaining: 12s
171:    learn: 0.0089486        total: 2.5s     remaining: 12s
172:    learn: 0.0089447        total: 2.51s    remaining: 12s
173:    learn: 0.0089322        total: 2.52s    remaining: 12s
174:    learn: 0.0089241        total: 2.54s    remaining: 12s
175:    learn: 0.0089140        total: 2.55s    remaining: 11.9s
```

```
176:    learn: 0.0089126        total: 2.56s    remaining: 11.9s
177:    learn: 0.0088983        total: 2.58s    remaining: 11.9s
178:    learn: 0.0088824        total: 2.59s    remaining: 11.9s
179:    learn: 0.0088689        total: 2.6s     remaining: 11.8s
180:    learn: 0.0088522        total: 2.61s    remaining: 11.8s
181:    learn: 0.0088281        total: 2.63s    remaining: 11.8s
182:    learn: 0.0088197        total: 2.64s    remaining: 11.8s
183:    learn: 0.0087891        total: 2.65s    remaining: 11.8s
184:    learn: 0.0087813        total: 2.67s    remaining: 11.7s
185:    learn: 0.0087796        total: 2.68s    remaining: 11.7s
186:    learn: 0.0087594        total: 2.69s    remaining: 11.7s
187:    learn: 0.0087363        total: 2.7s     remaining: 11.7s
188:    learn: 0.0087138        total: 2.72s    remaining: 11.7s
189:    learn: 0.0087033        total: 2.73s    remaining: 11.6s
190:    learn: 0.0086909        total: 2.74s    remaining: 11.6s
191:    learn: 0.0086831        total: 2.75s    remaining: 11.6s
192:    learn: 0.0086796        total: 2.76s    remaining: 11.6s
193:    learn: 0.0086758        total: 2.78s    remaining: 11.5s
194:    learn: 0.0086676        total: 2.79s    remaining: 11.5s
195:    learn: 0.0086558        total: 2.8s     remaining: 11.5s
196:    learn: 0.0086457        total: 2.81s    remaining: 11.5s
197:    learn: 0.0086353        total: 2.83s    remaining: 11.5s
198:    learn: 0.0086154        total: 2.85s    remaining: 11.5s
199:    learn: 0.0085960        total: 2.86s    remaining: 11.5s
200:    learn: 0.0085940        total: 2.88s    remaining: 11.4s
201:    learn: 0.0085924        total: 2.89s    remaining: 11.4s
202:    learn: 0.0085859        total: 2.9s     remaining: 11.4s
203:    learn: 0.0085711        total: 2.91s    remaining: 11.4s
204:    learn: 0.0085569        total: 2.92s    remaining: 11.3s
205:    learn: 0.0085525        total: 2.94s    remaining: 11.3s
206:    learn: 0.0085327        total: 2.95s    remaining: 11.3s
207:    learn: 0.0085221        total: 2.96s    remaining: 11.3s
208:    learn: 0.0085071        total: 2.98s    remaining: 11.3s
209:    learn: 0.0085038        total: 2.99s    remaining: 11.2s
210:    learn: 0.0084978        total: 3s       remaining: 11.2s
211:    learn: 0.0084891        total: 3.01s    remaining: 11.2s
212:    learn: 0.0084724        total: 3.02s    remaining: 11.2s
213:    learn: 0.0084635        total: 3.04s    remaining: 11.2s
214:    learn: 0.0084551        total: 3.05s    remaining: 11.1s
215:    learn: 0.0084476        total: 3.06s    remaining: 11.1s
216:    learn: 0.0084398        total: 3.07s    remaining: 11.1s
217:    learn: 0.0084308        total: 3.08s    remaining: 11.1s
218:    learn: 0.0084178        total: 3.1s     remaining: 11.1s
219:    learn: 0.0084159        total: 3.11s    remaining: 11s
220:    learn: 0.0084010        total: 3.12s    remaining: 11s
221:    learn: 0.0083927        total: 3.13s    remaining: 11s
222:    learn: 0.0083881        total: 3.15s    remaining: 11s
223:    learn: 0.0083672        total: 3.16s    remaining: 11s
```

```
224:    learn: 0.0083568        total: 3.18s    remaining: 10.9s
225:    learn: 0.0083512        total: 3.19s    remaining: 10.9s
226:    learn: 0.0083482        total: 3.2s     remaining: 10.9s
227:    learn: 0.0083350        total: 3.22s    remaining: 10.9s
228:    learn: 0.0083301        total: 3.23s    remaining: 10.9s
229:    learn: 0.0083051        total: 3.24s    remaining: 10.8s
230:    learn: 0.0082948        total: 3.25s    remaining: 10.8s
231:    learn: 0.0082875        total: 3.27s    remaining: 10.8s
232:    learn: 0.0082712        total: 3.28s    remaining: 10.8s
233:    learn: 0.0082668        total: 3.29s    remaining: 10.8s
234:    learn: 0.0082515        total: 3.3s     remaining: 10.7s
235:    learn: 0.0082294        total: 3.31s    remaining: 10.7s
236:    learn: 0.0082167        total: 3.33s    remaining: 10.7s
237:    learn: 0.0082069        total: 3.34s    remaining: 10.7s
238:    learn: 0.0081762        total: 3.35s    remaining: 10.7s
239:    learn: 0.0081611        total: 3.36s    remaining: 10.6s
240:    learn: 0.0081485        total: 3.37s    remaining: 10.6s
241:    learn: 0.0081409        total: 3.39s    remaining: 10.6s
242:    learn: 0.0081302        total: 3.4s     remaining: 10.6s
243:    learn: 0.0080969        total: 3.41s    remaining: 10.6s
244:    learn: 0.0080939        total: 3.42s    remaining: 10.5s
245:    learn: 0.0080789        total: 3.43s    remaining: 10.5s
246:    learn: 0.0080545        total: 3.44s    remaining: 10.5s
247:    learn: 0.0080337        total: 3.46s    remaining: 10.5s
248:    learn: 0.0080285        total: 3.47s    remaining: 10.5s
249:    learn: 0.0080108        total: 3.48s    remaining: 10.4s
250:    learn: 0.0079902        total: 3.49s    remaining: 10.4s
251:    learn: 0.0079786        total: 3.5s     remaining: 10.4s
252:    learn: 0.0079688        total: 3.51s    remaining: 10.4s
253:    learn: 0.0079627        total: 3.52s    remaining: 10.4s
254:    learn: 0.0079594        total: 3.54s    remaining: 10.3s
255:    learn: 0.0079555        total: 3.55s    remaining: 10.3s
256:    learn: 0.0079531        total: 3.56s    remaining: 10.3s
257:    learn: 0.0079377        total: 3.57s    remaining: 10.3s
258:    learn: 0.0079306        total: 3.59s    remaining: 10.3s
259:    learn: 0.0078985        total: 3.6s     remaining: 10.2s
260:    learn: 0.0078890        total: 3.61s    remaining: 10.2s
261:    learn: 0.0078833        total: 3.62s    remaining: 10.2s
262:    learn: 0.0078820        total: 3.64s    remaining: 10.2s
263:    learn: 0.0078695        total: 3.65s    remaining: 10.2s
264:    learn: 0.0078604        total: 3.66s    remaining: 10.2s
265:    learn: 0.0078554        total: 3.67s    remaining: 10.1s
266:    learn: 0.0078500        total: 3.68s    remaining: 10.1s
267:    learn: 0.0078415        total: 3.7s     remaining: 10.1s
268:    learn: 0.0078235        total: 3.71s    remaining: 10.1s
269:    learn: 0.0078173        total: 3.72s    remaining: 10.1s
270:    learn: 0.0078102        total: 3.74s    remaining: 10.1s
271:    learn: 0.0077923        total: 3.75s    remaining: 10s
```

