higgs_classification_XGBClassifier

February 24, 2021

This is an example Notebook for running training on Higgs vs background signal classification.

Background: High-energy collisions at the Large Hadron Collider (LHC) produce particles that interact with particle detectors. One important task is to classify different types of collisions based on their physics content, allowing physicists to find patterns in the data and to potentially unravel new discoveries.

Problem statement: The discovery of the Higgs boson by CMS and ATLAS Collaborations was announced at CERN in 2012. In this work, we focus on the potential of Machine Learning and Deep Learning in detecting potential Higgs signal from one of the background processes that mimics it.

Dataset: The dataset is made available by the Center for Machine Learning and Intelligent Systems at University of California, Irvine. The dataset can be found on the UCI Machine learning Repository

Description: The dataset consists of a total of 11 million labeled samples of Higgs vs background events produced by Monte Carlo simulations. Each sample consists of 28 features. The first 21 features are kinematic properties measured at the level of the detectors. The last seven are functions of the first 21.

Steps to load the training dataset 1. Download the dataset from the UCI website.

[1]:
```
from sklearn.datasets import make_gaussian_quantiles
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
```

[2]:
```
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
```

[3]:
```
import numpy as np
np.random.seed(1337)  # for reproducibility
import h5py



from sklearn.metrics import roc_curve, auc
```

```
import matplotlib.pyplot as plt
```

Load the file using pandas library

[4]:
```
data=pd.read_csv(r'./HIGGS.csv')
```

Assign first column 0 to class labels (labeled 1 for signal, 0 for background) and all others to feature matrix X.

In this example, for the sake of fast checking, we use 1000 samples. To train on the entire dataset, proceed with uncommenting the lines below.

[30]:
```
X=data.iloc[:1000000,1:]
y=data.iloc[:1000000,0]
```

[31]:
```
X.shape
```

[31]:
```
(1000000, 28)
```

Split your data into training and validation samples where the fraction of the data used for validation is 33%.

[32]:
```
X_train1, X_test, y_train1, y_test = train_test_split(X, y, test_size=0.1,␣
↪random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train1, y_train1,␣
↪test_size=0.33, random_state=42)
```

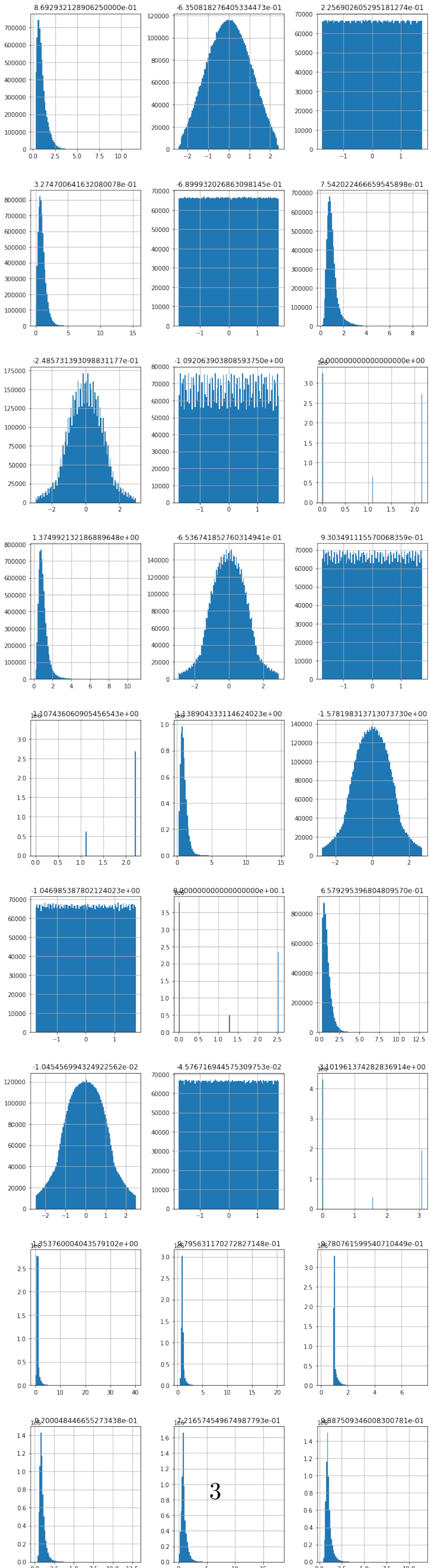
Visualize your data - One histogram per feature column

Detailed information on what each feature column is can be found in *Attribute Information* section on the UCI Machine learning Repositery. For further information, refer to the paper by Baldi et. al

[ ]:
```
from itertools import combinations
import matplotlib.pyplot as plt

fig, axes = plt.subplots(len(X_train.columns)//3, 3, figsize=(12, 48))

i = 0
for triaxis in axes:
    for axis in triaxis:
        X_train.hist(column = X_train.columns[i], bins = 100, ax=axis)
        i = i+1
```


8.692932128906250000e-01
-6.350818276405334473e-01
2.256902605295181274e-01
3.274700641632080078e-01
-6.899932026863098145e-01
7.542022466659545898e-01
-2.485731393098831177e-01
-1.092063903808593750e+00
0.000000000000000000e+00
1.374992132186889648e+00
-6.536741852760314941e-01
9.303491115570068359e-01
-1.107436060905456543e+00
1.138904333114624023e+00
-1.578198313713073730e+00
-1.046985387802124023e+00
0.000000000000000000e+00
6.579295396804809570e-01
-1.045456994324922562e-02
-4.576716944575309753e-02
3.101961374282836914e+00
1.353760004043579102e+00
9.795631170272827148e-01
8.780761599540710449e-01
9.200048446655273438e-01
7.216574549674987793e-01
8.887509346008300781e-01

XGBClassifier

[33]:
```
from xgboost import XGBClassifier
```

[ ]:
```
import xgboost as xgb
xgb.__version__
```

[34]:
```
from sklearn.model_selection import RandomizedSearchCV, StratifiedKFold
import numpy.random as rnd
from sklearn.metrics import make_scorer
from sklearn.metrics import classification_report
```

[ ]:
```
xgb_clf = XGBClassifier(objective='binary:logistic',
                        random_state=42,
                        n_jobs=-1,
                        eval_metric=["error", "auc"],
                        booster='dart',
                        sample_type= 'weighted',
                        normalize_type= 'forest',
                        #tree_method = "gpu_hist",
                        #sampling_method='gradient_based'
                        use_label_encoder=False,
                       ) #tree_method = "gpu_hist" may create data loss
param_search = {
    "n_estimators" : rnd.randint(0,1000,50),
    "max_depth" : rnd.randint(0,10,5),
    "learning_rate" : rnd.uniform(0,0.5,10),
    #'gamma' : rnd.uniform(0,10),
    "colsample_bytree" : rnd.uniform(0.5,1.,10),
    "colsample_bylevel" : rnd.uniform(0.5,1.,10),
    "colsample_bynode" : rnd.uniform(0.5,1.,10),
    "subsample" : rnd.uniform(0.2,1.,10),
    "rate_drop" : rnd.uniform(0.1,1.,10),
    "skip_drop" : rnd.uniform(0.1,1.,10)
}
scoring = {
    'AUC': 'roc_auc',
    'Accuracy': make_scorer(accuracy_score)
}
num_folds = 3
num_iter = 50
kfold = StratifiedKFold(n_splits=num_folds, random_state=42, shuffle=True)
grid_search = RandomizedSearchCV(
    estimator=xgb_clf,
    param_distributions=param_search,
```

```
    cv=kfold,
    scoring=scoring,
    #n_jobs=-1,
    n_iter=num_iter,
    refit="AUC",
)
best_model = grid_search.fit(X_train,y_train)
```

