Parameter Tuning - XGBoost

July 9, 2018

In [2]:
```
# Import Libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xgboost as xgb
from xgboost import XGBClassifier

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.grid_search import GridSearchCV
```

In [3]:
```
# Import data with header
data = pd.read_csv('breastcancer.csv')
```

In [4]:
```
# Print shape of the dataframe and its first few rows
print ("Dataframe Shape: " + str(data.shape) + "\n")
data.head()
```

```
Dataframe Shape: (286, 10)
```

Out[4]:
```
                  class    age menopause tumor-size inv-nodes node-caps  \
0     recurrence-events  40-49   premeno      15-19       0-2       yes
1  no-recurrence-events  50-59      ge40      15-19       0-2        no
2     recurrence-events  50-59      ge40      35-39       0-2        no
3  no-recurrence-events  40-49   premeno      35-39       0-2       yes
4     recurrence-events  40-49   premeno      30-34       3-5       yes

   deg-malig breast breast-quad irradiat
0          3  right     left_up       no
1          1  right     central       no
2          2   left    left_low       no
3          3  right    left_low      yes
4          2   left    right_up       no
```

```
In [5]: # Extract Target feature into a variable.
        Y = data["class"]

        # Remove Target feature to form predictors dataset
        X = data.drop(["class"], axis=1)
```

```
In [6]: # Print unique values of each category variables
        for colname in data.columns:
            print (colname + ": " + str(data[colname].unique()))
```

```
class: ['recurrence-events' 'no-recurrence-events']
age: ['40-49' '50-59' '60-69' '30-39' '70-79' '20-29']
menopause: ['premeno' 'ge40' 'lt40']
tumor-size: ['15-19' '35-39' '30-34' '25-29' '40-44' '10-14' '0-4' '20-24' '45-49'
 '50-54' '5-9']
inv-nodes: ['0-2' '3-5' '15-17' '6-8' '9-11' '24-26' '12-14']
node-caps: ['yes' 'no' nan]
deg-malig: [3 1 2]
breast: ['right' 'left']
breast-quad: ['left_up' 'central' 'left_low' 'right_up' 'right_low' nan]
irradiat: ['no' 'yes']
```

```
In [7]: # Perform one-hot encoding.
        X = pd.get_dummies(X, drop_first=False) # Remove first coulmn to avoid collineartiy

        # Print first few features
        X.iloc[0:5, 0:6]
```

Out[7]:

	deg-malig	age_20-29	age_30-39	age_40-49	age_50-59	age_60-69
0	3	0	0	1	0	0
1	1	0	0	0	1	0
2	2	0	0	0	1	0
3	3	0	0	1	0	0
4	2	0	0	1	0	0

```
In [8]: # Encode Target feature [class] as Integer
        label_encoder = LabelEncoder()
        label_encoded_y = label_encoder.fit_transform(Y)
        print(label_encoded_y[0:20], "\n \n", label_encoded_y.shape)
```

```
[1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]

 (286,)
```

0.0.1 Let us consider XGBoost algorithm to train the model and tune its parameters

```
In [9]: # Split data into Train and Test sets
        X_train, X_test, y_train, y_test = train_test_split(X,
```

```
                                                              label_encoded_y,
                                                              test_size = 0.3,
                                                              random_state = 2)

        # Generate Model
        model = XGBClassifier(learning_rate =0.01,
                              subsample=0.75,
                              colsample_bytree=0.72,
                              min_child_weight=8,
                              max_depth=5)
        model.fit(X_train, y_train)
        print(model)
```

```
XGBClassifier(base_score=0.5, colsample_bylevel=1, colsample_bytree=0.72,
       gamma=0, learning_rate=0.01, max_delta_step=0, max_depth=5,
       min_child_weight=8, missing=None, n_estimators=100, nthread=-1,
       objective='binary:logistic', reg_alpha=0, reg_lambda=1,
       scale_pos_weight=1, seed=0, silent=True, subsample=0.75)
```

```
In [10]: # Make predictions for test data
         y_pred = model.predict(X_test)  # Array into list

         print(y_pred[0:25])
```

```
[0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
```

```
In [11]: # Evaluate predictions
         accuracy = accuracy_score(y_test, y_pred)
         print("XGBoost model accuracy: %.2f%% " % (100 * accuracy))
```

```
XGBoost model accuracy: 73.26%
```