```
272:    learn: 0.0077823        total: 3.76s    remaining: 10s
273:    learn: 0.0077589        total: 3.77s    remaining: 10s
274:    learn: 0.0077535        total: 3.79s    remaining: 9.98s
275:    learn: 0.0077488        total: 3.8s     remaining: 9.96s
276:    learn: 0.0077324        total: 3.81s    remaining: 9.95s
277:    learn: 0.0077154        total: 3.82s    remaining: 9.93s
278:    learn: 0.0077028        total: 3.85s    remaining: 9.94s
279:    learn: 0.0076991        total: 3.86s    remaining: 9.92s
280:    learn: 0.0076967        total: 3.87s    remaining: 9.9s
281:    learn: 0.0076937        total: 3.88s    remaining: 9.88s
282:    learn: 0.0076839        total: 3.89s    remaining: 9.86s
283:    learn: 0.0076780        total: 3.91s    remaining: 9.86s
284:    learn: 0.0076676        total: 3.92s    remaining: 9.85s
285:    learn: 0.0076652        total: 3.94s    remaining: 9.83s
286:    learn: 0.0076604        total: 3.95s    remaining: 9.81s
287:    learn: 0.0076587        total: 3.96s    remaining: 9.79s
288:    learn: 0.0076537        total: 3.97s    remaining: 9.77s
289:    learn: 0.0076512        total: 3.98s    remaining: 9.76s
290:    learn: 0.0076212        total: 4s       remaining: 9.74s
291:    learn: 0.0076102        total: 4.01s    remaining: 9.72s
292:    learn: 0.0076046        total: 4.02s    remaining: 9.7s
293:    learn: 0.0075898        total: 4.03s    remaining: 9.69s
294:    learn: 0.0075876        total: 4.05s    remaining: 9.67s
295:    learn: 0.0075798        total: 4.06s    remaining: 9.65s
296:    learn: 0.0075728        total: 4.07s    remaining: 9.64s
297:    learn: 0.0075545        total: 4.1s     remaining: 9.65s
298:    learn: 0.0075443        total: 4.11s    remaining: 9.63s
299:    learn: 0.0075409        total: 4.12s    remaining: 9.61s
300:    learn: 0.0075304        total: 4.13s    remaining: 9.59s
301:    learn: 0.0075129        total: 4.14s    remaining: 9.57s
302:    learn: 0.0074973        total: 4.15s    remaining: 9.55s
303:    learn: 0.0074830        total: 4.17s    remaining: 9.54s
304:    learn: 0.0074810        total: 4.18s    remaining: 9.52s
305:    learn: 0.0074606        total: 4.19s    remaining: 9.51s
306:    learn: 0.0074567        total: 4.2s     remaining: 9.48s
307:    learn: 0.0074453        total: 4.21s    remaining: 9.46s
308:    learn: 0.0074216        total: 4.22s    remaining: 9.45s
309:    learn: 0.0074080        total: 4.24s    remaining: 9.44s
310:    learn: 0.0074010        total: 4.25s    remaining: 9.42s
311:    learn: 0.0073730        total: 4.27s    remaining: 9.41s
312:    learn: 0.0073624        total: 4.28s    remaining: 9.39s
313:    learn: 0.0073583        total: 4.29s    remaining: 9.37s
314:    learn: 0.0073540        total: 4.3s     remaining: 9.35s
315:    learn: 0.0073436        total: 4.31s    remaining: 9.33s
316:    learn: 0.0073408        total: 4.33s    remaining: 9.32s
317:    learn: 0.0073362        total: 4.34s    remaining: 9.3s
318:    learn: 0.0073288        total: 4.35s    remaining: 9.28s
319:    learn: 0.0073226        total: 4.36s    remaining: 9.27s
```

```
320:    learn: 0.0073187        total: 4.37s    remaining: 9.25s
321:    learn: 0.0073140        total: 4.38s    remaining: 9.23s
322:    learn: 0.0073116        total: 4.4s     remaining: 9.22s
323:    learn: 0.0073033        total: 4.41s    remaining: 9.21s
324:    learn: 0.0073003        total: 4.42s    remaining: 9.19s
325:    learn: 0.0072903        total: 4.44s    remaining: 9.17s
326:    learn: 0.0072885        total: 4.45s    remaining: 9.16s
327:    learn: 0.0072809        total: 4.47s    remaining: 9.15s
328:    learn: 0.0072781        total: 4.48s    remaining: 9.13s
329:    learn: 0.0072611        total: 4.49s    remaining: 9.12s
330:    learn: 0.0072475        total: 4.5s     remaining: 9.1s
331:    learn: 0.0072246        total: 4.52s    remaining: 9.09s
332:    learn: 0.0072221        total: 4.53s    remaining: 9.07s
333:    learn: 0.0072167        total: 4.54s    remaining: 9.05s
334:    learn: 0.0071953        total: 4.55s    remaining: 9.04s
335:    learn: 0.0071836        total: 4.58s    remaining: 9.05s
336:    learn: 0.0071741        total: 4.59s    remaining: 9.03s
337:    learn: 0.0071670        total: 4.6s     remaining: 9.01s
338:    learn: 0.0071651        total: 4.62s    remaining: 9s
339:    learn: 0.0071602        total: 4.63s    remaining: 8.99s
340:    learn: 0.0071519        total: 4.65s    remaining: 8.98s
341:    learn: 0.0071455        total: 4.66s    remaining: 8.97s
342:    learn: 0.0071431        total: 4.67s    remaining: 8.95s
343:    learn: 0.0071150        total: 4.69s    remaining: 8.95s
344:    learn: 0.0071076        total: 4.7s     remaining: 8.93s
345:    learn: 0.0070949        total: 4.71s    remaining: 8.91s
346:    learn: 0.0070879        total: 4.73s    remaining: 8.9s
347:    learn: 0.0070675        total: 4.74s    remaining: 8.88s
348:    learn: 0.0070584        total: 4.75s    remaining: 8.87s
349:    learn: 0.0070539        total: 4.77s    remaining: 8.85s
350:    learn: 0.0070492        total: 4.78s    remaining: 8.84s
351:    learn: 0.0070428        total: 4.79s    remaining: 8.83s
352:    learn: 0.0070359        total: 4.81s    remaining: 8.81s
353:    learn: 0.0070294        total: 4.82s    remaining: 8.79s
354:    learn: 0.0070205        total: 4.84s    remaining: 8.79s
355:    learn: 0.0070182        total: 4.85s    remaining: 8.77s
356:    learn: 0.0069984        total: 4.86s    remaining: 8.76s
357:    learn: 0.0069955        total: 4.87s    remaining: 8.74s
358:    learn: 0.0069871        total: 4.88s    remaining: 8.72s
359:    learn: 0.0069852        total: 4.9s     remaining: 8.71s
360:    learn: 0.0069732        total: 4.91s    remaining: 8.69s
361:    learn: 0.0069688        total: 4.92s    remaining: 8.68s
362:    learn: 0.0069649        total: 4.94s    remaining: 8.66s
363:    learn: 0.0069566        total: 4.95s    remaining: 8.65s
364:    learn: 0.0069529        total: 4.96s    remaining: 8.63s
365:    learn: 0.0069475        total: 4.97s    remaining: 8.62s
366:    learn: 0.0069442        total: 4.99s    remaining: 8.6s
367:    learn: 0.0069227        total: 5s       remaining: 8.59s
```

```
368:    learn: 0.0069129        total: 5.01s    remaining: 8.58s
369:    learn: 0.0068947        total: 5.03s    remaining: 8.56s
370:    learn: 0.0068769        total: 5.04s    remaining: 8.54s
371:    learn: 0.0068705        total: 5.05s    remaining: 8.53s
372:    learn: 0.0068624        total: 5.07s    remaining: 8.52s
373:    learn: 0.0068584        total: 5.08s    remaining: 8.5s
374:    learn: 0.0068530        total: 5.09s    remaining: 8.49s
375:    learn: 0.0068508        total: 5.12s    remaining: 8.49s
376:    learn: 0.0068324        total: 5.13s    remaining: 8.47s
377:    learn: 0.0068240        total: 5.14s    remaining: 8.46s
378:    learn: 0.0068066        total: 5.15s    remaining: 8.45s
379:    learn: 0.0068040        total: 5.17s    remaining: 8.43s
380:    learn: 0.0067965        total: 5.18s    remaining: 8.42s
381:    learn: 0.0067800        total: 5.2s     remaining: 8.41s
382:    learn: 0.0067653        total: 5.21s    remaining: 8.39s
383:    learn: 0.0067494        total: 5.22s    remaining: 8.38s
384:    learn: 0.0067429        total: 5.24s    remaining: 8.37s
385:    learn: 0.0067271        total: 5.25s    remaining: 8.36s
386:    learn: 0.0067251        total: 5.26s    remaining: 8.34s
387:    learn: 0.0067224        total: 5.27s    remaining: 8.32s
388:    learn: 0.0067134        total: 5.29s    remaining: 8.3s
389:    learn: 0.0067056        total: 5.3s     remaining: 8.29s
390:    learn: 0.0067017        total: 5.31s    remaining: 8.27s
391:    learn: 0.0066945        total: 5.32s    remaining: 8.25s
392:    learn: 0.0066927        total: 5.33s    remaining: 8.23s
393:    learn: 0.0066781        total: 5.34s    remaining: 8.22s
394:    learn: 0.0066709        total: 5.36s    remaining: 8.21s
395:    learn: 0.0066641        total: 5.37s    remaining: 8.2s
396:    learn: 0.0066568        total: 5.39s    remaining: 8.19s
397:    learn: 0.0066494        total: 5.41s    remaining: 8.18s
398:    learn: 0.0066457        total: 5.42s    remaining: 8.16s
399:    learn: 0.0066387        total: 5.43s    remaining: 8.15s
400:    learn: 0.0066299        total: 5.45s    remaining: 8.13s
401:    learn: 0.0066208        total: 5.46s    remaining: 8.13s
402:    learn: 0.0066129        total: 5.47s    remaining: 8.11s
403:    learn: 0.0066076        total: 5.49s    remaining: 8.1s
404:    learn: 0.0065940        total: 5.5s     remaining: 8.08s
405:    learn: 0.0065810        total: 5.51s    remaining: 8.06s
406:    learn: 0.0065773        total: 5.54s    remaining: 8.07s
407:    learn: 0.0065741        total: 5.55s    remaining: 8.05s
408:    learn: 0.0065662        total: 5.57s    remaining: 8.05s
409:    learn: 0.0065642        total: 5.58s    remaining: 8.03s
410:    learn: 0.0065610        total: 5.6s     remaining: 8.03s
411:    learn: 0.0065538        total: 5.61s    remaining: 8.01s
412:    learn: 0.0065471        total: 5.62s    remaining: 7.99s
413:    learn: 0.0065320        total: 5.65s    remaining: 8s
414:    learn: 0.0065222        total: 5.66s    remaining: 7.98s
415:    learn: 0.0065169        total: 5.68s    remaining: 7.97s
```