[36]:
```
best_model.cv_results_
```

[36]:
```
{'mean_fit_time': array([69.30064106, 25.52339872, 79.40737637, 63.15403128,
73.40759261,
        22.87666504, 81.27012459, 55.67759339, 67.56116899, 78.6487844 ,
        57.07299121, 23.51831667, 75.30270855,  9.46554939, 67.16508698,
        92.32556915,  9.60410031, 70.1104331 , 82.72037538,  9.50667198,
        12.55819702, 94.34127355,  9.1917874 , 70.71104844, 70.764642  ,
        72.3040514 , 11.7794776 , 83.70684163, 58.81702956, 77.52419631,
        78.60914906, 95.44175529, 77.16676585, 49.77839891, 61.56748462,
        77.49153606, 76.81652466, 67.01249329, 70.54881398, 68.67368674,
        12.88604307, 20.6761562 , 62.42326744, 66.45851763, 74.71959058,
        22.99034293, 11.87409306,  7.8979706 , 26.84083231, 88.7164731 ]),
 'std_fit_time': array([1.74911416, 0.56010329, 0.6409464 , 1.39476337,
0.69230573,
        0.16513723, 0.4406365 , 0.29806806, 0.40786012, 1.00692034,
        2.66889126, 1.37339602, 5.40848814, 0.31017415, 1.87775217,
        0.16672438, 0.66186596, 1.13934208, 2.98981756, 0.14732786,
        0.81223561, 0.61319762, 0.09395639, 0.10095908, 0.41940377,
        0.37111801, 0.0809181 , 0.20112218, 1.59502439, 0.28530797,
        2.29323651, 3.03494068, 1.66621769, 1.17228268, 0.19689782,
        1.40323187, 0.22041946, 3.58441313, 3.01553574, 1.17814906,
        0.23872194, 0.5296568 , 0.47959624, 0.78861709, 0.0990834 ,
        0.4650299 , 0.17522614, 0.07680413, 1.0018671 , 1.16326038]),
 'mean_score_time': array([1.26622732, 0.511537  , 1.29505928, 1.03398395,
1.25290831,
        0.47663967, 1.28255073, 1.00399462, 1.23402095, 1.03939295,
        1.00598105, 0.53067056, 1.323488  , 0.25028165, 1.00515564,
        1.27385132, 0.26233602, 1.4432253 , 1.33848453, 0.24700268,
        0.25840902, 1.26478537, 0.2511394 , 0.99787259, 1.02019159,
        1.31717157, 0.2458454 , 1.26549935, 1.06745712, 1.1431303 ,
        1.47842566, 1.00415007, 1.04265738, 1.05846659, 0.99916617,
        1.36329651, 1.06083274, 1.17577767, 1.05414764, 1.2973512 ,
        0.24449253, 0.50993188, 1.03659789, 1.03744411, 1.32694872,
        0.5004584 , 0.24697797, 0.24100415, 0.50731842, 1.46756132]),
 'std_score_time': array([0.03996241, 0.0145614 , 0.02305774, 0.07049324,
0.02395025,
        0.00696864, 0.03749239, 0.01396993, 0.0190412 , 0.02497258,
        0.02137795, 0.01153468, 0.10094924, 0.01279486, 0.01113124,
```

```
       0.04949942, 0.01622007, 0.08474608, 0.06266971, 0.00597808,
       0.01227326, 0.01666395, 0.01559172, 0.00747897, 0.0307596 ,
       0.0479164 , 0.00806341, 0.0323117 , 0.07699052, 0.10672462,
       0.17200446, 0.07151347, 0.02620966, 0.03309419, 0.00822964,
       0.12275694, 0.03001648, 0.10045702, 0.02510574, 0.05824397,
       0.00312004, 0.01016744, 0.01995344, 0.01342141, 0.09367349,
       0.00413058, 0.00617876, 0.0051371 , 0.00729736, 0.20582121]),
'param_subsample': masked_array(data=[0.21874367546954707, 0.6339685021121403,
                   0.6339685021121403, 0.7776798243640404,
                   0.7776798243640404, 0.7776798243640404,
                   0.6339685021121403, 0.21874367546954707,
                   0.4014519197059101, 0.7776798243640404,
                   0.8050495878049877, 0.7929257942422379,
                   0.43235674045068806, 0.397255377136108,
                   0.4014519197059101, 0.8934513126093477,
                   0.8934513126093477, 0.21874367546954707,
                   0.8934513126093477, 0.4406612018432605,
                   0.4406612018432605, 0.43235674045068806,
                   0.43235674045068806, 0.8050495878049877,
                   0.7929257942422379, 0.21874367546954707,
                   0.8934513126093477, 0.43235674045068806,
                   0.21874367546954707, 0.6339685021121403,
                   0.21874367546954707, 0.7929257942422379,
                   0.8050495878049877, 0.43235674045068806,
                   0.7929257942422379, 0.21874367546954707,
                   0.397255377136108, 0.7776798243640404,
                   0.4014519197059101, 0.43235674045068806,
                   0.6339685021121403, 0.397255377136108,
                   0.4406612018432605, 0.8934513126093477,
                   0.8934513126093477, 0.43235674045068806,
                   0.4014519197059101, 0.43235674045068806,
                   0.4014519197059101, 0.7929257942422379],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False],
       fill_value='?',
            dtype=object),
'param_skip_drop': masked_array(data=[0.4521417611230585, 0.882382579396944,
                   0.6850847939434404, 0.4521417611230585,
                   0.7187095900544428, 0.4521417611230585,
                   0.742612890895215, 0.4521417611230585,
                   0.742612890895215, 0.7187095900544428,
                   0.15848096425356484, 0.742612890895215,
```

```
                    0.4521417611230585, 0.6850847939434404,
                    0.7187095900544428, 0.6850847939434404,
                    0.7187095900544428, 0.7187095900544428,
                    0.6850847939434404, 0.7187095900544428,
                    0.15848096425356484, 0.882382579396944,
                    0.4521417611230585, 0.4521417611230585,
                    0.6850847939434404, 0.7660488172474501,
                    0.13386471967898056, 0.6850847939434404,
                    0.13386471967898056, 0.7187095900544428,
                    0.882382579396944, 0.6420613257291072,
                    0.6850847939434404, 0.13386471967898056,
                    0.13386471967898056, 0.6850847939434404,
                    0.882382579396944, 0.742612890895215,
                    0.882382579396944, 0.43725400817197146,
                    0.15848096425356484, 0.742612890895215,
                    0.4521417611230585, 0.7660488172474501,
                    0.13386471967898056, 0.43725400817197146,
                    0.13386471967898056, 0.882382579396944,
                    0.6420613257291072, 0.4521417611230585],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_rate_drop': masked_array(data=[0.4150587892712523, 0.630421008031628,
                    0.6696153298755829, 0.4048562507762252,
                    0.4150587892712523, 0.6696153298755829,
                    0.4048562507762252, 0.592514239837861,
                    0.9430943447179028, 0.23333976182134705,
                    0.6696153298755829, 0.14280754683425845,
                    0.6761091087458944, 0.4048562507762252,
                    0.6761091087458944, 0.14280754683425845,
                    0.7751337673546641, 0.7751337673546641,
                    0.14280754683425845, 0.7751337673546641,
                    0.4048562507762252, 0.4150587892712523,
                    0.23333976182134705, 0.7751337673546641,
                    0.592514239837861, 0.6761091087458944,
                    0.4048562507762252, 0.6761091087458944,
                    0.4048562507762252, 0.592514239837861,
                    0.4048562507762252, 0.592514239837861,
                    0.14280754683425845, 0.9430943447179028,
                    0.630421008031628, 0.592514239837861,
                    0.6761091087458944, 0.6696153298755829,
```

```
                    0.14280754683425845, 0.630421008031628,
                    0.9430943447179028, 0.9430943447179028,
                    0.7751337673546641, 0.23333976182134705,
                    0.6761091087458944, 0.4048562507762252,
                    0.9430943447179028, 0.23333976182134705,
                    0.592514239837861, 0.4150587892712523],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_n_estimator': masked_array(data=[799, 859, 379, 246, 695, 866, 311, 208,
562, 600, 16,
                    600, 757, 866, 138, 866, 797, 867, 867, 296, 695, 416,
                    379, 866, 379, 232, 797, 497, 311, 957, 600, 16, 863,
                    16, 314, 497, 208, 634, 497, 287, 311, 208, 246, 497,
                    296, 228, 224, 878, 846, 799],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_max_depth': masked_array(data=[7, 2, 7, 6, 7, 2, 7, 6, 7, 6, 6, 2, 7, 0,
6, 7, 0, 7,
                    7, 0, 0, 7, 0, 6, 6, 7, 0, 7, 6, 6, 7, 6, 6, 6, 6, 7,
                    6, 6, 6, 7, 0, 2, 6, 6, 7, 2, 0, 0, 2, 7],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_learning_rate': masked_array(data=[0.18075630780312962,
0.07054968444586057,
                    0.07054968444586057, 0.15042365197192287,
                    0.1897063329673045, 0.07054968444586057,
```

```
                    0.2617584145793186, 0.18075630780312962,
                    0.05768423482014051, 0.46431336652656724,
                    0.181767608772604, 0.1897063329673045,
                    0.2617584145793186, 0.181767608772604,
                    0.2617584145793186, 0.46431336652656724,
                    0.05768423482014051, 0.46431336652656724,
                    0.1897063329673045, 0.181767608772604,
                    0.181767608772604, 0.46431336652656724,
                    0.24024393404498523, 0.24024393404498523,
                    0.24024393404498523, 0.46431336652656724,
                    0.15042365197192287, 0.07054968444586057,
                    0.017189418902530618, 0.1897063329673045,
                    0.07054968444586057, 0.017189418902530618,
                    0.05768423482014051, 0.07054968444586057,
                    0.05768423482014051, 0.05768423482014051,