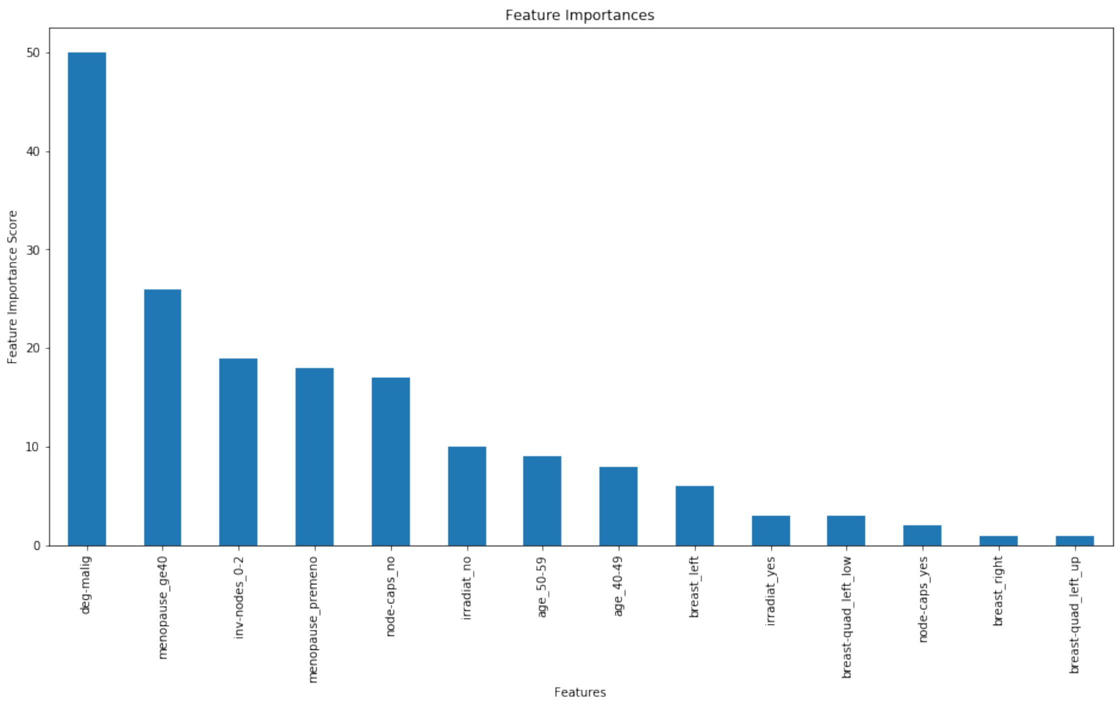
```
In [12]: pd.crosstab(y_test, y_pred, margins=True)
```

Out[12]:

col_0	0	1	All
row_0			
0	56	4	60
1	19	7	26
All	75	11	86

```
In [13]: plt.figure(figsize = (16, 8))

         feat_imp = pd.Series(model.booster().get_fscore()).sort_values(ascending=False)
         feat_imp.plot(kind='bar', title='Feature Importances')
         plt.ylabel('Feature Importance Score')
         plt.xlabel('Features')
         plt.show()
```



##
Parameter Tuning

0.0.2 Tune max_depth and min_child_weight

```
In [14]: # Set range of parameters for max_depth and min_child_weight
         param_test1 = {
             'max_depth':list(range(1, 10, 1)),
             'min_child_weight':list(range(1, 10, 1))
         }

         # Build the XGBoost model for the range of max_depth and min_child_weight values
         gridSearch = GridSearchCV(XGBClassifier(learning_rate =0.01,
                                                  n_estimators=140,
                                                 # max_depth=5,
                                                 # min_child_weight=2,
                                                  gamma=0,
                                                  subsample=0.75,
                                                  colsample_bytree=0.72,
                                                  silent=False),
                      param_grid = param_test1,
                      scoring = 'roc_auc',
                      n_jobs = 4,
                      iid = False,
                      cv = 5)
```

```python
# Fit the train dataset
gridSearch.fit(X_train, y_train)

# Print scores for each parameters.
# REMEMBER the scores are based on train dataset only and NOT on test dataset
gridSearch.grid_scores_, gridSearch.best_params_, gridSearch.best_score_
```