```
416:    learn: 0.0065105        total: 5.69s    remaining: 7.96s
417:    learn: 0.0065042        total: 5.71s    remaining: 7.94s
418:    learn: 0.0064974        total: 5.72s    remaining: 7.93s
419:    learn: 0.0064897        total: 5.74s    remaining: 7.92s
420:    learn: 0.0064834        total: 5.75s    remaining: 7.91s
421:    learn: 0.0064771        total: 5.76s    remaining: 7.89s
422:    learn: 0.0064643        total: 5.77s    remaining: 7.87s
423:    learn: 0.0064521        total: 5.79s    remaining: 7.87s
424:    learn: 0.0064478        total: 5.8s     remaining: 7.85s
425:    learn: 0.0064343        total: 5.82s    remaining: 7.84s
426:    learn: 0.0064311        total: 5.83s    remaining: 7.83s
427:    learn: 0.0064267        total: 5.85s    remaining: 7.82s
428:    learn: 0.0064164        total: 5.86s    remaining: 7.8s
429:    learn: 0.0064104        total: 5.87s    remaining: 7.79s
430:    learn: 0.0064049        total: 5.89s    remaining: 7.78s
431:    learn: 0.0063990        total: 5.91s    remaining: 7.77s
432:    learn: 0.0063976        total: 5.92s    remaining: 7.75s
433:    learn: 0.0063862        total: 5.93s    remaining: 7.74s
434:    learn: 0.0063820        total: 5.95s    remaining: 7.73s
435:    learn: 0.0063762        total: 5.96s    remaining: 7.72s
436:    learn: 0.0063707        total: 5.98s    remaining: 7.7s
437:    learn: 0.0063644        total: 5.99s    remaining: 7.69s
438:    learn: 0.0063581        total: 6.01s    remaining: 7.68s
439:    learn: 0.0063559        total: 6.02s    remaining: 7.67s
440:    learn: 0.0063492        total: 6.04s    remaining: 7.65s
441:    learn: 0.0063373        total: 6.05s    remaining: 7.64s
442:    learn: 0.0063246        total: 6.07s    remaining: 7.63s
443:    learn: 0.0063218        total: 6.08s    remaining: 7.61s
444:    learn: 0.0062965        total: 6.09s    remaining: 7.6s
445:    learn: 0.0062951        total: 6.1s     remaining: 7.58s
446:    learn: 0.0062912        total: 6.12s    remaining: 7.57s
447:    learn: 0.0062856        total: 6.13s    remaining: 7.55s
448:    learn: 0.0062797        total: 6.14s    remaining: 7.54s
449:    learn: 0.0062670        total: 6.16s    remaining: 7.52s
450:    learn: 0.0062642        total: 6.17s    remaining: 7.51s
451:    learn: 0.0062584        total: 6.18s    remaining: 7.49s
452:    learn: 0.0062510        total: 6.19s    remaining: 7.48s
453:    learn: 0.0062442        total: 6.21s    remaining: 7.46s
454:    learn: 0.0062421        total: 6.23s    remaining: 7.46s
455:    learn: 0.0062396        total: 6.24s    remaining: 7.45s
456:    learn: 0.0062371        total: 6.25s    remaining: 7.43s
457:    learn: 0.0062350        total: 6.27s    remaining: 7.42s
458:    learn: 0.0062330        total: 6.28s    remaining: 7.41s
459:    learn: 0.0062304        total: 6.3s     remaining: 7.39s
460:    learn: 0.0062280        total: 6.31s    remaining: 7.38s
461:    learn: 0.0062198        total: 6.32s    remaining: 7.36s
462:    learn: 0.0062179        total: 6.33s    remaining: 7.34s
463:    learn: 0.0062126        total: 6.34s    remaining: 7.33s
```

```
464:    learn: 0.0062107        total: 6.35s    remaining: 7.31s
465:    learn: 0.0062084        total: 6.37s    remaining: 7.3s
466:    learn: 0.0062065        total: 6.38s    remaining: 7.29s
467:    learn: 0.0062047        total: 6.4s     remaining: 7.27s
468:    learn: 0.0061970        total: 6.41s    remaining: 7.26s
469:    learn: 0.0061946        total: 6.42s    remaining: 7.24s
470:    learn: 0.0061928        total: 6.44s    remaining: 7.23s
471:    learn: 0.0061911        total: 6.45s    remaining: 7.21s
472:    learn: 0.0061791        total: 6.46s    remaining: 7.2s
473:    learn: 0.0061658        total: 6.47s    remaining: 7.18s
474:    learn: 0.0061587        total: 6.49s    remaining: 7.17s
475:    learn: 0.0061487        total: 6.5s     remaining: 7.15s
476:    learn: 0.0061412        total: 6.51s    remaining: 7.14s
477:    learn: 0.0061388        total: 6.53s    remaining: 7.13s
478:    learn: 0.0061321        total: 6.54s    remaining: 7.11s
479:    learn: 0.0061264        total: 6.55s    remaining: 7.1s
480:    learn: 0.0061116        total: 6.57s    remaining: 7.09s
481:    learn: 0.0061080        total: 6.58s    remaining: 7.07s
482:    learn: 0.0060966        total: 6.59s    remaining: 7.06s
483:    learn: 0.0060871        total: 6.6s     remaining: 7.04s
484:    learn: 0.0060849        total: 6.62s    remaining: 7.03s
485:    learn: 0.0060825        total: 6.63s    remaining: 7.01s
486:    learn: 0.0060802        total: 6.64s    remaining: 7s
487:    learn: 0.0060701        total: 6.66s    remaining: 6.98s
488:    learn: 0.0060663        total: 6.68s    remaining: 6.98s
489:    learn: 0.0060617        total: 6.69s    remaining: 6.97s
490:    learn: 0.0060565        total: 6.7s     remaining: 6.95s
491:    learn: 0.0060529        total: 6.72s    remaining: 6.94s
492:    learn: 0.0060441        total: 6.73s    remaining: 6.92s
493:    learn: 0.0060359        total: 6.75s    remaining: 6.91s
494:    learn: 0.0060310        total: 6.76s    remaining: 6.9s
495:    learn: 0.0060287        total: 6.77s    remaining: 6.88s
496:    learn: 0.0060272        total: 6.79s    remaining: 6.87s
497:    learn: 0.0060223        total: 6.8s     remaining: 6.86s
498:    learn: 0.0060208        total: 6.81s    remaining: 6.84s
499:    learn: 0.0060145        total: 6.83s    remaining: 6.83s
500:    learn: 0.0060102        total: 6.84s    remaining: 6.82s
501:    learn: 0.0059963        total: 6.85s    remaining: 6.8s
502:    learn: 0.0059940        total: 6.86s    remaining: 6.78s
503:    learn: 0.0059917        total: 6.88s    remaining: 6.78s
504:    learn: 0.0059871        total: 6.89s    remaining: 6.76s
505:    learn: 0.0059849        total: 6.91s    remaining: 6.75s
506:    learn: 0.0059834        total: 6.92s    remaining: 6.73s
507:    learn: 0.0059728        total: 6.93s    remaining: 6.71s
508:    learn: 0.0059687        total: 6.95s    remaining: 6.7s
509:    learn: 0.0059578        total: 6.96s    remaining: 6.68s
510:    learn: 0.0059532        total: 6.97s    remaining: 6.67s
511:    learn: 0.0059510        total: 6.98s    remaining: 6.65s
```