                    0.18075630780312962, 0.15042365197192287,
                    0.181767608772604, 0.15042365197192287,
                    0.017189418902530618, 0.017189418902530618,
                    0.2617584145793186, 0.181767608772604,
                    0.1897063329673045, 0.017189418902530618,
                    0.24024393404498523, 0.07054968444586057,
                    0.07054968444586057, 0.1897063329673045],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_colsample_bytree': masked_array(data=[0.6857552125215001,
0.753421581521268,
                    0.78585275861402, 0.6857552125215001,
                    0.710339669990618, 0.753421581521268,
                    0.9265785042552919, 0.753421581521268,
                    0.5399869213008575, 0.9201355494992907,
                    0.5399869213008575, 0.5946770023419758,
                    0.5946770023419758, 0.5218909349213473,
                    0.753421581521268, 0.78585275861402, 0.78585275861402,
                    0.753421581521268, 0.5218909349213473,
                    0.9201355494992907, 0.5399869213008575,
                    0.9265785042552919, 0.5946770023419758,
                    0.5946770023419758, 0.5946770023419758,
                    0.5218909349213473, 0.9201355494992907,
                    0.753421581521268, 0.6857552125215001,
                    0.7506693234415424, 0.710339669990618,
```

```
                    0.9265785042552919, 0.710339669990618,
                    0.5946770023419758, 0.9265785042552919,
                    0.7506693234415424, 0.9201355494992907,
                    0.5218909349213473, 0.710339669990618,
                    0.5218909349213473, 0.753421581521268,
                    0.5399869213008575, 0.6857552125215001,
                    0.5946770023419758, 0.6857552125215001,
                    0.5946770023419758, 0.78585275861402,
                    0.5399869213008575, 0.753421581521268,
                    0.9265785042552919],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_colsample_bynode': masked_array(data=[0.9567371209476705,
0.809429134568728,
                    0.809429134568728, 0.7631123373536698,
                    0.5980626405928076, 0.6994847591372648,
                    0.5980626405928076, 0.6994847591372648,
                    0.9573878995846159, 0.9597058019518225,
                    0.5980626405928076, 0.5980626405928076,
                    0.72017164873173, 0.9567371209476705,
                    0.9573878995846159, 0.9573878995846159,
                    0.5980626405928076, 0.5980626405928076,
                    0.9597058019518225, 0.8645338244842306,
                    0.9530637252411054, 0.7631123373536698,
                    0.5980626405928076, 0.9530637252411054,
                    0.9597058019518225, 0.8645338244842306,
                    0.809429134568728, 0.809429134568728,
                    0.809429134568728, 0.9567371209476705,
                    0.809429134568728, 0.9597058019518225,
                    0.9567371209476705, 0.6994847591372648,
                    0.6994847591372648, 0.8645338244842306,
                    0.8645338244842306, 0.72017164873173,
                    0.9567371209476705, 0.72017164873173,
                    0.9573878995846159, 0.6994847591372648,
                    0.6994847591372648, 0.809429134568728,
                    0.7631123373536698, 0.72017164873173,
                    0.9567371209476705, 0.809429134568728,
                    0.9530637252411054, 0.809429134568728],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
```

```
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False],
        fill_value='?',
             dtype=object),
 'param_colsample_bylevel': masked_array(data=[0.5889946405144686,
0.8621252435139835,
                  0.6854717478419741, 0.669479056483163,
                  0.6854717478419741, 0.669479056483163,
                  0.6854717478419741, 0.8849989076709623,
                  0.669479056483163, 0.669479056483163,
                  0.8849989076709623, 0.5889946405144686,
                  0.7537839481281112, 0.669479056483163,
                  0.6854717478419741, 0.6276033033890085,
                  0.6848027987759879, 0.7537839481281112,
                  0.5889946405144686, 0.5889946405144686,
                  0.669479056483163, 0.9224103840543187,
                  0.6276033033890085, 0.9013009259760407,
                  0.8621252435139835, 0.8621252435139835,
                  0.9224103840543187, 0.9224103840543187,
                  0.9013009259760407, 0.7537839481281112,
                  0.6848027987759879, 0.9224103840543187,
                  0.6854717478419741, 0.7537839481281112,
                  0.6854717478419741, 0.9013009259760407,
                  0.9224103840543187, 0.8621252435139835,
                  0.6854717478419741, 0.8849989076709623,
                  0.7537839481281112, 0.5889946405144686,
                  0.8849989076709623, 0.6854717478419741,
                  0.669479056483163, 0.7537839481281112,
                  0.7537839481281112, 0.6854717478419741,
                  0.8621252435139835, 0.669479056483163],
            mask=[False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False],
        fill_value='?',
             dtype=object),
 'params': [{'subsample': 0.21874367546954707,
   'skip_drop': 0.4521417611230585,
   'rate_drop': 0.4150587892712523,
   'n_estimator': 799,
   'max_depth': 7,
```

```
 'learning_rate': 0.18075630780312962,
 'colsample_bytree': 0.6857552125215001,
 'colsample_bynode': 0.9567371209476705,
 'colsample_bylevel': 0.5889946405144686},
{'subsample': 0.6339685021121403,
 'skip_drop': 0.882382579396944,
 'rate_drop': 0.630421008031628,
 'n_estimator': 859,
 'max_depth': 2,
 'learning_rate': 0.07054968444586057,
 'colsample_bytree': 0.753421581521268,
 'colsample_bynode': 0.809429134568728,
 'colsample_bylevel': 0.8621252435139835},
{'subsample': 0.6339685021121403,
 'skip_drop': 0.6850847939434404,
 'rate_drop': 0.6696153298755829,
 'n_estimator': 379,
 'max_depth': 7,
 'learning_rate': 0.07054968444586057,
 'colsample_bytree': 0.78585275861402,
 'colsample_bynode': 0.809429134568728,
 'colsample_bylevel': 0.6854717478419741},
{'subsample': 0.7776798243640404,
 'skip_drop': 0.4521417611230585,
 'rate_drop': 0.4048562507762252,
 'n_estimator': 246,
 'max_depth': 6,
 'learning_rate': 0.15042365197192287,
 'colsample_bytree': 0.6857552125215001,
 'colsample_bynode': 0.7631123373536698,
 'colsample_bylevel': 0.669479056483163},
{'subsample': 0.7776798243640404,
 'skip_drop': 0.7187095900544428,
 'rate_drop': 0.4150587892712523,
 'n_estimator': 695,
 'max_depth': 7,
 'learning_rate': 0.1897063329673045,
 'colsample_bytree': 0.710339669990618,
 'colsample_bynode': 0.5980626405928076,
 'colsample_bylevel': 0.6854717478419741},
{'subsample': 0.7776798243640404,
 'skip_drop': 0.4521417611230585,
 'rate_drop': 0.6696153298755829,
 'n_estimator': 866,
 'max_depth': 2,
 'learning_rate': 0.07054968444586057,
 'colsample_bytree': 0.753421581521268,
```

```
 'colsample_bynode': 0.6994847591372648,
 'colsample_bylevel': 0.669479056483163},
{'subsample': 0.6339685021121403,
 'skip_drop': 0.742612890895215,
 'rate_drop': 0.4048562507762252,
 'n_estimator': 311,
 'max_depth': 7,
 'learning_rate': 0.2617584145793186,
 'colsample_bytree': 0.9265785042552919,
 'colsample_bynode': 0.5980626405928076,
 'colsample_bylevel': 0.6854717478419741},
{'subsample': 0.21874367546954707,
 'skip_drop': 0.4521417611230585,
 'rate_drop': 0.592514239837861,
 'n_estimator': 208,
 'max_depth': 6,
 'learning_rate': 0.18075630780312962,
 'colsample_bytree': 0.753421581521268,
 'colsample_bynode': 0.6994847591372648,
 'colsample_bylevel': 0.8849989076709623},
{'subsample': 0.4014519197059101,
 'skip_drop': 0.742612890895215,
 'rate_drop': 0.9430943447179028,
 'n_estimator': 562,
 'max_depth': 7,
 'learning_rate': 0.05768423482014051,
 'colsample_bytree': 0.5399869213008575,
 'colsample_bynode': 0.9573878995846159,
 'colsample_bylevel': 0.669479056483163},
{'subsample': 0.7776798243640404,
 'skip_drop': 0.7187095900544428,
 'rate_drop': 0.23333976182134705,
 'n_estimator': 600,
 'max_depth': 6,
 'learning_rate': 0.46431336652656724,
 'colsample_bytree': 0.9201355494992907,
 'colsample_bynode': 0.9597058019518225,
 'colsample_bylevel': 0.669479056483163},
{'subsample': 0.8050495878049877,
 'skip_drop': 0.15848096425356484,
 'rate_drop': 0.6696153298755829,
 'n_estimator': 16,
 'max_depth': 6,
 'learning_rate': 0.181767608772604,
 'colsample_bytree': 0.5399869213008575,
 'colsample_bynode': 0.5980626405928076,
 'colsample_bylevel': 0.8849989076709623},
```

```
{'subsample': 0.7929257942422379,
 'skip_drop': 0.742612890895215,
 'rate_drop': 0.14280754683425845,
 'n_estimator': 600,
 'max_depth': 2,
 'learning_rate': 0.1897063329673045,
 'colsample_bytree': 0.5946770023419758,
 'colsample_bynode': 0.5980626405928076,
 'colsample_bylevel': 0.5889946405144686},
{'subsample': 0.43235674045068806,
 'skip_drop': 0.4521417611230585,
 'rate_drop': 0.6761091087458944,
 'n_estimator': 757,
 'max_depth': 7,
 'learning_rate': 0.2617584145793186,
 'colsample_bytree': 0.5946770023419758,
 'colsample_bynode': 0.72017164873173,
 'colsample_bylevel': 0.7537839481281112},
{'subsample': 0.397255377136108,
 'skip_drop': 0.6850847939434404,
 'rate_drop': 0.4048562507762252,
 'n_estimator': 866,
 'max_depth': 0,
 'learning_rate': 0.181767608772604,