Out[14]:

```
([mean: 0.70170, std: 0.12188, params: {'max_depth': 1, 'min_child_weight': 1},
  mean: 0.69642, std: 0.12545, params: {'max_depth': 1, 'min_child_weight': 2},
  mean: 0.69940, std: 0.12522, params: {'max_depth': 1, 'min_child_weight': 3},
  mean: 0.68040, std: 0.12847, params: {'max_depth': 1, 'min_child_weight': 4},
  mean: 0.67985, std: 0.12972, params: {'max_depth': 1, 'min_child_weight': 5},
  mean: 0.68559, std: 0.12289, params: {'max_depth': 1, 'min_child_weight': 6},
  mean: 0.66835, std: 0.11714, params: {'max_depth': 1, 'min_child_weight': 7},
  mean: 0.65530, std: 0.09798, params: {'max_depth': 1, 'min_child_weight': 8},
  mean: 0.61794, std: 0.13305, params: {'max_depth': 1, 'min_child_weight': 9},
  mean: 0.69890, std: 0.12565, params: {'max_depth': 2, 'min_child_weight': 1},
  mean: 0.70742, std: 0.12419, params: {'max_depth': 2, 'min_child_weight': 2},
  mean: 0.69198, std: 0.12130, params: {'max_depth': 2, 'min_child_weight': 3},
  mean: 0.68084, std: 0.11920, params: {'max_depth': 2, 'min_child_weight': 4},
  mean: 0.67385, std: 0.11921, params: {'max_depth': 2, 'min_child_weight': 5},
  mean: 0.65933, std: 0.11111, params: {'max_depth': 2, 'min_child_weight': 6},
  mean: 0.65165, std: 0.09795, params: {'max_depth': 2, 'min_child_weight': 7},
  mean: 0.64255, std: 0.08300, params: {'max_depth': 2, 'min_child_weight': 8},
  mean: 0.61611, std: 0.13243, params: {'max_depth': 2, 'min_child_weight': 9},
  mean: 0.69702, std: 0.12259, params: {'max_depth': 3, 'min_child_weight': 1},
  mean: 0.70107, std: 0.11851, params: {'max_depth': 3, 'min_child_weight': 2},
  mean: 0.67673, std: 0.10202, params: {'max_depth': 3, 'min_child_weight': 3},
  mean: 0.68959, std: 0.11438, params: {'max_depth': 3, 'min_child_weight': 4},
  mean: 0.67329, std: 0.11948, params: {'max_depth': 3, 'min_child_weight': 5},
  mean: 0.66171, std: 0.10984, params: {'max_depth': 3, 'min_child_weight': 6},
  mean: 0.65165, std: 0.09795, params: {'max_depth': 3, 'min_child_weight': 7},
  mean: 0.64255, std: 0.08300, params: {'max_depth': 3, 'min_child_weight': 8},
  mean: 0.61611, std: 0.13243, params: {'max_depth': 3, 'min_child_weight': 9},
  mean: 0.69235, std: 0.10994, params: {'max_depth': 4, 'min_child_weight': 1},
  mean: 0.69782, std: 0.10929, params: {'max_depth': 4, 'min_child_weight': 2},
  mean: 0.67869, std: 0.10276, params: {'max_depth': 4, 'min_child_weight': 3},
  mean: 0.68843, std: 0.11048, params: {'max_depth': 4, 'min_child_weight': 4},
  mean: 0.67329, std: 0.11948, params: {'max_depth': 4, 'min_child_weight': 5},
  mean: 0.66171, std: 0.10984, params: {'max_depth': 4, 'min_child_weight': 6},
  mean: 0.65165, std: 0.09795, params: {'max_depth': 4, 'min_child_weight': 7},
  mean: 0.64255, std: 0.08300, params: {'max_depth': 4, 'min_child_weight': 8},
  mean: 0.61611, std: 0.13243, params: {'max_depth': 4, 'min_child_weight': 9},
  mean: 0.69055, std: 0.11056, params: {'max_depth': 5, 'min_child_weight': 1},
  mean: 0.69741, std: 0.11096, params: {'max_depth': 5, 'min_child_weight': 2},
  mean: 0.67663, std: 0.10314, params: {'max_depth': 5, 'min_child_weight': 3},
  mean: 0.68843, std: 0.11048, params: {'max_depth': 5, 'min_child_weight': 4},
```

```
 mean: 0.67329, std: 0.11948, params: {'max_depth': 5, 'min_child_weight': 5},