```
512:    learn: 0.0059496        total: 7s       remaining: 6.64s
513:    learn: 0.0059443        total: 7.01s    remaining: 6.63s
514:    learn: 0.0059402        total: 7.02s    remaining: 6.61s
515:    learn: 0.0059368        total: 7.03s    remaining: 6.6s
516:    learn: 0.0059346        total: 7.05s    remaining: 6.59s
517:    learn: 0.0059325        total: 7.07s    remaining: 6.57s
518:    learn: 0.0059312        total: 7.08s    remaining: 6.56s
519:    learn: 0.0059215        total: 7.09s    remaining: 6.55s
520:    learn: 0.0059123        total: 7.1s     remaining: 6.53s
521:    learn: 0.0059095        total: 7.12s    remaining: 6.52s
522:    learn: 0.0059025        total: 7.13s    remaining: 6.5s
523:    learn: 0.0058997        total: 7.14s    remaining: 6.48s
524:    learn: 0.0058951        total: 7.15s    remaining: 6.47s
525:    learn: 0.0058846        total: 7.16s    remaining: 6.46s
526:    learn: 0.0058740        total: 7.17s    remaining: 6.44s
527:    learn: 0.0058689        total: 7.18s    remaining: 6.42s
528:    learn: 0.0058546        total: 7.2s     remaining: 6.41s
529:    learn: 0.0058510        total: 7.21s    remaining: 6.4s
530:    learn: 0.0058466        total: 7.22s    remaining: 6.38s
531:    learn: 0.0058416        total: 7.24s    remaining: 6.37s
532:    learn: 0.0058343        total: 7.25s    remaining: 6.35s
533:    learn: 0.0058322        total: 7.26s    remaining: 6.33s
534:    learn: 0.0058302        total: 7.27s    remaining: 6.32s
535:    learn: 0.0058291        total: 7.29s    remaining: 6.31s
536:    learn: 0.0058231        total: 7.3s     remaining: 6.29s
537:    learn: 0.0058186        total: 7.31s    remaining: 6.28s
538:    learn: 0.0058133        total: 7.32s    remaining: 6.26s
539:    learn: 0.0058113        total: 7.33s    remaining: 6.25s
540:    learn: 0.0058081        total: 7.35s    remaining: 6.23s
541:    learn: 0.0058056        total: 7.36s    remaining: 6.22s
542:    learn: 0.0058036        total: 7.37s    remaining: 6.2s
543:    learn: 0.0057967        total: 7.38s    remaining: 6.19s
544:    learn: 0.0057930        total: 7.39s    remaining: 6.17s
545:    learn: 0.0057881        total: 7.41s    remaining: 6.16s
546:    learn: 0.0057810        total: 7.42s    remaining: 6.15s
547:    learn: 0.0057752        total: 7.43s    remaining: 6.13s
548:    learn: 0.0057692        total: 7.44s    remaining: 6.11s
549:    learn: 0.0057645        total: 7.45s    remaining: 6.1s
550:    learn: 0.0057625        total: 7.46s    remaining: 6.08s
551:    learn: 0.0057594        total: 7.48s    remaining: 6.07s
552:    learn: 0.0057574        total: 7.49s    remaining: 6.05s
553:    learn: 0.0057554        total: 7.5s     remaining: 6.04s
554:    learn: 0.0057470        total: 7.51s    remaining: 6.02s
555:    learn: 0.0057443        total: 7.52s    remaining: 6s
556:    learn: 0.0057385        total: 7.53s    remaining: 5.99s
557:    learn: 0.0057338        total: 7.55s    remaining: 5.98s
558:    learn: 0.0057163        total: 7.56s    remaining: 5.96s
559:    learn: 0.0057144        total: 7.57s    remaining: 5.95s
```

```
560:    learn: 0.0057053        total: 7.58s    remaining: 5.93s
561:    learn: 0.0057027        total: 7.59s    remaining: 5.92s
562:    learn: 0.0057008        total: 7.61s    remaining: 5.9s
563:    learn: 0.0056984        total: 7.62s    remaining: 5.89s
564:    learn: 0.0056975        total: 7.63s    remaining: 5.87s
565:    learn: 0.0056956        total: 7.64s    remaining: 5.86s
566:    learn: 0.0056920        total: 7.65s    remaining: 5.84s
567:    learn: 0.0056782        total: 7.66s    remaining: 5.83s
568:    learn: 0.0056748        total: 7.67s    remaining: 5.81s
569:    learn: 0.0056651        total: 7.68s    remaining: 5.8s
570:    learn: 0.0056605        total: 7.7s     remaining: 5.78s
571:    learn: 0.0056548        total: 7.71s    remaining: 5.77s
572:    learn: 0.0056494        total: 7.72s    remaining: 5.75s
573:    learn: 0.0056428        total: 7.73s    remaining: 5.74s
574:    learn: 0.0056265        total: 7.74s    remaining: 5.72s
575:    learn: 0.0056223        total: 7.75s    remaining: 5.71s
576:    learn: 0.0056205        total: 7.77s    remaining: 5.69s
577:    learn: 0.0056187        total: 7.78s    remaining: 5.68s
578:    learn: 0.0056169        total: 7.79s    remaining: 5.66s
579:    learn: 0.0056146        total: 7.8s     remaining: 5.65s
580:    learn: 0.0056085        total: 7.81s    remaining: 5.63s
581:    learn: 0.0056031        total: 7.83s    remaining: 5.62s
582:    learn: 0.0056010        total: 7.84s    remaining: 5.6s
583:    learn: 0.0055970        total: 7.85s    remaining: 5.59s
584:    learn: 0.0055918        total: 7.86s    remaining: 5.58s
585:    learn: 0.0055880        total: 7.87s    remaining: 5.56s
586:    learn: 0.0055836        total: 7.88s    remaining: 5.55s
587:    learn: 0.0055798        total: 7.9s     remaining: 5.53s
588:    learn: 0.0055662        total: 7.91s    remaining: 5.52s
589:    learn: 0.0055644        total: 7.93s    remaining: 5.51s
590:    learn: 0.0055622        total: 7.94s    remaining: 5.49s
591:    learn: 0.0055604        total: 7.95s    remaining: 5.48s
592:    learn: 0.0055570        total: 7.96s    remaining: 5.46s
593:    learn: 0.0055553        total: 7.97s    remaining: 5.45s
594:    learn: 0.0055535        total: 7.99s    remaining: 5.44s
595:    learn: 0.0055513        total: 8s       remaining: 5.42s
596:    learn: 0.0055471        total: 8.01s    remaining: 5.41s
597:    learn: 0.0055375        total: 8.02s    remaining: 5.39s
598:    learn: 0.0055358        total: 8.03s    remaining: 5.38s
599:    learn: 0.0055341        total: 8.04s    remaining: 5.36s
600:    learn: 0.0055248        total: 8.05s    remaining: 5.35s
601:    learn: 0.0055170        total: 8.07s    remaining: 5.33s
602:    learn: 0.0055116        total: 8.08s    remaining: 5.32s
603:    learn: 0.0055026        total: 8.09s    remaining: 5.3s
604:    learn: 0.0054975        total: 8.1s     remaining: 5.29s
605:    learn: 0.0054936        total: 8.12s    remaining: 5.28s
606:    learn: 0.0054869        total: 8.13s    remaining: 5.26s
607:    learn: 0.0054836        total: 8.14s    remaining: 5.25s
```

```
608:    learn: 0.0054819        total: 8.15s    remaining: 5.23s
609:    learn: 0.0054783        total: 8.16s    remaining: 5.22s
610:    learn: 0.0054767        total: 8.18s    remaining: 5.21s
611:    learn: 0.0054736        total: 8.19s    remaining: 5.19s
612:    learn: 0.0054661        total: 8.2s     remaining: 5.17s
613:    learn: 0.0054645        total: 8.21s    remaining: 5.16s
614:    learn: 0.0054597        total: 8.22s    remaining: 5.14s
615:    learn: 0.0054510        total: 8.23s    remaining: 5.13s
616:    learn: 0.0054464        total: 8.24s    remaining: 5.12s
617:    learn: 0.0054448        total: 8.25s    remaining: 5.1s
618:    learn: 0.0054403        total: 8.27s    remaining: 5.09s
619:    learn: 0.0054387        total: 8.28s    remaining: 5.07s
620:    learn: 0.0054372        total: 8.29s    remaining: 5.06s
621:    learn: 0.0054337        total: 8.3s     remaining: 5.04s
622:    learn: 0.0054290        total: 8.31s    remaining: 5.03s
623:    learn: 0.0054274        total: 8.32s    remaining: 5.01s
624:    learn: 0.0054191        total: 8.33s    remaining: 5s
625:    learn: 0.0054172        total: 8.34s    remaining: 4.98s
626:    learn: 0.0054128        total: 8.36s    remaining: 4.97s
627:    learn: 0.0054113        total: 8.37s    remaining: 4.96s
628:    learn: 0.0054008        total: 8.38s    remaining: 4.94s
629:    learn: 0.0053965        total: 8.39s    remaining: 4.93s
630:    learn: 0.0053923        total: 8.4s     remaining: 4.91s
631:    learn: 0.0053889        total: 8.41s    remaining: 4.9s
632:    learn: 0.0053819        total: 8.42s    remaining: 4.88s
633:    learn: 0.0053778        total: 8.44s    remaining: 4.87s
634:    learn: 0.0053758        total: 8.45s    remaining: 4.86s
635:    learn: 0.0053727        total: 8.46s    remaining: 4.84s
636:    learn: 0.0053643        total: 8.47s    remaining: 4.83s
637:    learn: 0.0053540        total: 8.48s    remaining: 4.81s
638:    learn: 0.0053525        total: 8.5s     remaining: 4.8s
639:    learn: 0.0053510        total: 8.51s    remaining: 4.79s
640:    learn: 0.0053367        total: 8.52s    remaining: 4.77s
641:    learn: 0.0053352        total: 8.53s    remaining: 4.76s
642:    learn: 0.0053273        total: 8.54s    remaining: 4.74s
643:    learn: 0.0053230        total: 8.55s    remaining: 4.73s
644:    learn: 0.0053174        total: 8.56s    remaining: 4.71s
645:    learn: 0.0053046        total: 8.57s    remaining: 4.7s
646:    learn: 0.0053031        total: 8.59s    remaining: 4.68s
647:    learn: 0.0052984        total: 8.6s     remaining: 4.67s
648:    learn: 0.0052970        total: 8.61s    remaining: 4.66s
649:    learn: 0.0052955        total: 8.62s    remaining: 4.64s
650:    learn: 0.0052923        total: 8.63s    remaining: 4.63s
651:    learn: 0.0052884        total: 8.64s    remaining: 4.61s
652:    learn: 0.0052752        total: 8.66s    remaining: 4.6s
653:    learn: 0.0052689        total: 8.67s    remaining: 4.58s
654:    learn: 0.0052659        total: 8.68s    remaining: 4.57s
655:    learn: 0.0052614        total: 8.69s    remaining: 4.56s
```