 'colsample_bytree': 0.5218909349213473,
 'colsample_bynode': 0.9567371209476705,
 'colsample_bylevel': 0.669479056483163},
{'subsample': 0.4014519197059101,
 'skip_drop': 0.7187095900544428,
 'rate_drop': 0.6761091087458944,
 'n_estimator': 138,
 'max_depth': 6,
 'learning_rate': 0.2617584145793186,
 'colsample_bytree': 0.753421581521268,
 'colsample_bynode': 0.9573878995846159,
 'colsample_bylevel': 0.6854717478419741},
{'subsample': 0.8934513126093477,
 'skip_drop': 0.6850847939434404,
 'rate_drop': 0.14280754683425845,
 'n_estimator': 866,
 'max_depth': 7,
 'learning_rate': 0.46431336652656724,
 'colsample_bytree': 0.78585275861402,
 'colsample_bynode': 0.9573878995846159,
 'colsample_bylevel': 0.6276033033890085},
{'subsample': 0.8934513126093477,
 'skip_drop': 0.7187095900544428,
```

```
 'rate_drop': 0.7751337673546641,
 'n_estimator': 797,
 'max_depth': 0,
 'learning_rate': 0.05768423482014051,
 'colsample_bytree': 0.78585275861402,
 'colsample_bynode': 0.5980626405928076,
 'colsample_bylevel': 0.6848027987759879},
{'subsample': 0.21874367546954707,
 'skip_drop': 0.7187095900544428,
 'rate_drop': 0.7751337673546641,
 'n_estimator': 867,
 'max_depth': 7,
 'learning_rate': 0.46431336652656724,
 'colsample_bytree': 0.753421581521268,
 'colsample_bynode': 0.5980626405928076,
 'colsample_bylevel': 0.7537839481281112},
{'subsample': 0.8934513126093477,
 'skip_drop': 0.6850847939434404,
 'rate_drop': 0.14280754683425845,
 'n_estimator': 867,
 'max_depth': 7,
 'learning_rate': 0.1897063329673045,
 'colsample_bytree': 0.5218909349213473,
 'colsample_bynode': 0.9597058019518225,
 'colsample_bylevel': 0.5889946405144686},
{'subsample': 0.4406612018432605,
 'skip_drop': 0.7187095900544428,
 'rate_drop': 0.7751337673546641,
 'n_estimator': 296,
 'max_depth': 0,
 'learning_rate': 0.181767608772604,
 'colsample_bytree': 0.9201355494992907,
 'colsample_bynode': 0.8645338244842306,
 'colsample_bylevel': 0.5889946405144686},
{'subsample': 0.4406612018432605,
 'skip_drop': 0.15848096425356484,
 'rate_drop': 0.4048562507762252,
 'n_estimator': 695,
 'max_depth': 0,
 'learning_rate': 0.181767608772604,
 'colsample_bytree': 0.5399869213008575,
 'colsample_bynode': 0.9530637252411054,
 'colsample_bylevel': 0.669479056483163},
{'subsample': 0.43235674045068806,
 'skip_drop': 0.882382579396944,
 'rate_drop': 0.4150587892712523,
 'n_estimator': 416,
```

```
 'max_depth': 7,
 'learning_rate': 0.46431336652656724,
 'colsample_bytree': 0.9265785042552919,
 'colsample_bynode': 0.7631123373536698,
 'colsample_bylevel': 0.9224103840543187},
{'subsample': 0.43235674045068806,
 'skip_drop': 0.4521417611230585,
 'rate_drop': 0.23333976182134705,
 'n_estimator': 379,
 'max_depth': 0,
 'learning_rate': 0.24024393404498523,
 'colsample_bytree': 0.5946770023419758,
 'colsample_bynode': 0.5980626405928076,
 'colsample_bylevel': 0.6276033033890085},
{'subsample': 0.8050495878049877,
 'skip_drop': 0.4521417611230585,
 'rate_drop': 0.7751337673546641,
 'n_estimator': 866,
 'max_depth': 6,
 'learning_rate': 0.24024393404498523,
 'colsample_bytree': 0.5946770023419758,
 'colsample_bynode': 0.9530637252411054,
 'colsample_bylevel': 0.9013009259760407},
{'subsample': 0.7929257942422379,
 'skip_drop': 0.6850847939434404,
 'rate_drop': 0.592514239837861,
 'n_estimator': 379,
 'max_depth': 6,
 'learning_rate': 0.24024393404498523,
 'colsample_bytree': 0.5946770023419758,
 'colsample_bynode': 0.9597058019518225,
 'colsample_bylevel': 0.8621252435139835},
{'subsample': 0.21874367546954707,
 'skip_drop': 0.7660488172474501,
 'rate_drop': 0.6761091087458944,
 'n_estimator': 232,
 'max_depth': 7,
 'learning_rate': 0.46431336652656724,
 'colsample_bytree': 0.5218909349213473,
 'colsample_bynode': 0.8645338244842306,
 'colsample_bylevel': 0.8621252435139835},
{'subsample': 0.8934513126093477,
 'skip_drop': 0.13386471967898056,
 'rate_drop': 0.4048562507762252,
 'n_estimator': 797,
 'max_depth': 0,
 'learning_rate': 0.15042365197192287,
```

```
 'colsample_bytree': 0.9201355494992907,
 'colsample_bynode': 0.809429134568728,
 'colsample_bylevel': 0.9224103840543187},
{'subsample': 0.43235674045068806,
 'skip_drop': 0.6850847939434404,
 'rate_drop': 0.6761091087458944,
 'n_estimator': 497,
 'max_depth': 7,
 'learning_rate': 0.07054968444586057,
 'colsample_bytree': 0.753421581521268,
 'colsample_bynode': 0.809429134568728,
 'colsample_bylevel': 0.9224103840543187},
{'subsample': 0.21874367546954707,
 'skip_drop': 0.13386471967898056,
 'rate_drop': 0.4048562507762252,
 'n_estimator': 311,
 'max_depth': 6,
 'learning_rate': 0.017189418902530618,
 'colsample_bytree': 0.6857552125215001,
 'colsample_bynode': 0.809429134568728,
 'colsample_bylevel': 0.9013009259760407},
{'subsample': 0.6339685021121403,
 'skip_drop': 0.7187095900544428,
 'rate_drop': 0.592514239837861,
 'n_estimator': 957,
 'max_depth': 6,
 'learning_rate': 0.1897063329673045,
 'colsample_bytree': 0.7506693234415424,
 'colsample_bynode': 0.9567371209476705,
 'colsample_bylevel': 0.7537839481281112},
{'subsample': 0.21874367546954707,
 'skip_drop': 0.882382579396944,
 'rate_drop': 0.4048562507762252,
 'n_estimator': 600,
 'max_depth': 7,
 'learning_rate': 0.07054968444586057,
 'colsample_bytree': 0.710339669990618,
 'colsample_bynode': 0.809429134568728,
 'colsample_bylevel': 0.6848027987759879},
{'subsample': 0.7929257942422379,
 'skip_drop': 0.6420613257291072,
 'rate_drop': 0.592514239837861,
 'n_estimator': 16,
 'max_depth': 6,
 'learning_rate': 0.017189418902530618,
 'colsample_bytree': 0.9265785042552919,
 'colsample_bynode': 0.9597058019518225,
```

```
 'colsample_bylevel': 0.9224103840543187},
{'subsample': 0.8050495878049877,
 'skip_drop': 0.6850847939434404,
 'rate_drop': 0.14280754683425845,
 'n_estimator': 863,
 'max_depth': 6,
 'learning_rate': 0.05768423482014051,
 'colsample_bytree': 0.710339669990618,
 'colsample_bynode': 0.9567371209476705,
 'colsample_bylevel': 0.6854717478419741},
{'subsample': 0.43235674045068806,
 'skip_drop': 0.13386471967898056,
 'rate_drop': 0.9430943447179028,
 'n_estimator': 16,
 'max_depth': 6,
 'learning_rate': 0.07054968444586057,
 'colsample_bytree': 0.5946770023419758,
 'colsample_bynode': 0.6994847591372648,
 'colsample_bylevel': 0.7537839481281112},
{'subsample': 0.7929257942422379,
 'skip_drop': 0.13386471967898056,
 'rate_drop': 0.630421008031628,
 'n_estimator': 314,
 'max_depth': 6,
 'learning_rate': 0.05768423482014051,
 'colsample_bytree': 0.9265785042552919,
 'colsample_bynode': 0.6994847591372648,
 'colsample_bylevel': 0.6854717478419741},
{'subsample': 0.21874367546954707,
 'skip_drop': 0.6850847939434404,
 'rate_drop': 0.592514239837861,
 'n_estimator': 497,
 'max_depth': 7,
 'learning_rate': 0.05768423482014051,
 'colsample_bytree': 0.7506693234415424,
 'colsample_bynode': 0.8645338244842306,
 'colsample_bylevel': 0.9013009259760407},
{'subsample': 0.397255377136108,
 'skip_drop': 0.882382579396944,
 'rate_drop': 0.6761091087458944,
 'n_estimator': 208,
 'max_depth': 6,
 'learning_rate': 0.18075630780312962,
 'colsample_bytree': 0.9201355494992907,
 'colsample_bynode': 0.8645338244842306,
 'colsample_bylevel': 0.9224103840543187},
{'subsample': 0.7776798243640404,
```

```
 'skip_drop': 0.742612890895215,
 'rate_drop': 0.6696153298755829,
 'n_estimator': 634,
 'max_depth': 6,
 'learning_rate': 0.15042365197192287,
 'colsample_bytree': 0.5218909349213473,
 'colsample_bynode': 0.72017164873173,
 'colsample_bylevel': 0.8621252435139835},
{'subsample': 0.4014519197059101,
 'skip_drop': 0.882382579396944,
 'rate_drop': 0.14280754683425845,
 'n_estimator': 497,
 'max_depth': 6,
 'learning_rate': 0.181767608772604,
 'colsample_bytree': 0.710339669990618,
 'colsample_bynode': 0.9567371209476705,
 'colsample_bylevel': 0.6854717478419741},
{'subsample': 0.43235674045068806,
 'skip_drop': 0.43725400817197146,
 'rate_drop': 0.630421008031628,
 'n_estimator': 287,
 'max_depth': 7,
 'learning_rate': 0.15042365197192287,
 'colsample_bytree': 0.5218909349213473,
 'colsample_bynode': 0.72017164873173,
 'colsample_bylevel': 0.8849989076709623},
{'subsample': 0.6339685021121403,
 'skip_drop': 0.15848096425356484,
 'rate_drop': 0.9430943447179028,
 'n_estimator': 311,
 'max_depth': 0,
 'learning_rate': 0.017189418902530618,
 'colsample_bytree': 0.753421581521268,
 'colsample_bynode': 0.9573878995846159,
 'colsample_bylevel': 0.7537839481281112},
{'subsample': 0.397255377136108,
 'skip_drop': 0.742612890895215,
 'rate_drop': 0.9430943447179028,
 'n_estimator': 208,
 'max_depth': 2,
 'learning_rate': 0.017189418902530618,
 'colsample_bytree': 0.5399869213008575,
 'colsample_bynode': 0.6994847591372648,
 'colsample_bylevel': 0.5889946405144686},
{'subsample': 0.4406612018432605,
 'skip_drop': 0.4521417611230585,
 'rate_drop': 0.7751337673546641,
```

```
 'n_estimator': 246,
 'max_depth': 6,
 'learning_rate': 0.2617584145793186,
 'colsample_bytree': 0.6857552125215001,
 'colsample_bynode': 0.6994847591372648,
 'colsample_bylevel': 0.8849989076709623},
{'subsample': 0.8934513126093477,
 'skip_drop': 0.7660488172474501,
 'rate_drop': 0.23333976182134705,
 'n_estimator': 497,
 'max_depth': 6,
 'learning_rate': 0.181767608772604,
 'colsample_bytree': 0.5946770023419758,
 'colsample_bynode': 0.809429134568728,
 'colsample_bylevel': 0.6854717478419741},
{'subsample': 0.8934513126093477,
 'skip_drop': 0.13386471967898056,
 'rate_drop': 0.6761091087458944,
 'n_estimator': 296,
 'max_depth': 7,
 'learning_rate': 0.1897063329673045,
 'colsample_bytree': 0.6857552125215001,