 mean: 0.66171, std: 0.10984, params: {'max_depth': 5, 'min_child_weight': 6},
 mean: 0.65165, std: 0.09795, params: {'max_depth': 5, 'min_child_weight': 7},
 mean: 0.64255, std: 0.08300, params: {'max_depth': 5, 'min_child_weight': 8},
 mean: 0.61611, std: 0.13243, params: {'max_depth': 5, 'min_child_weight': 9},
 mean: 0.68931, std: 0.10918, params: {'max_depth': 6, 'min_child_weight': 1},
 mean: 0.69676, std: 0.10992, params: {'max_depth': 6, 'min_child_weight': 2},
 mean: 0.67669, std: 0.10225, params: {'max_depth': 6, 'min_child_weight': 3},
 mean: 0.68843, std: 0.11048, params: {'max_depth': 6, 'min_child_weight': 4},
 mean: 0.67329, std: 0.11948, params: {'max_depth': 6, 'min_child_weight': 5},
 mean: 0.66171, std: 0.10984, params: {'max_depth': 6, 'min_child_weight': 6},
 mean: 0.65165, std: 0.09795, params: {'max_depth': 6, 'min_child_weight': 7},
 mean: 0.64255, std: 0.08300, params: {'max_depth': 6, 'min_child_weight': 8},
 mean: 0.61611, std: 0.13243, params: {'max_depth': 6, 'min_child_weight': 9},
 mean: 0.68640, std: 0.10783, params: {'max_depth': 7, 'min_child_weight': 1},
 mean: 0.69741, std: 0.11074, params: {'max_depth': 7, 'min_child_weight': 2},
 mean: 0.67669, std: 0.10225, params: {'max_depth': 7, 'min_child_weight': 3},
 mean: 0.68843, std: 0.11048, params: {'max_depth': 7, 'min_child_weight': 4},
 mean: 0.67329, std: 0.11948, params: {'max_depth': 7, 'min_child_weight': 5},
 mean: 0.66171, std: 0.10984, params: {'max_depth': 7, 'min_child_weight': 6},
 mean: 0.65165, std: 0.09795, params: {'max_depth': 7, 'min_child_weight': 7},
 mean: 0.64255, std: 0.08300, params: {'max_depth': 7, 'min_child_weight': 8},
 mean: 0.61611, std: 0.13243, params: {'max_depth': 7, 'min_child_weight': 9},
 mean: 0.68575, std: 0.10854, params: {'max_depth': 8, 'min_child_weight': 1},
 mean: 0.69681, std: 0.11001, params: {'max_depth': 8, 'min_child_weight': 2},
 mean: 0.67669, std: 0.10225, params: {'max_depth': 8, 'min_child_weight': 3},
 mean: 0.68843, std: 0.11048, params: {'max_depth': 8, 'min_child_weight': 4},
 mean: 0.67329, std: 0.11948, params: {'max_depth': 8, 'min_child_weight': 5},
 mean: 0.66171, std: 0.10984, params: {'max_depth': 8, 'min_child_weight': 6},
 mean: 0.65165, std: 0.09795, params: {'max_depth': 8, 'min_child_weight': 7},
 mean: 0.64255, std: 0.08300, params: {'max_depth': 8, 'min_child_weight': 8},
 mean: 0.61611, std: 0.13243, params: {'max_depth': 8, 'min_child_weight': 9},
 mean: 0.68754, std: 0.10775, params: {'max_depth': 9, 'min_child_weight': 1},
 mean: 0.69681, std: 0.11001, params: {'max_depth': 9, 'min_child_weight': 2},
 mean: 0.67669, std: 0.10225, params: {'max_depth': 9, 'min_child_weight': 3},
 mean: 0.68843, std: 0.11048, params: {'max_depth': 9, 'min_child_weight': 4},
 mean: 0.67329, std: 0.11948, params: {'max_depth': 9, 'min_child_weight': 5},
 mean: 0.66171, std: 0.10984, params: {'max_depth': 9, 'min_child_weight': 6},
 mean: 0.65165, std: 0.09795, params: {'max_depth': 9, 'min_child_weight': 7},
 mean: 0.64255, std: 0.08300, params: {'max_depth': 9, 'min_child_weight': 8},
 mean: 0.61611, std: 0.13243, params: {'max_depth': 9, 'min_child_weight': 9}],
{'max_depth': 2, 'min_child_weight': 2},
0.7074171518137036)
```