```
656:    learn: 0.0052596        total: 8.7s     remaining: 4.54s
657:    learn: 0.0052551        total: 8.71s    remaining: 4.53s
658:    learn: 0.0052510        total: 8.72s    remaining: 4.51s
659:    learn: 0.0052382        total: 8.73s    remaining: 4.5s
660:    learn: 0.0052325        total: 8.75s    remaining: 4.49s
661:    learn: 0.0052311        total: 8.76s    remaining: 4.47s
662:    learn: 0.0052192        total: 8.77s    remaining: 4.46s
663:    learn: 0.0052178        total: 8.78s    remaining: 4.44s
664:    learn: 0.0052045        total: 8.79s    remaining: 4.43s
665:    learn: 0.0051997        total: 8.8s     remaining: 4.42s
666:    learn: 0.0051959        total: 8.82s    remaining: 4.4s
667:    learn: 0.0051922        total: 8.83s    remaining: 4.39s
668:    learn: 0.0051909        total: 8.84s    remaining: 4.37s
669:    learn: 0.0051892        total: 8.85s    remaining: 4.36s
670:    learn: 0.0051876        total: 8.86s    remaining: 4.34s
671:    learn: 0.0051822        total: 8.87s    remaining: 4.33s
672:    learn: 0.0051695        total: 8.88s    remaining: 4.32s
673:    learn: 0.0051659        total: 8.9s     remaining: 4.3s
674:    learn: 0.0051645        total: 8.91s    remaining: 4.29s
675:    learn: 0.0051609        total: 8.92s    remaining: 4.27s
676:    learn: 0.0051547        total: 8.93s    remaining: 4.26s
677:    learn: 0.0051534        total: 8.94s    remaining: 4.25s
678:    learn: 0.0051486        total: 8.95s    remaining: 4.23s
679:    learn: 0.0051473        total: 8.97s    remaining: 4.22s
680:    learn: 0.0051431        total: 8.98s    remaining: 4.21s
681:    learn: 0.0051410        total: 8.99s    remaining: 4.19s
682:    learn: 0.0051372        total: 9s       remaining: 4.18s
683:    learn: 0.0051313        total: 9.01s    remaining: 4.16s
684:    learn: 0.0051278        total: 9.02s    remaining: 4.15s
685:    learn: 0.0051241        total: 9.03s    remaining: 4.13s
686:    learn: 0.0051208        total: 9.04s    remaining: 4.12s
687:    learn: 0.0051196        total: 9.06s    remaining: 4.11s
688:    learn: 0.0051155        total: 9.07s    remaining: 4.09s
689:    learn: 0.0051035        total: 9.08s    remaining: 4.08s
690:    learn: 0.0050971        total: 9.09s    remaining: 4.06s
691:    learn: 0.0050959        total: 9.1s     remaining: 4.05s
692:    learn: 0.0050945        total: 9.11s    remaining: 4.04s
693:    learn: 0.0050830        total: 9.12s    remaining: 4.02s
694:    learn: 0.0050811        total: 9.13s    remaining: 4.01s
695:    learn: 0.0050796        total: 9.14s    remaining: 3.99s
696:    learn: 0.0050778        total: 9.16s    remaining: 3.98s
697:    learn: 0.0050764        total: 9.17s    remaining: 3.97s
698:    learn: 0.0050751        total: 9.18s    remaining: 3.95s
699:    learn: 0.0050738        total: 9.19s    remaining: 3.94s
700:    learn: 0.0050704        total: 9.2s     remaining: 3.92s
701:    learn: 0.0050691        total: 9.21s    remaining: 3.91s
702:    learn: 0.0050679        total: 9.22s    remaining: 3.9s
703:    learn: 0.0050667        total: 9.23s    remaining: 3.88s
```

```
704:    learn: 0.0050657        total: 9.25s    remaining: 3.87s
705:    learn: 0.0050641        total: 9.26s    remaining: 3.85s
706:    learn: 0.0050580        total: 9.27s    remaining: 3.84s
707:    learn: 0.0050570        total: 9.28s    remaining: 3.83s
708:    learn: 0.0050556        total: 9.29s    remaining: 3.81s
709:    learn: 0.0050513        total: 9.3s     remaining: 3.8s
710:    learn: 0.0050418        total: 9.31s    remaining: 3.78s
711:    learn: 0.0050366        total: 9.32s    remaining: 3.77s
712:    learn: 0.0050324        total: 9.33s    remaining: 3.76s
713:    learn: 0.0050307        total: 9.35s    remaining: 3.74s
714:    learn: 0.0050275        total: 9.36s    remaining: 3.73s
715:    learn: 0.0050222        total: 9.37s    remaining: 3.72s
716:    learn: 0.0050181        total: 9.38s    remaining: 3.7s
717:    learn: 0.0050074        total: 9.39s    remaining: 3.69s
718:    learn: 0.0049968        total: 9.4s     remaining: 3.67s
719:    learn: 0.0049956        total: 9.41s    remaining: 3.66s
720:    learn: 0.0049855        total: 9.43s    remaining: 3.65s
721:    learn: 0.0049837        total: 9.44s    remaining: 3.63s
722:    learn: 0.0049782        total: 9.45s    remaining: 3.62s
723:    learn: 0.0049749        total: 9.46s    remaining: 3.61s
724:    learn: 0.0049732        total: 9.47s    remaining: 3.59s
725:    learn: 0.0049630        total: 9.48s    remaining: 3.58s
726:    learn: 0.0049618        total: 9.5s     remaining: 3.57s
727:    learn: 0.0049609        total: 9.51s    remaining: 3.55s
728:    learn: 0.0049597        total: 9.53s    remaining: 3.54s
729:    learn: 0.0049565        total: 9.54s    remaining: 3.53s
730:    learn: 0.0049512        total: 9.55s    remaining: 3.51s
731:    learn: 0.0049495        total: 9.56s    remaining: 3.5s
732:    learn: 0.0049486        total: 9.57s    remaining: 3.49s
733:    learn: 0.0049474        total: 9.58s    remaining: 3.47s
734:    learn: 0.0049466        total: 9.6s     remaining: 3.46s
735:    learn: 0.0049455        total: 9.61s    remaining: 3.45s
736:    learn: 0.0049446        total: 9.62s    remaining: 3.43s
737:    learn: 0.0049438        total: 9.63s    remaining: 3.42s
738:    learn: 0.0049398        total: 9.64s    remaining: 3.4s
739:    learn: 0.0049366        total: 9.65s    remaining: 3.39s
740:    learn: 0.0049326        total: 9.66s    remaining: 3.38s
741:    learn: 0.0049275        total: 9.68s    remaining: 3.36s
742:    learn: 0.0049177        total: 9.69s    remaining: 3.35s
743:    learn: 0.0049166        total: 9.71s    remaining: 3.34s
744:    learn: 0.0049139        total: 9.72s    remaining: 3.33s
745:    learn: 0.0049112        total: 9.73s    remaining: 3.31s
746:    learn: 0.0049082        total: 9.74s    remaining: 3.3s
747:    learn: 0.0049030        total: 9.75s    remaining: 3.29s
748:    learn: 0.0048951        total: 9.76s    remaining: 3.27s
749:    learn: 0.0048859        total: 9.78s    remaining: 3.26s
750:    learn: 0.0048816        total: 9.79s    remaining: 3.24s
751:    learn: 0.0048765        total: 9.8s     remaining: 3.23s
```