 'colsample_bynode': 0.7631123373536698,
 'colsample_bylevel': 0.669479056483163},
{'subsample': 0.43235674045068806,
 'skip_drop': 0.43725400817197146,
 'rate_drop': 0.4048562507762252,
 'n_estimator': 228,
 'max_depth': 2,
 'learning_rate': 0.017189418902530618,
 'colsample_bytree': 0.5946770023419758,
 'colsample_bynode': 0.72017164873173,
 'colsample_bylevel': 0.7537839481281112},
{'subsample': 0.4014519197059101,
 'skip_drop': 0.13386471967898056,
 'rate_drop': 0.9430943447179028,
 'n_estimator': 224,
 'max_depth': 0,
 'learning_rate': 0.24024393404498523,
 'colsample_bytree': 0.78585275861402,
 'colsample_bynode': 0.9567371209476705,
 'colsample_bylevel': 0.7537839481281112},
{'subsample': 0.43235674045068806,
 'skip_drop': 0.882382579396944,
 'rate_drop': 0.23333976182134705,
 'n_estimator': 878,
 'max_depth': 0,
```

```
   'learning_rate': 0.07054968444586057,
   'colsample_bytree': 0.5399869213008575,
   'colsample_bynode': 0.809429134568728,
   'colsample_bylevel': 0.6854717478419741},
  {'subsample': 0.4014519197059101,
   'skip_drop': 0.6420613257291072,
   'rate_drop': 0.592514239837861,
   'n_estimator': 846,
   'max_depth': 2,
   'learning_rate': 0.07054968444586057,
   'colsample_bytree': 0.753421581521268,
   'colsample_bynode': 0.9530637252411054,
   'colsample_bylevel': 0.8621252435139835},
  {'subsample': 0.7929257942422379,
   'skip_drop': 0.4521417611230585,
   'rate_drop': 0.4150587892712523,
   'n_estimator': 799,
   'max_depth': 7,
   'learning_rate': 0.1897063329673045,
   'colsample_bytree': 0.9265785042552919,
   'colsample_bynode': 0.809429134568728,
   'colsample_bylevel': 0.669479056483163}],
 'split0_test_AUC': array([0.80025439, 0.75255002, 0.80046951, 0.79576346,
0.8051774 ,
        0.73683384, 0.80808705, 0.78854428, 0.7978099 , 0.80919971,
        0.76905176, 0.76834547, 0.79027386, 0.5       , 0.79592188,
        0.81510674, 0.5       , 0.79199783, 0.81134982, 0.5       ,
        0.5       , 0.81019031, 0.5       , 0.78756129, 0.79607567,
        0.79804025, 0.5       , 0.80164866, 0.779686  , 0.79821223,
        0.80469167, 0.78040218, 0.80100947, 0.77350385, 0.77769394,
        0.80122328, 0.80385422, 0.79621297, 0.81006488, 0.79329729,
        0.5       , 0.72565445, 0.7871614 , 0.80777152, 0.78018605,
        0.72934477, 0.5       , 0.5       , 0.7445744 , 0.79935804]),
 'split1_test_AUC': array([0.8017586 , 0.75220819, 0.80190697, 0.79663849,
0.80690425,
        0.73744546, 0.8098556 , 0.78935099, 0.79921963, 0.81101133,
        0.76847065, 0.76910873, 0.79181089, 0.5       , 0.79771139,
        0.81829999, 0.5       , 0.79289886, 0.8130579 , 0.5       ,
        0.5       , 0.81161458, 0.5       , 0.78760244, 0.79777317,
        0.79820747, 0.5       , 0.80345904, 0.78032937, 0.79995826,
        0.80612972, 0.78149143, 0.80291283, 0.77362469, 0.77882396,
        0.80254495, 0.80551306, 0.79863043, 0.81221358, 0.79312691,
        0.5       , 0.72752484, 0.78672223, 0.81019148, 0.78138027,
        0.73037015, 0.5       , 0.5       , 0.74499563, 0.80129449]),
 'split2_test_AUC': array([0.80045744, 0.75235888, 0.80152943, 0.79672777,
0.80670263,
        0.73510033, 0.81022964, 0.79016448, 0.798565  , 0.80961213,
```

```
        0.76973625, 0.76991759, 0.79079213, 0.5       , 0.79727576,
        0.81660158, 0.5       , 0.79219711, 0.81257519, 0.5       ,
        0.5       , 0.81052832, 0.5       , 0.78745251, 0.79677382,
        0.79814161, 0.5       , 0.80265006, 0.78094695, 0.79836994,
        0.80546175, 0.78157281, 0.80203574, 0.77353228, 0.77882546,
        0.80181256, 0.80406272, 0.7978507 , 0.81039237, 0.79317611,
        0.5       , 0.72652621, 0.78724797, 0.80851043, 0.78230273,
        0.72959227, 0.5       , 0.5       , 0.74521111, 0.80053269]),
 'mean_test_AUC': array([0.80082348, 0.75237236, 0.80130197, 0.79637657,
0.80626143,
        0.73645988, 0.80939076, 0.78935325, 0.79853151, 0.80994106,
        0.76908622, 0.76912393, 0.79095896, 0.5       , 0.79696968,
        0.81666944, 0.5       , 0.7923646 , 0.81232764, 0.5       ,
        0.5       , 0.81077774, 0.5       , 0.78753875, 0.79687422,
        0.79812978, 0.5       , 0.80258592, 0.78032077, 0.79884681,
        0.80542771, 0.78115547, 0.80198602, 0.7735536 , 0.77844779,
        0.80186026, 0.80447667, 0.7975647 , 0.81089028, 0.7932001 ,
        0.5       , 0.7265685 , 0.78704387, 0.80882447, 0.78128968,
        0.72976907, 0.5       , 0.5       , 0.74492704, 0.80039507]),
 'std_test_AUC': array([6.66405008e-04, 1.39875264e-04, 6.08482461e-04,
4.35065598e-04,
        7.70928764e-04, 9.93242695e-04, 9.34425795e-04, 6.61447665e-04,
        5.76006302e-04, 7.75301370e-04, 5.17255610e-04, 6.41905999e-04,
        6.38484280e-04, 0.00000000e+00, 7.61950103e-04, 1.30452147e-03,
        0.00000000e+00, 3.86441517e-04, 7.18953319e-04, 0.00000000e+00,
        0.00000000e+00, 6.07612308e-04, 0.00000000e+00, 6.32511181e-05,
        6.96629505e-04, 6.87795963e-05, 0.00000000e+00, 7.40471660e-04,
        5.14818162e-04, 7.88545115e-04, 5.87574295e-04, 5.33692567e-04,
        7.77839334e-04, 5.15859726e-05, 5.33048475e-04, 5.40625171e-04,
        7.37768843e-04, 1.00742987e-03, 9.45221836e-04, 7.15941381e-05,
        0.00000000e+00, 7.64168358e-04, 2.30158042e-04, 1.01259245e-03,
        8.66502323e-04, 4.36876842e-04, 0.00000000e+00, 0.00000000e+00,
        2.64422080e-04, 7.96521484e-04]),
 'rank_test_AUC': array([15, 37, 14, 23,  8, 39,  6, 27, 18,  5, 36, 35, 26, 42,
21,  1, 42,
        25,  2, 42, 42,  4, 42, 28, 22, 19, 42, 11, 32, 17,  9, 31, 12, 34,
        33, 13, 10, 20,  3, 24, 42, 41, 29,  7, 30, 40, 42, 42, 38, 16],
       dtype=int32),
 'split0_test_Accuracy': array([0.7219204 , 0.67628358, 0.72217413, 0.71805473,
0.72551244,
        0.65995522, 0.72842289, 0.71153731, 0.71963682, 0.72872139,
        0.69537313, 0.69610448, 0.71361692, 0.5300398 , 0.71840796,
        0.73516915, 0.5300398 , 0.71439303, 0.73151741, 0.5300398 ,
        0.5300398 , 0.73025373, 0.5300398 , 0.71223881, 0.71833333,
        0.72022886, 0.5300398 , 0.72343284, 0.70537313, 0.7198806 ,
        0.72546766, 0.70703483, 0.7218209 , 0.70140796, 0.7018607 ,
        0.72272139, 0.72444279, 0.71789552, 0.73061692, 0.71586567,
```

```
        0.5300398 , 0.64391045, 0.71114428, 0.72794527, 0.70510945,
        0.65132836, 0.5300398 , 0.5300398 , 0.66890547, 0.72053234]),
 'split1_test_Accuracy': array([0.72251741, 0.67579104, 0.72314428, 0.7179403 ,
0.72740299,
        0.66062687, 0.72974129, 0.71240796, 0.72023383, 0.73137811,
        0.69442289, 0.69702985, 0.71469652, 0.5300398 , 0.72044279,
        0.7369602 , 0.5300398 , 0.71642786, 0.73296517, 0.5300398 ,
        0.5300398 , 0.73163682, 0.5300398 , 0.710801  , 0.71914428,
        0.72085075, 0.5300398 , 0.72492537, 0.70506468, 0.72195522,
        0.72671642, 0.7070995 , 0.7240597 , 0.7019403 , 0.70314925,
        0.72332836, 0.72538806, 0.72058706, 0.73271144, 0.71478607,
        0.5300398 , 0.64735821, 0.70995522, 0.730199  , 0.70574627,
        0.65241294, 0.5300398 , 0.5300398 , 0.67068159, 0.72195025]),
 'split2_test_Accuracy': array([0.72274129, 0.6758209 , 0.72362687, 0.71910448,
0.72770149,
        0.65732338, 0.73064179, 0.71435323, 0.72133333, 0.7291194 ,
        0.69624378, 0.69721393, 0.71481095, 0.5300398 , 0.71945771,
        0.73617413, 0.5300398 , 0.71487562, 0.73301493, 0.5300398 ,
        0.5300398 , 0.73052736, 0.5300398 , 0.71163682, 0.72065174,
        0.72057214, 0.5300398 , 0.72464179, 0.70771144, 0.72056716,
        0.72662189, 0.70839303, 0.72337811, 0.70293035, 0.7051194 ,
        0.72371144, 0.72503483, 0.72006468, 0.73062687, 0.71665672,
        0.5300398 , 0.64147761, 0.71142786, 0.72944279, 0.70800995,
        0.65076617, 0.5300398 , 0.5300398 , 0.66974627, 0.72260697]),
 'mean_test_Accuracy': array([0.72239303, 0.67596517, 0.72298176, 0.7183665 ,
0.72687231,
        0.65930182, 0.72960199, 0.71276617, 0.72040133, 0.72973964,
        0.6953466 , 0.69678275, 0.71437479, 0.5300398 , 0.71943615,
        0.73610116, 0.5300398 , 0.71523217, 0.73249917, 0.5300398 ,
        0.5300398 , 0.73080597, 0.5300398 , 0.71155887, 0.71937645,
        0.72055058, 0.5300398 , 0.72433333, 0.70604975, 0.720801  ,
        0.72626866, 0.70750912, 0.72308624, 0.70209287, 0.70337645,
        0.72325373, 0.72495522, 0.71951575, 0.73131841, 0.71576949,
        0.5300398 , 0.64424876, 0.71084245, 0.72919569, 0.70628856,
        0.65150249, 0.5300398 , 0.5300398 , 0.66977778, 0.72169652]),
 'std_test_Accuracy': array([0.00034648, 0.00022548, 0.00060411, 0.00052392,
0.00096926,
        0.00142559, 0.0009112 , 0.00117717, 0.00070265, 0.00116991,
        0.00074361, 0.00048546, 0.00053793, 0.        , 0.00083085,
        0.00073301, 0.        , 0.00086813, 0.0006945 , 0.        ,
        0.        , 0.00059802, 0.        , 0.00058957, 0.00096062,
        0.00025434, 0.        , 0.00064719, 0.00118172, 0.00086295,
        0.0005677 , 0.00062558, 0.000937  , 0.00063081, 0.00134003,
        0.00040762, 0.00038999, 0.00116536, 0.00098503, 0.00076671,
        0.        , 0.00241263, 0.00063796, 0.00093653, 0.00124466,
        0.00068347, 0.        , 0.        , 0.00072544, 0.00086576]),
 'rank_test_Accuracy': array([15, 37, 14, 23,  8, 39,  6, 27, 19,  5, 36, 35,
```

```
26, 42, 21,  1, 42,
        25,  2, 42, 42,  4, 42, 28, 22, 18, 42, 11, 32, 17,  9, 30, 13, 34,
        33, 12, 10, 20,  3, 24, 42, 41, 29,  7, 31, 40, 42, 42, 38, 16],
       dtype=int32)}
```

[37]:
```python
best_model.best_estimator_
```

[37]:
```
XGBClassifier(base_score=0.5, booster='dart',
              colsample_bylevel=0.6276033033890085,
              colsample_bynode=0.9573878995846159,
              colsample_bytree=0.78585275861402, eval_metric=['error', 'auc'],
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.46431336652656724,
              max_delta_step=0, max_depth=7, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimator=866, n_estimators=100,
              n_jobs=-1, normalize_type='forest', num_parallel_tree=1,
              random_state=42, rate_drop=0.14280754683425845, reg_alpha=0,
              reg_lambda=1, sample_type='weighted', scale_pos_weight=1,
              skip_drop=0.6850847939434404, subsample=0.8934513126093477,
              tree_method='exact', …)
```

[46]:
```python
best_model.best_score_
```

[46]:
```
0.8166694389581771
```

[47]:
```python
store_best_paramters = best_model.best_params_
```

[48]:
```python
best_parameters = best_model.best_params_
best_parameters
```

[48]:
```
{'subsample': 0.8934513126093477,
 'skip_drop': 0.6850847939434404,
 'rate_drop': 0.14280754683425845,
 'n_estimator': 866,
 'max_depth': 7,
 'learning_rate': 0.46431336652656724,
 'colsample_bytree': 0.78585275861402,
 'colsample_bynode': 0.9573878995846159,
 'colsample_bylevel': 0.6276033033890085}
```

[43]:
```python
eval_set=[(X_val,y_val)]
best_XGB_model = XGBClassifier(base_score=0.5, booster='dart',
              colsample_bylevel=0.6276033033890085,
              colsample_bynode=0.9573878995846159,
              colsample_bytree=0.78585275861402, eval_metric=['error', 'auc'],
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.46431336652656724,
```

```
                max_delta_step=0, max_depth=7, min_child_weight=1,
                monotone_constraints='()', n_estimators=100,
                n_jobs=-1, normalize_type='forest', num_parallel_tree=1,
                random_state=42, rate_drop=0.14280754683425845, reg_alpha=0,
                reg_lambda=1, sample_type='weighted', scale_pos_weight=1,
                skip_drop=0.6850847939434404, subsample=0.8934513126093477,
                tree_method='exact',eval_set=eval_set, early_stopping_rounds=10)

best_XGB_model.fit(X_train,y_train)
y_pred = best_XGB_model.predict(X_val)
confusion_matrix(y_val,y_pred)
accuracy_score(y_val,y_pred)
#classification_report(y_val,y_pred, output_dict=True)
```

```
/opt/anaconda3/lib/python3.8/site-packages/xgboost/sklearn.py:888: UserWarning:
The use of label encoder in XGBClassifier is deprecated and will be removed in a
future release. To remove this warning, do the following: 1) Pass option
use_label_encoder=False when constructing XGBClassifier object; and 2) Encode
your labels (y) as integers starting with 0, i.e. 0, 1, 2, …, [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

```
[20:01:22] WARNING: /Users/travis/build/dmlc/xgboost/src/learner.cc:541:
Parameters: { early_stopping_round, eval_set } might not be used.

  This may not be accurate due to some parameters are only used in language
bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through
this
  verification. Please open an issue if you find above cases.
```

[43]: 0.7384781144781145

[49]:
```
y_pred = best_model.predict(X_val)
```

[50]:
```
accuracy_score(y_val,y_pred)
```

[50]: 0.7384781144781145

[51]:
```
confusion_matrix(y_val,y_pred)
```

[51]:
```
array([[ 99372,  40462],
       [ 37210, 119956]])
```

[52]:
```
from sklearn.metrics import roc_curve
fpr, tpr, threshold = roc_curve(y_val,y_pred)
auc(fpr,tpr)
```

[52]: 0.7369432888612573

[58]:
```
from sklearn.metrics import classification_report
classification_report(y_val,y_pred, output_dict=True)
```

[58]:
```
{'0.0': {'precision': 0.7275629292293274,
  'recall': 0.7106426191055109,
  'f1-score': 0.7190032414910859,
  'support': 139834},
 '1.0': {'precision': 0.7477714470944657,
  'recall': 0.7632439586170037,
  'f1-score': 0.7554284850622198,
  'support': 157166},
 'accuracy': 0.7384781144781145,
 'macro avg': {'precision': 0.7376671881618966,
  'recall': 0.7369432888612573,
  'f1-score': 0.7372158632766528,
  'support': 297000},
 'weighted avg': {'precision': 0.7382568414138133,
  'recall': 0.7384781144781145,
  'f1-score': 0.7382786954678564,
  'support': 297000}}
```

Plot the ROC (Receiver Operating Characteristic) Curve (more info on ROC could be found here

[54]:
```
pip install plot-metric
```

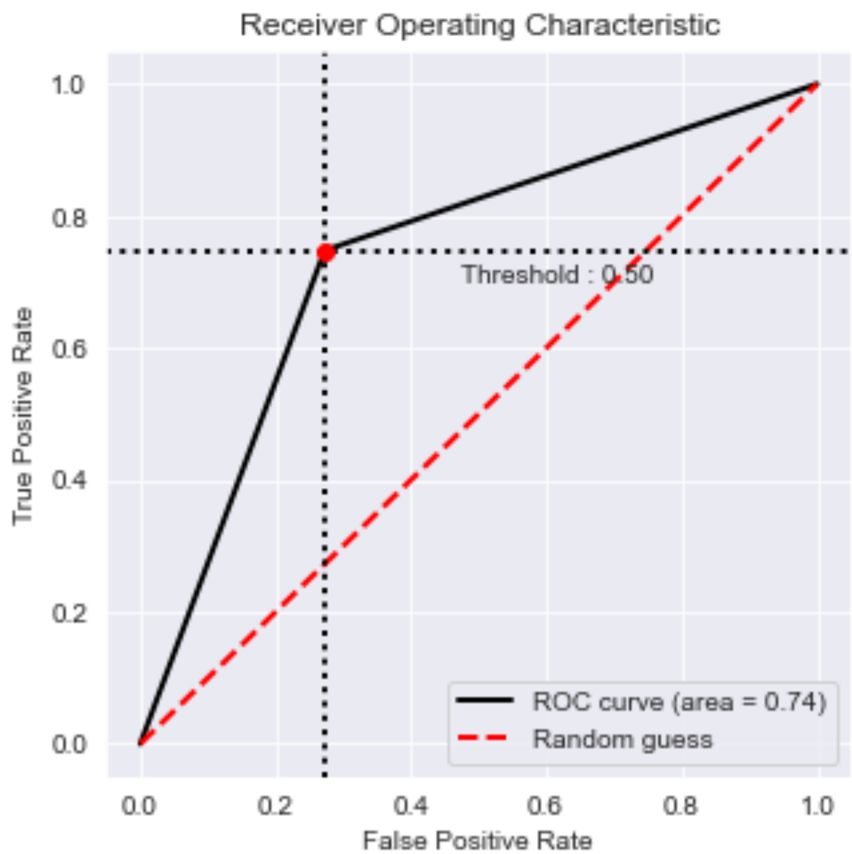
```
Collecting plot-metric
  Downloading plot_metric-0.0.6-py3-none-any.whl (13 kB)
Requirement already satisfied: pandas>=0.23.4 in
/opt/anaconda3/lib/python3.8/site-packages (from plot-metric) (1.1.3)
Requirement already satisfied: scikit-learn>=0.21.2 in
/opt/anaconda3/lib/python3.8/site-packages (from plot-metric) (0.23.2)
Requirement already satisfied: matplotlib>=3.0.2 in
/opt/anaconda3/lib/python3.8/site-packages (from plot-metric) (3.3.2)
Collecting colorlover>=0.3.0
  Downloading colorlover-0.3.0-py3-none-any.whl (8.9 kB)
Requirement already satisfied: scipy>=1.1.0 in
/opt/anaconda3/lib/python3.8/site-packages (from plot-metric) (1.5.2)
Requirement already satisfied: numpy>=1.15.4 in
/opt/anaconda3/lib/python3.8/site-packages (from plot-metric) (1.19.2)
Requirement already satisfied: seaborn>=0.9.0 in
/opt/anaconda3/lib/python3.8/site-packages (from plot-metric) (0.11.0)
Requirement already satisfied: python-dateutil>=2.7.3 in
/opt/anaconda3/lib/python3.8/site-packages (from pandas>=0.23.4->plot-metric)
(2.8.1)
Requirement already satisfied: pytz>=2017.2 in
```