```
752:    learn: 0.0048675        total: 9.81s    remaining: 3.22s
753:    learn: 0.0048659        total: 9.82s    remaining: 3.2s
754:    learn: 0.0048648        total: 9.83s    remaining: 3.19s
755:    learn: 0.0048609        total: 9.84s    remaining: 3.18s
756:    learn: 0.0048562        total: 9.85s    remaining: 3.16s
757:    learn: 0.0048551        total: 9.87s    remaining: 3.15s
758:    learn: 0.0048448        total: 9.88s    remaining: 3.14s
759:    learn: 0.0048409        total: 9.89s    remaining: 3.12s
760:    learn: 0.0048399        total: 9.91s    remaining: 3.11s
761:    learn: 0.0048341        total: 9.92s    remaining: 3.1s
762:    learn: 0.0048325        total: 9.93s    remaining: 3.08s
763:    learn: 0.0048239        total: 9.94s    remaining: 3.07s
764:    learn: 0.0048223        total: 9.96s    remaining: 3.06s
765:    learn: 0.0048167        total: 9.97s    remaining: 3.04s
766:    learn: 0.0048129        total: 9.98s    remaining: 3.03s
767:    learn: 0.0048033        total: 10s      remaining: 3.02s
768:    learn: 0.0048017        total: 10s      remaining: 3.01s
769:    learn: 0.0047980        total: 10s      remaining: 2.99s
770:    learn: 0.0047950        total: 10s      remaining: 2.98s
771:    learn: 0.0047930        total: 10s      remaining: 2.97s
772:    learn: 0.0047899        total: 10.1s    remaining: 2.95s
773:    learn: 0.0047843        total: 10.1s    remaining: 2.94s
774:    learn: 0.0047816        total: 10.1s    remaining: 2.93s
775:    learn: 0.0047762        total: 10.1s    remaining: 2.91s
776:    learn: 0.0047698        total: 10.1s    remaining: 2.9s
777:    learn: 0.0047669        total: 10.1s    remaining: 2.89s
778:    learn: 0.0047633        total: 10.1s    remaining: 2.88s
779:    learn: 0.0047597        total: 10.2s    remaining: 2.86s
780:    learn: 0.0047568        total: 10.2s    remaining: 2.85s
781:    learn: 0.0047541        total: 10.2s    remaining: 2.84s
782:    learn: 0.0047473        total: 10.2s    remaining: 2.83s
783:    learn: 0.0047454        total: 10.2s    remaining: 2.81s
784:    learn: 0.0047416        total: 10.2s    remaining: 2.8s
785:    learn: 0.0047390        total: 10.2s    remaining: 2.79s
786:    learn: 0.0047381        total: 10.2s    remaining: 2.77s
787:    learn: 0.0047264        total: 10.3s    remaining: 2.76s
788:    learn: 0.0047256        total: 10.3s    remaining: 2.75s
789:    learn: 0.0047246        total: 10.3s    remaining: 2.73s
790:    learn: 0.0047238        total: 10.3s    remaining: 2.72s
791:    learn: 0.0047173        total: 10.3s    remaining: 2.71s
792:    learn: 0.0047119        total: 10.3s    remaining: 2.69s
793:    learn: 0.0047098        total: 10.3s    remaining: 2.68s
794:    learn: 0.0047088        total: 10.3s    remaining: 2.67s
795:    learn: 0.0047080        total: 10.4s    remaining: 2.65s
796:    learn: 0.0047072        total: 10.4s    remaining: 2.64s
797:    learn: 0.0047062        total: 10.4s    remaining: 2.63s
798:    learn: 0.0047055        total: 10.4s    remaining: 2.61s
799:    learn: 0.0047008        total: 10.4s    remaining: 2.6s
```

```
800:    learn: 0.0046922        total: 10.4s    remaining: 2.59s
801:    learn: 0.0046820        total: 10.4s    remaining: 2.58s
802:    learn: 0.0046740        total: 10.4s    remaining: 2.56s
803:    learn: 0.0046730        total: 10.5s    remaining: 2.55s
804:    learn: 0.0046709        total: 10.5s    remaining: 2.54s
805:    learn: 0.0046689        total: 10.5s    remaining: 2.52s
806:    learn: 0.0046653        total: 10.5s    remaining: 2.51s
807:    learn: 0.0046479        total: 10.5s    remaining: 2.5s
808:    learn: 0.0046408        total: 10.5s    remaining: 2.48s
809:    learn: 0.0046389        total: 10.5s    remaining: 2.47s
810:    learn: 0.0046338        total: 10.5s    remaining: 2.46s
811:    learn: 0.0046320        total: 10.6s    remaining: 2.44s
812:    learn: 0.0046250        total: 10.6s    remaining: 2.43s
813:    learn: 0.0046236        total: 10.6s    remaining: 2.42s
814:    learn: 0.0046183        total: 10.6s    remaining: 2.4s
815:    learn: 0.0046158        total: 10.6s    remaining: 2.39s
816:    learn: 0.0046132        total: 10.6s    remaining: 2.38s
817:    learn: 0.0046112        total: 10.6s    remaining: 2.37s
818:    learn: 0.0046102        total: 10.6s    remaining: 2.35s
819:    learn: 0.0046082        total: 10.7s    remaining: 2.34s
820:    learn: 0.0046061        total: 10.7s    remaining: 2.33s
821:    learn: 0.0046041        total: 10.7s    remaining: 2.31s
822:    learn: 0.0046021        total: 10.7s    remaining: 2.3s
823:    learn: 0.0045957        total: 10.7s    remaining: 2.29s
824:    learn: 0.0045932        total: 10.7s    remaining: 2.27s
825:    learn: 0.0045844        total: 10.7s    remaining: 2.26s
826:    learn: 0.0045747        total: 10.7s    remaining: 2.25s
827:    learn: 0.0045712        total: 10.7s    remaining: 2.23s
828:    learn: 0.0045693        total: 10.8s    remaining: 2.22s
829:    learn: 0.0045620        total: 10.8s    remaining: 2.21s
830:    learn: 0.0045598        total: 10.8s    remaining: 2.19s
831:    learn: 0.0045547        total: 10.8s    remaining: 2.18s
832:    learn: 0.0045528        total: 10.8s    remaining: 2.17s
833:    learn: 0.0045466        total: 10.8s    remaining: 2.15s
834:    learn: 0.0045446        total: 10.8s    remaining: 2.14s
835:    learn: 0.0045427        total: 10.8s    remaining: 2.13s
836:    learn: 0.0045403        total: 10.9s    remaining: 2.11s
837:    learn: 0.0045289        total: 10.9s    remaining: 2.1s
838:    learn: 0.0045243        total: 10.9s    remaining: 2.09s
839:    learn: 0.0045109        total: 10.9s    remaining: 2.07s
840:    learn: 0.0045073        total: 10.9s    remaining: 2.06s
841:    learn: 0.0045030        total: 10.9s    remaining: 2.05s
842:    learn: 0.0045021        total: 10.9s    remaining: 2.04s
843:    learn: 0.0045012        total: 10.9s    remaining: 2.02s
844:    learn: 0.0044950        total: 11s      remaining: 2.01s
845:    learn: 0.0044932        total: 11s      remaining: 2s
846:    learn: 0.0044913        total: 11s      remaining: 1.99s
847:    learn: 0.0044895        total: 11s      remaining: 1.97s
```

```
848:    learn: 0.0044877        total: 11s      remaining: 1.96s
849:    learn: 0.0044869        total: 11s      remaining: 1.95s
850:    learn: 0.0044850        total: 11s      remaining: 1.93s
851:    learn: 0.0044841        total: 11.1s    remaining: 1.92s
852:    learn: 0.0044834        total: 11.1s    remaining: 1.91s
853:    learn: 0.0044709        total: 11.1s    remaining: 1.89s
854:    learn: 0.0044614        total: 11.1s    remaining: 1.88s
855:    learn: 0.0044592        total: 11.1s    remaining: 1.87s
856:    learn: 0.0044531        total: 11.1s    remaining: 1.85s
857:    learn: 0.0044513        total: 11.1s    remaining: 1.84s
858:    learn: 0.0044495        total: 11.1s    remaining: 1.83s
859:    learn: 0.0044478        total: 11.2s    remaining: 1.82s
860:    learn: 0.0044456        total: 11.2s    remaining: 1.8s
861:    learn: 0.0044434        total: 11.2s    remaining: 1.79s
862:    learn: 0.0044417        total: 11.2s    remaining: 1.78s
863:    learn: 0.0044396        total: 11.2s    remaining: 1.76s
864:    learn: 0.0044368        total: 11.2s    remaining: 1.75s
865:    learn: 0.0044350        total: 11.2s    remaining: 1.74s
866:    learn: 0.0044333        total: 11.3s    remaining: 1.73s
867:    learn: 0.0044231        total: 11.3s    remaining: 1.71s
868:    learn: 0.0044209        total: 11.3s    remaining: 1.7s
869:    learn: 0.0044152        total: 11.3s    remaining: 1.69s
870:    learn: 0.0044132        total: 11.3s    remaining: 1.67s
871:    learn: 0.0044055        total: 11.3s    remaining: 1.66s
872:    learn: 0.0044035        total: 11.3s    remaining: 1.65s
873:    learn: 0.0044028        total: 11.3s    remaining: 1.63s
874:    learn: 0.0044015        total: 11.4s    remaining: 1.62s
875:    learn: 0.0043952        total: 11.4s    remaining: 1.61s
876:    learn: 0.0043925        total: 11.4s    remaining: 1.59s
877:    learn: 0.0043879        total: 11.4s    remaining: 1.58s
878:    learn: 0.0043804        total: 11.4s    remaining: 1.57s
879:    learn: 0.0043788        total: 11.4s    remaining: 1.56s
880:    learn: 0.0043773        total: 11.4s    remaining: 1.54s
881:    learn: 0.0043725        total: 11.4s    remaining: 1.53s
882:    learn: 0.0043695        total: 11.5s    remaining: 1.52s
883:    learn: 0.0043606        total: 11.5s    remaining: 1.5s
884:    learn: 0.0043535        total: 11.5s    remaining: 1.49s
885:    learn: 0.0043467        total: 11.5s    remaining: 1.48s
886:    learn: 0.0043379        total: 11.5s    remaining: 1.47s
887:    learn: 0.0043362        total: 11.5s    remaining: 1.45s
888:    learn: 0.0043305        total: 11.5s    remaining: 1.44s
889:    learn: 0.0043241        total: 11.6s    remaining: 1.43s
890:    learn: 0.0043223        total: 11.6s    remaining: 1.42s
891:    learn: 0.0043157        total: 11.6s    remaining: 1.4s
892:    learn: 0.0043130        total: 11.6s    remaining: 1.39s
893:    learn: 0.0043090        total: 11.6s    remaining: 1.38s
894:    learn: 0.0043070        total: 11.6s    remaining: 1.36s
895:    learn: 0.0043053        total: 11.6s    remaining: 1.35s
```