```
/opt/anaconda3/lib/python3.8/site-packages (from pandas>=0.23.4->plot-metric)
(2020.1)
Requirement already satisfied: joblib>=0.11 in
/opt/anaconda3/lib/python3.8/site-packages (from scikit-learn>=0.21.2->plot-
metric) (0.17.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/opt/anaconda3/lib/python3.8/site-packages (from scikit-learn>=0.21.2->plot-
metric) (2.1.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in
/opt/anaconda3/lib/python3.8/site-packages (from matplotlib>=3.0.2->plot-metric)
(2.4.7)
Requirement already satisfied: pillow>=6.2.0 in
/opt/anaconda3/lib/python3.8/site-packages (from matplotlib>=3.0.2->plot-metric)
(8.0.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/anaconda3/lib/python3.8/site-packages (from matplotlib>=3.0.2->plot-metric)
(1.3.0)
Requirement already satisfied: cycler>=0.10 in
/opt/anaconda3/lib/python3.8/site-packages (from matplotlib>=3.0.2->plot-metric)
(0.10.0)
Requirement already satisfied: certifi>=2020.06.20 in
/opt/anaconda3/lib/python3.8/site-packages (from matplotlib>=3.0.2->plot-metric)
(2020.6.20)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3.8/site-
packages (from python-dateutil>=2.7.3->pandas>=0.23.4->plot-metric) (1.15.0)
Installing collected packages: colorlover, plot-metric
Successfully installed colorlover-0.3.0 plot-metric-0.0.6
Note: you may need to restart the kernel to use updated packages.
```

[56]:

```python
from plot_metric.functions import BinaryClassification
# Visualisation with plot_metric
bc = BinaryClassification(y_pred.round(), y_val, labels=["Class 1", "Class 2"])

# Figures
plt.figure(figsize=(5,5))
bc.plot_roc_curve()
plt.show()
```



Deliverables

Please submit the following:

- Your full notebook used for training including the ROC Curves, model weights and loss and accuracy plots wrt number of epochs.

References

Baldi, P., Sadowski, P. and Whiteson, D. "Searching for Exotic Particles in High-energy Physics with Deep Learning." Nature Communications 5 (July 2, 2014).

Contributors

Ali Hariri *(American University of Beirut)*

Sergei V. Gleyzer *(University of Alabama)*

SGD Classifier

February 25, 2021

0.1 Training SGDClassifier

[ ]:
```
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {
    'loss':['hinge', 'log', 'modified_huber'], 'penalty':['l2', 'l1',␣
↪'elasticnet'],
    'alpha':[0.01, 0.1, 1.0, 10.0, 100.0], 'max_iter':[10000], 'n_jobs':[-1],
    'early_stopping':[True], 'warm_start':[True]
}

sgd_clf = SGDClassifier()

grid_search = GridSearchCV(sgd_clf, param_grid, cv=4, verbose=3)

grid_search.fit(X_train, y_train)
```

[16]:
```
grid_search.best_params_
```

[16]:
```
{'alpha': 0.01,
 'early_stopping': True,
 'loss': 'modified_huber',
 'max_iter': 10000,
 'n_jobs': -1,
 'penalty': 'l2',
 'warm_start': True}
```

[37]:
```
#sgd_best_estimator = grid_search.best_estimator_
sgd_best_estimator = SGDClassifier(alpha=0.01, early_stopping=True,␣
↪loss='modified_huber',
              max_iter=10000, n_jobs=-1, warm_start=True)
sgd_best_estimator.fit(X_train, y_train)
```

[37]:
```
SGDClassifier(alpha=0.01, early_stopping=True, loss='modified_huber',
              max_iter=10000, n_jobs=-1, warm_start=True)
```

[38]:
```
y_predict_sgd = sgd_best_estimator.predict(X_test)
```

[39]:
```
confusion_matrix(y_predict_sgd, y_test)
```

[39]:
```
array([[433451, 248325],
       [395019, 683205]], dtype=int64)
```

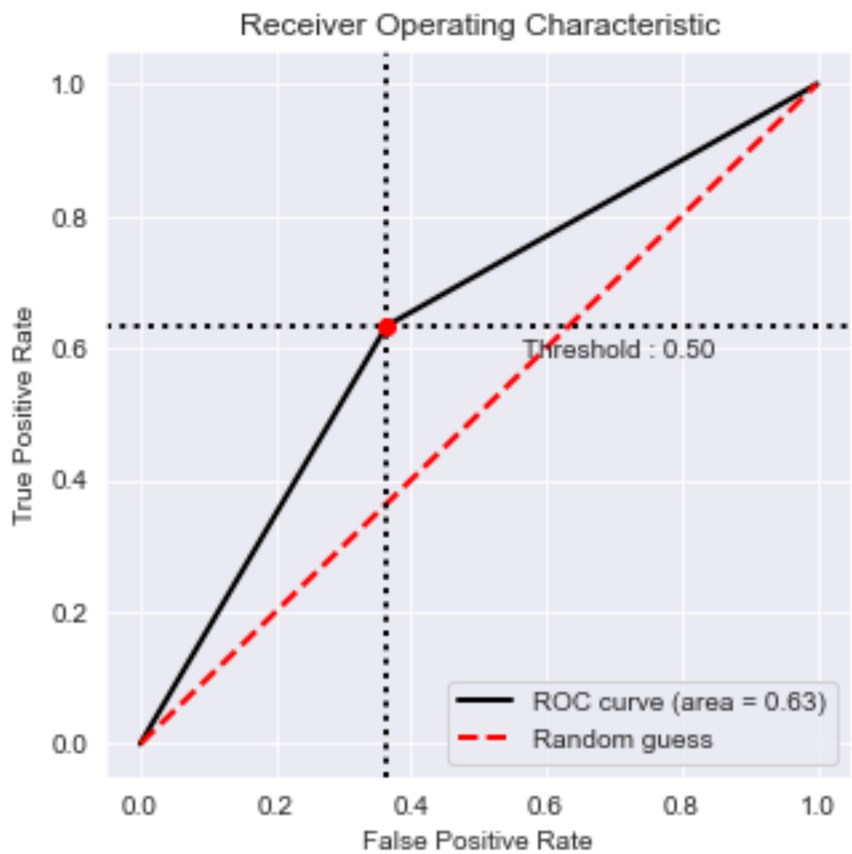
[46]:
```
accuracy_scor = accuracy_score(y_predict_sgd, y_test)
error = 1 - accuracy_scor
precision_scor = precision_score(y_predict_sgd, y_test)
recall_scor = recall_score(y_predict_sgd, y_test)
f1_scor = f1_score(y_predict_sgd, y_test)
auc = roc_auc_score(y_predict_sgd, y_test)

print("Accuracy score:", accuracy_scor)
print("Error:", error)
print("Precison score: ", precision_scor)
print("Recall score: ", recall_scor)
print("F1-Score:", f1_scor)
print("ROC Score:", auc)
```

```
Accuracy score: 0.6344636363636363
Error: 0.3655363636363636
Precison score:  0.7334224340600947
Recall score:  0.6336392066954547
F1-Score: 0.6798891804668631
ROC Score: 0.6347033364213468
```

[47]:
```
from plot_metric.functions import BinaryClassification
# Visualisation with plot_metric
bc = BinaryClassification(y_predict_sgd.round(), y_test, labels=["Class1",␣
 ↪"Class 2"])

# Figures
plt.figure(figsize=(5,5))
bc.plot_roc_curve()
plt.show()
```



1 DecisionTreeClassifier

[9]:
```
list(np.arange(2,19,3))
```

[9]: [2, 5, 8, 11, 14, 17]

[ ]:
```
decision_clf = DecisionTreeClassifier()

param_grid = [
    {"max_depth":list(np.arange(2,20,3)), "min_samples_split":list(np.
 ↪arange(10,40,5)), "min_samples_leaf":list(np.arange(5,20,5))}
        ]

grid_search_knn = GridSearchCV(decision_clf, param_grid, cv=4, verbose=3)

grid_search_knn.fit(X_train, y_train)
```

[14]:
```
decision_clf_best = grid_search_knn.best_estimator_
decision_clf_best
```

[14]: DecisionTreeClassifier(max_depth=14, min_samples_leaf=15, min_samples_split=35)

[16]:
```
y_pred_decision= decision_clf_best.predict(X_test)
confusion_matrix(y_pred_decision, y_test)
```

[16]:
```
array([[575984, 246065],
       [252486, 685465]], dtype=int64)
```

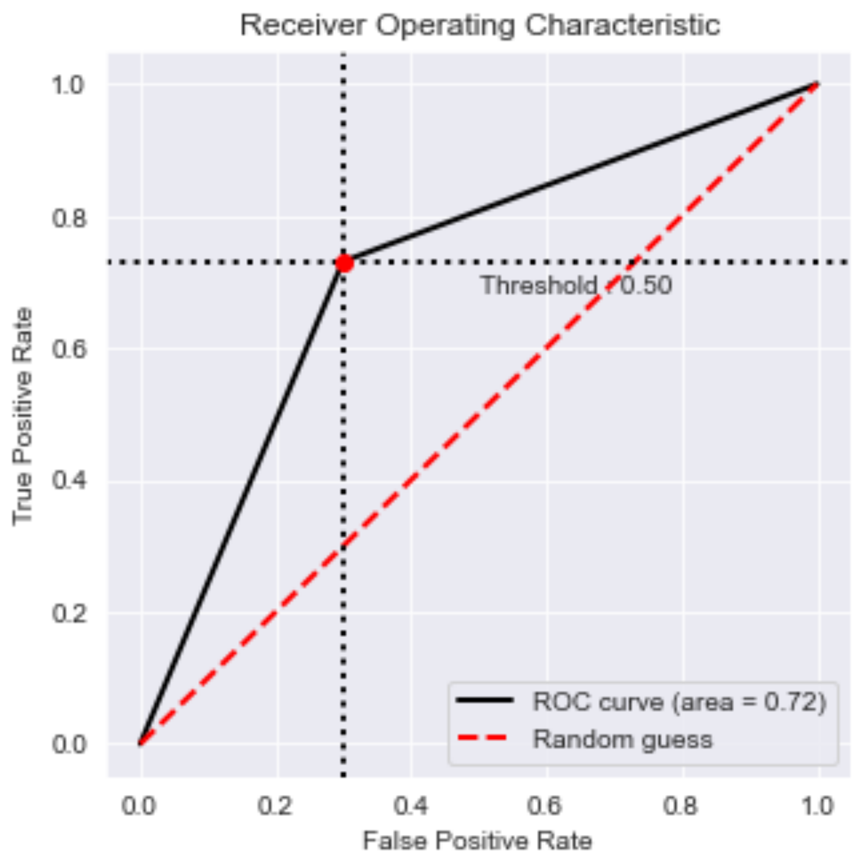
[45]:
```
accuracy_scor_decision = accuracy_score(y_pred_decision, y_test)
decision_error = 1 - accuracy_scor_decision
precision_scor_decision = precision_score(y_pred_decision, y_test)
recall_scor_decision = recall_score(y_pred_decision, y_test)
f1_scor_decision = f1_score(y_pred_decision, y_test)
decision_auc = roc_auc_score(y_pred_decision, y_test)

print("Accuracy Score:", accuracy_scor_decision)
print("Error:", decision_error)
print("Precison score: ", precision_scor_decision)
print("Recall score: ", recall_scor_decision)
print("F1_score:", f1_scor_decision)
print("ROC Score:", decision_auc)
```