```
896:    learn: 0.0043010        total: 11.6s    remaining: 1.34s
897:    learn: 0.0042984        total: 11.7s    remaining: 1.32s
898:    learn: 0.0042964        total: 11.7s    remaining: 1.31s
899:    learn: 0.0042937        total: 11.7s    remaining: 1.3s
900:    learn: 0.0042922        total: 11.7s    remaining: 1.28s
901:    learn: 0.0042897        total: 11.7s    remaining: 1.27s
902:    learn: 0.0042871        total: 11.7s    remaining: 1.26s
903:    learn: 0.0042845        total: 11.7s    remaining: 1.25s
904:    learn: 0.0042822        total: 11.7s    remaining: 1.23s
905:    learn: 0.0042751        total: 11.8s    remaining: 1.22s
906:    learn: 0.0042629        total: 11.8s    remaining: 1.21s
907:    learn: 0.0042587        total: 11.8s    remaining: 1.19s
908:    learn: 0.0042560        total: 11.8s    remaining: 1.18s
909:    learn: 0.0042536        total: 11.8s    remaining: 1.17s
910:    learn: 0.0042513        total: 11.8s    remaining: 1.16s
911:    learn: 0.0042490        total: 11.8s    remaining: 1.14s
912:    learn: 0.0042468        total: 11.8s    remaining: 1.13s
913:    learn: 0.0042446        total: 11.9s    remaining: 1.12s
914:    learn: 0.0042424        total: 11.9s    remaining: 1.1s
915:    learn: 0.0042403        total: 11.9s    remaining: 1.09s
916:    learn: 0.0042381        total: 11.9s    remaining: 1.08s
917:    learn: 0.0042360        total: 11.9s    remaining: 1.06s
918:    learn: 0.0042320        total: 11.9s    remaining: 1.05s
919:    learn: 0.0042299        total: 12s      remaining: 1.04s
920:    learn: 0.0042266        total: 12s      remaining: 1.03s
921:    learn: 0.0042247        total: 12s      remaining: 1.01s
922:    learn: 0.0042227        total: 12s      remaining: 1s
923:    learn: 0.0042206        total: 12s      remaining: 988ms
924:    learn: 0.0042179        total: 12s      remaining: 974ms
925:    learn: 0.0042159        total: 12s      remaining: 962ms
926:    learn: 0.0042139        total: 12.1s    remaining: 949ms
927:    learn: 0.0042120        total: 12.1s    remaining: 936ms
928:    learn: 0.0042102        total: 12.1s    remaining: 923ms
929:    learn: 0.0042083        total: 12.1s    remaining: 910ms
930:    learn: 0.0042064        total: 12.1s    remaining: 897ms
931:    learn: 0.0042042        total: 12.1s    remaining: 884ms
932:    learn: 0.0042023        total: 12.1s    remaining: 871ms
933:    learn: 0.0042005        total: 12.1s    remaining: 858ms
934:    learn: 0.0041978        total: 12.2s    remaining: 845ms
935:    learn: 0.0041939        total: 12.2s    remaining: 832ms
936:    learn: 0.0041921        total: 12.2s    remaining: 819ms
937:    learn: 0.0041904        total: 12.2s    remaining: 806ms
938:    learn: 0.0041887        total: 12.2s    remaining: 793ms
939:    learn: 0.0041866        total: 12.2s    remaining: 780ms
940:    learn: 0.0041849        total: 12.2s    remaining: 767ms
941:    learn: 0.0041822        total: 12.3s    remaining: 754ms
942:    learn: 0.0041817        total: 12.3s    remaining: 741ms
943:    learn: 0.0041800        total: 12.3s    remaining: 728ms
```

```
944:    learn: 0.0041784        total: 12.3s    remaining: 715ms
945:    learn: 0.0041768        total: 12.3s    remaining: 702ms
946:    learn: 0.0041727        total: 12.3s    remaining: 689ms
947:    learn: 0.0041708        total: 12.3s    remaining: 676ms
948:    learn: 0.0041688        total: 12.3s    remaining: 663ms
949:    learn: 0.0041672        total: 12.4s    remaining: 651ms
950:    learn: 0.0041654        total: 12.4s    remaining: 638ms
951:    learn: 0.0041639        total: 12.4s    remaining: 625ms
952:    learn: 0.0041612        total: 12.4s    remaining: 612ms
953:    learn: 0.0041601        total: 12.4s    remaining: 599ms
954:    learn: 0.0041505        total: 12.4s    remaining: 586ms
955:    learn: 0.0041386        total: 12.4s    remaining: 573ms
956:    learn: 0.0041377        total: 12.5s    remaining: 560ms
957:    learn: 0.0041362        total: 12.5s    remaining: 547ms
958:    learn: 0.0041335        total: 12.5s    remaining: 534ms
959:    learn: 0.0041300        total: 12.5s    remaining: 520ms
960:    learn: 0.0041278        total: 12.5s    remaining: 507ms
961:    learn: 0.0041264        total: 12.5s    remaining: 494ms
962:    learn: 0.0041249        total: 12.5s    remaining: 481ms
963:    learn: 0.0041235        total: 12.5s    remaining: 468ms
964:    learn: 0.0041218        total: 12.6s    remaining: 455ms
965:    learn: 0.0041214        total: 12.6s    remaining: 442ms
966:    learn: 0.0041176        total: 12.6s    remaining: 429ms
967:    learn: 0.0041139        total: 12.6s    remaining: 416ms
968:    learn: 0.0041126        total: 12.6s    remaining: 404ms
969:    learn: 0.0041112        total: 12.6s    remaining: 391ms
970:    learn: 0.0041108        total: 12.6s    remaining: 377ms
971:    learn: 0.0041095        total: 12.7s    remaining: 365ms
972:    learn: 0.0041081        total: 12.7s    remaining: 352ms
973:    learn: 0.0041062        total: 12.7s    remaining: 338ms
974:    learn: 0.0041059        total: 12.7s    remaining: 326ms
975:    learn: 0.0041046        total: 12.7s    remaining: 313ms
976:    learn: 0.0041033        total: 12.7s    remaining: 300ms
977:    learn: 0.0041029        total: 12.7s    remaining: 286ms
978:    learn: 0.0041007        total: 12.7s    remaining: 273ms
979:    learn: 0.0040994        total: 12.8s    remaining: 260ms
980:    learn: 0.0040973        total: 12.8s    remaining: 247ms
981:    learn: 0.0040947        total: 12.8s    remaining: 234ms
982:    learn: 0.0040883        total: 12.8s    remaining: 221ms
983:    learn: 0.0040848        total: 12.8s    remaining: 208ms
984:    learn: 0.0040806        total: 12.8s    remaining: 195ms
985:    learn: 0.0040784        total: 12.8s    remaining: 182ms
986:    learn: 0.0040758        total: 12.8s    remaining: 169ms
987:    learn: 0.0040734        total: 12.9s    remaining: 156ms
988:    learn: 0.0040680        total: 12.9s    remaining: 143ms
989:    learn: 0.0040673        total: 12.9s    remaining: 130ms
990:    learn: 0.0040585        total: 12.9s    remaining: 117ms
991:    learn: 0.0040562        total: 12.9s    remaining: 104ms
```

```
992:    learn: 0.0040549        total: 12.9s    remaining: 91.1ms
993:    learn: 0.0040536        total: 12.9s    remaining: 78.1ms
994:    learn: 0.0040523        total: 13s      remaining: 65.2ms
995:    learn: 0.0040511        total: 13s      remaining: 52.2ms
996:    learn: 0.0040485        total: 13s      remaining: 39.1ms
997:    learn: 0.0040462        total: 13s      remaining: 26.1ms
998:    learn: 0.0040319        total: 13s      remaining: 13ms
999:    learn: 0.0040296        total: 13.1s    remaining: 0us
```

[308]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=<catboost.core.CatBoostClassifier object at
0x0000025775ED1CC8>,
             iid='deprecated', n_jobs=-1,
             param_grid={'depth': range(2, 10, 2),
                         'l2_leaf_reg': range(2, 10, 2),
                         'learning_rate': [0.03, 0.1]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

[313]:
```
print(f'Best Mean Cross Validation Score is {catb_grid.best_params_}')
print(f'Best Mean Cross Validation Score is {catb_grid.best_score_}')
print(f'Train score is {catb_grid.score(X_train,y_train)}')
```

```
Best Mean Cross Validation Score is {'depth': 4, 'l2_leaf_reg': 6,
'learning_rate': 0.03}
Best Mean Cross Validation Score is 0.8564556204653645
Train score is 0.9181969949916529
```

4.3.3 LightGBM Classifier - .79816

[310]:
```
#pip install lightgbm
```

[248]:
```
from lightgbm import LGBMClassifier
```

[311]:
```
lgbm = LGBMClassifier()
param_lgbm = {
    'n_estimators': range(50,70,10),
    'colsample_bytree': [0.7, 0.8],
    'max_depth': range(2,10,5),
    'num_leaves': range(2,10,2),
    'reg_alpha': [1.1, 1.2, 1.3],
    'reg_lambda': [1.1, 1.2, 1.3],
    'min_split_gain': [0.3, 0.4],
    'subsample': [0.7, 0.8, 0.9],
    'subsample_freq': [20]
}
```

```
grid_lgbm = GridSearchCV(lgbm, param_lgbm,cv=5,n_jobs=-1,␣
 ↪return_train_score=True, scoring =f2score)
grid_lgbm.fit(X_train,y_train)
```

[311]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=LGBMClassifier(boosting_type='gbdt', class_weight=None,
                                      colsample_bytree=1.0,
                                      importance_type='split',
                                      learning_rate=0.1, max_depth=-1,
                                      min_child_samples=20,
                                      min_child_weight=0.001,
                                      min_split_gain=0.0, n_estimators=100,
                                      n_jobs=-1, num_leaves=31, objective=None,
                                      random_state=None, reg_alpha=0.0,
                                      reg_lambda=0.0, silent=Tru…
             param_grid={'colsample_bytree': [0.7, 0.8],
                         'max_depth': range(2, 10, 5),
                         'min_split_gain': [0.3, 0.4],
                         'n_estimators': range(50, 70, 10),
                         'num_leaves': range(2, 10, 2),
                         'reg_alpha': [1.1, 1.2, 1.3],
                         'reg_lambda': [1.1, 1.2, 1.3],
                         'subsample': [0.7, 0.8, 0.9], 'subsample_freq': [20]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

[312]:
```
print(f'Best Mean Cross Validation Score is {grid_lgbm.best_params_}')
print(f'Best Mean Cross Validation Score is {grid_lgbm.best_score_}')
print(f'Train score is {grid_lgbm.score(X_train,y_train)}')
```

```
Best Mean Cross Validation Score is {'colsample_bytree': 0.7, 'max_depth': 7,
'min_split_gain': 0.4, 'n_estimators': 60, 'num_leaves': 8, 'reg_alpha': 1.3,
'reg_lambda': 1.3, 'subsample': 0.9, 'subsample_freq': 20}
Best Mean Cross Validation Score is 0.8398943998871164
Train score is 0.8912133891213391
```

4.3.4 One Class Classification for imbalanced data (Isolation forest)

[253]:
```
from sklearn.ensemble import IsolationForest
```

[273]:
```
f2score2 = make_scorer(fbeta_score,beta=2,average='macro')#[None, 'micro',␣
 ↪'macro', 'weighted']
```

[274]:
```
isoforest = IsolationForest(random_state=42)
iso_param = {
    'n_estimators':range(10,60,10),
    'max_samples':['auto'],
```

```
    'contamination':['auto']
}
grid_iso = GridSearchCV(isoforest, iso_param,cv=5,n_jobs=-1,␣
 ↪return_train_score=True, scoring =f2score2)
grid_iso.fit(X_train,y_train)
```

[274]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=IsolationForest(behaviour='deprecated', bootstrap=False,
                                       contamination='auto', max_features=1.0,
                                       max_samples='auto', n_estimators=100,
                                       n_jobs=None, random_state=42, verbose=0,
                                       warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'contamination': ['auto'], 'max_samples': ['auto'],
                         'n_estimators': range(10, 60, 10)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=make_scorer(fbeta_score, beta=2, average=macro),
             verbose=0)
```

[275]:
```
print(f'Best Mean Cross Validation Score is {grid_iso.best_params_}')
print(f'Best Mean Cross Validation Score is {grid_iso.best_score_}')
print(f'Train score is {grid_iso.score(X_train,y_train)}')
```

```
Best Mean Cross Validation Score is {'contamination': 'auto', 'max_samples':
'auto', 'n_estimators': 30}
Best Mean Cross Validation Score is 0.003558269018006486
Train score is 0.0031987205117952823
```

4.3.5 Naive Bayes

[284]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
```

GaussianNB - .79670

[304]:
```
naiveGaussian = GaussianNB()
naive_param = {}
grid_naiveGB = GridSearchCV(naivebayes, naive_param,cv=5,n_jobs=-1,␣
 ↪return_train_score=True, scoring =f2score2)
grid_naiveGB.fit(X_train,y_train)
```

[304]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None,
                                   fit_prior=True),
             iid='deprecated', n_jobs=-1, param_grid={},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
```

```
                scoring=make_scorer(fbeta_score, beta=2, average=macro),
                verbose=0)
```

[305]:
```
print(f'Best Mean Cross Validation Score is {grid_naiveGB.best_params_}')
print(f'Best Mean Cross Validation Score is {grid_naiveGB.best_score_}')
print(f'Train score is {grid_naiveGB.score(X_train,y_train)}')
```

```
Best Mean Cross Validation Score is {}
Best Mean Cross Validation Score is 0.8851045876061363
Train score is 0.8873985829077566
```

BernoulliNB - .79670

[318]:
```
naivebernoulli = BernoulliNB()
naive_param = {}
grid_naiveber = GridSearchCV(naivebernoulli, naive_param,cv=5,n_jobs=-1,␣
 ↪return_train_score=True, scoring =f2score2)
grid_naiveber.fit(X_train,y_train)
```

[318]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None,
                                   fit_prior=True),
             iid='deprecated', n_jobs=-1, param_grid={},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=make_scorer(fbeta_score, beta=2, average=macro),
             verbose=0)
```

[320]:
```
print(f'Best Mean Cross Validation Score is {grid_naiveber.best_params_}')
print(f'Best Mean Cross Validation Score is {grid_naiveber.best_score_}')
print(f'Train score is {grid_naiveber.score(X_train,y_train)}')
```

```
Best Mean Cross Validation Score is {}
Best Mean Cross Validation Score is 0.8851045876061363
Train score is 0.8873985829077566
```

5 Saving and Restoring Model

[377]:
```
import pickle
```

[378]:
```
# save the model to disk
filename = 'XGBoost_Cost_Sensitive.sav'
pickle.dump(grid_xgboost, open(filename, 'wb'))
```

[379]:
```
# load the model from disk
loaded_model = pickle.load(open(filename, 'rb'))
```

[380]:
```
loaded_model
```

```
[380]: GridSearchCV(cv=5, error_score=nan,
                    estimator=XGBClassifier(base_score=None, booster=None,
                                            colsample_bylevel=None,
                                            colsample_bynode=None,
                                            colsample_bytree=None,
                                            early_stopping_rounds=5, gamma=None,
                                            gpu_id=None, importance_type='gain',
                                            interaction_constraints=None,
                                            learning_rate=None, max_delta_step=None,
                                            max_depth=None, min_child_weight=None,
                                            missing=nan, monotone_…
                                            subsample=None, tree_method=None,
                                            validate_parameters=False,
                                            verbosity=None),
                    iid='deprecated', n_jobs=-1,
                    param_grid={'class_weight': ['balanced', 'balanced_subsample'],
                                'learning_rate': [0.05, 0.01, 0.1],
                                'n_estimators': range(1000, 1200, 20),
                                'scale_pos_weight': range(1000, 1200, 50)},
                    pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                    scoring=make_scorer(fbeta_score, beta=2), verbose=0)
```

6 The End