```
Accuracy Score: 0.7167323863636363
Error: 0.28326761363636366
Precison score:  0.7358485502345603
Recall score:  0.7308110978078812
F1_score: 0.7333211730956345
ROC Score: 0.715739896369846
```

[43]:
```
from plot_metric.functions import BinaryClassification
# Visualisation with plot_metric
bc = BinaryClassification(y_pred_decision.round(), y_test, labels=["Class1",␣
↪"Class 2"])

# Figures
plt.figure(figsize=(5,5))
bc.plot_roc_curve()
plt.show()
```


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
Threshold : 0.50
ROC curve (area = 0.72)
Random guess
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Classifiers

February 25, 2021

1 Logistic Regression Classifier:

[ ]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

grid_lr = {"C":np.logspace(-3,3,7), "penalty":["l1","l2"]}

logreg = LogisticRegression(solver="liblinear")

logreg_cv = GridSearchCV(logreg,grid_lr,cv=4,verbose=3)

logreg_cv.fit(X_train,y_train)
```

1.1 Best Parameters:

[89]:
```python
print("Best parameters of Logistic Regression from grid search:")
print(logreg_cv.best_params_)
```

```
Best parameters of Logistic Regression from grid search:
{'C': 1.0, 'penalty': 'l2'}
```

[90]:
```python
logreg_best_est = logreg_cv.best_estimator_
logreg_best_est
```

[90]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

[91]:
```python
y_predict_logreg = logreg_best_est.predict(X_test)
```

1.2 Confusion Matrix:

[92]:
```
from sklearn.metrics import confusion_matrix
print("Confusion Matrix for Logistic Regression Classifier:")
confusion_matrix(y_test,y_predict_logreg)
```

```
Confusion Matrix for Logistic Regression Classifier:
```

[92]:
```
array([[3978, 3601],
       [2165, 6256]])
```

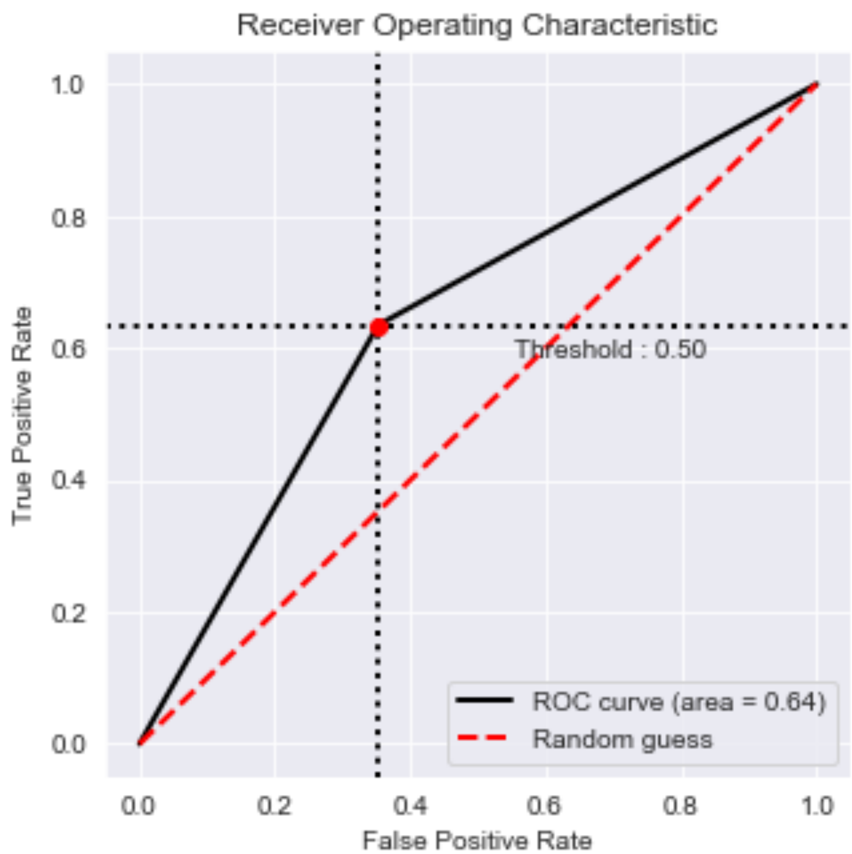
1.3 Calculating ROC and AUC:

[93]:
```
from plot_metric.functions import BinaryClassification

# Visualisation with plot_metric
bc_logreg = BinaryClassification(y_predict_logreg.round(), y_test,
                                 labels=["Class 1", "Class 2"])

# Figures
plt.figure(figsize=(5,5))
bc_logreg.plot_roc_curve()
plt.show()

print("Accuracy of logistic regression \
classifier on test set: {:.3f}".format(logreg_best_est.score
                                       (X_test, y_test)))
```




```
Accuracy of logistic regression classifier on test set: 0.640
```

[95]:

```
roc_auc_score(y_test, logreg_cv.predict_proba(X_test)[:,1])

z = confusion_matrix(y_test,y_predict_logreg)

err_lr = (z[1][0]+z[0][1])/(z[0][0]+z[0][1]+z[1][0]+z[1][1])
acc_lr = 1-err_lr
print("Error of logistic regression classifier on test set:",err_lr)
print("Accuracy of logistic regression classifier on test set:",acc_lr)
```

```
Error of logistic regression classifier on test set: 0.360375
Accuracy of logistic regression classifier on test set: 0.639625
```

2 Random Forest Classifier

[8]:
```
from sklearn.ensemble import RandomForestClassifier as rfc
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
```

[84]:
```
estimator = RandomForestClassifier()


estimators = [10, 50, 100, 200, 500]
max_depths = [3, 6, 10, 15, 20]
grid_values = {'n_estimators': estimators, 'max_depth':max_depths}
clf = GridSearchCV(RandomForestClassifier(),
                   grid_values, scoring='roc_auc', n_jobs=-1, cv=3)
clf.fit(X_train, y_train)
best_n_estimators_value = clf.best_params_['n_estimators']
best_max_depth_value = clf.best_params_['max_depth']
best_score = clf.best_score_
```

2.1 Best Parameters:

[85]:
```
print("Best parameters of RF from grid search:")
print(clf.best_params_)

clf_best_est = clf.best_estimator_
print("Best estimator:",clf_best_est)




print("Precison score:", precision_scor)
print("Recall score:", recall_scor)
```

```
Best parameters of RF from grid search:
{'max_depth': 20, 'n_estimators': 500}
Best estimator: RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',
                       max_depth=20, max_features='auto', 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=500,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

2.2 Confusion Matrix:

[86]:
```
y_predict_rf = clf_best_est.predict(X_test)
print("Confusion Matrix:")
confusion_matrix(y_test,y_predict_rf)
```

```
Confusion Matrix:
```

[86]:
```
array([[5182, 2397],
       [2142, 6279]])
```

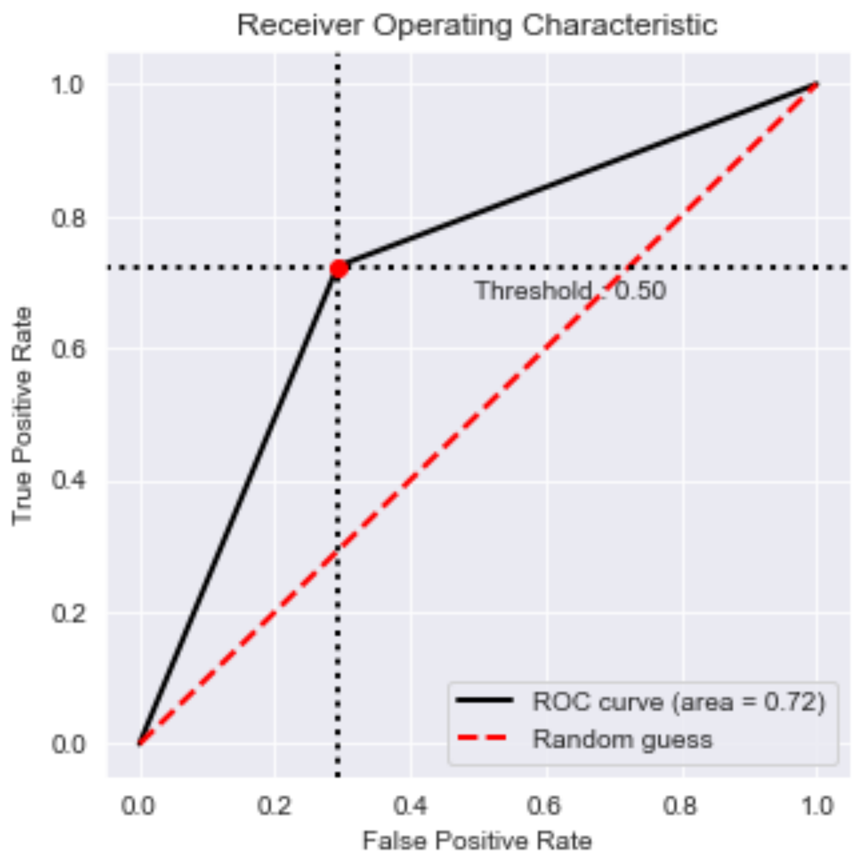
2.3 Calculating ROC and AUC:

[87]:
```
from plot_metric.functions import BinaryClassification

# Visualisation with plot_metric
bc_rf = BinaryClassification(y_predict_rf.round(), y_test,
                             labels=["Class 1", "Class 2"])

# Figures
plt.figure(figsize=(5,5))
bc_rf.plot_roc_curve()
plt.show()

print("Accuracy of random forest \
classifier on test set: {:.3f}".format(clf_best_est.score
                                       (X_test, y_test)))
```




```
Accuracy of random forest classifier on test set: 0.716
```