In [15]:
```
# Extract best scores from Grid search
maxdepthvalue = gridSearch.best_params_['max_depth']
minchildvalue = gridSearch.best_params_['min_child_weight']
```

0.0.3 Take the values of max_depth and min_child_weight from previous step and tune subsample and colsample_bytree

In [16]:
```
# Set range of parameters for subsample and colsample_bytree
param_test2 = {
 'subsample':[0.6, 0.65, 0.7, 0.75, 0.8],
 'colsample_bytree':[0.6, 0.65, 0.7, 0.75, 0.8]
}

# Build the XGBoost model for the range of subsample and colsample_bytree values
gridSearch = GridSearchCV(XGBClassifier(learning_rate = 0.01,
                                        n_estimators = 140,
                                        max_depth = maxdepthvalue,
                                        min_child_weight = minchildvalue,
                                        gamma = 0
                                        #subsample=0.75,
                                        #colsample_bytree=0.72
                                            ),
             param_grid = param_test2,
             scoring = 'roc_auc',
             n_jobs = 4,
             iid = False,
             cv = 5)

# Fit the train dataset
gridSearch.fit(X_train, y_train)

# Print scores for each parameters.
# REMEMBER the scores are based on train dataset only and NOT on test dataset
gridSearch.grid_scores_, gridSearch.best_params_, gridSearch.best_score_
```

Out[16]:
```
([mean: 0.70321, std: 0.13466, params: {'colsample_bytree': 0.6, 'subsample': 0.6},
  mean: 0.70981, std: 0.12428, params: {'colsample_bytree': 0.6, 'subsample': 0.65},
  mean: 0.70587, std: 0.13063, params: {'colsample_bytree': 0.6, 'subsample': 0.7},
  mean: 0.70532, std: 0.11993, params: {'colsample_bytree': 0.6, 'subsample': 0.75},
  mean: 0.70086, std: 0.13242, params: {'colsample_bytree': 0.6, 'subsample': 0.8},
  mean: 0.70269, std: 0.12932, params: {'colsample_bytree': 0.65, 'subsample': 0.6},
  mean: 0.70448, std: 0.12695, params: {'colsample_bytree': 0.65, 'subsample': 0.65},
  mean: 0.70154, std: 0.13127, params: {'colsample_bytree': 0.65, 'subsample': 0.7},
  mean: 0.70675, std: 0.12099, params: {'colsample_bytree': 0.65, 'subsample': 0.75},
  mean: 0.70372, std: 0.12895, params: {'colsample_bytree': 0.65, 'subsample': 0.8},
  mean: 0.69849, std: 0.12667, params: {'colsample_bytree': 0.7, 'subsample': 0.6},
  mean: 0.70207, std: 0.12744, params: {'colsample_bytree': 0.7, 'subsample': 0.65},
  mean: 0.70030, std: 0.13200, params: {'colsample_bytree': 0.7, 'subsample': 0.7},
  mean: 0.70798, std: 0.12614, params: {'colsample_bytree': 0.7, 'subsample': 0.75},
  mean: 0.70494, std: 0.12342, params: {'colsample_bytree': 0.7, 'subsample': 0.8},
  mean: 0.70209, std: 0.12629, params: {'colsample_bytree': 0.75, 'subsample': 0.6},
  mean: 0.70158, std: 0.12611, params: {'colsample_bytree': 0.75, 'subsample': 0.65},
  mean: 0.70142, std: 0.13182, params: {'colsample_bytree': 0.75, 'subsample': 0.7},
```

```
 mean: 0.70909, std: 0.12633, params: {'colsample_bytree': 0.75, 'subsample': 0.75},
 mean: 0.70356, std: 0.12658, params: {'colsample_bytree': 0.75, 'subsample': 0.8},
 mean: 0.70440, std: 0.12373, params: {'colsample_bytree': 0.8, 'subsample': 0.6},
 mean: 0.69832, std: 0.12957, params: {'colsample_bytree': 0.8, 'subsample': 0.65},
 mean: 0.70552, std: 0.12882, params: {'colsample_bytree': 0.8, 'subsample': 0.7},
 mean: 0.70656, std: 0.12491, params: {'colsample_bytree': 0.8, 'subsample': 0.75},
 mean: 0.70202, std: 0.12343, params: {'colsample_bytree': 0.8, 'subsample': 0.8}],
{'colsample_bytree': 0.6, 'subsample': 0.65},
0.7098139647708612)
```

In [17]:
```
# Extract best scores from Grid search
colsamplevalue = gridSearch.best_params_['colsample_bytree']
subsamplevalue = gridSearch.best_params_['subsample']
```

0.0.4 Take the values of max_depth, min_child_weight, subsample, and colsample_bytree from previous steps and tune gamma

In [18]:
```
# Set range of parameters for gamma
param_test3 = {
 'gamma':[0.0, 0.01, 0.001, 0.2, 0.002]
}

# Build the XGBoost model for the range of gamma values
gridSearch = GridSearchCV(XGBClassifier(learning_rate = 0.01,
                                         n_estimators = 150,
                                         max_depth = maxdepthvalue,
                                         min_child_weight = minchildvalue,
                                         #gamma=0
                                         subsample = subsamplevalue,
                                         colsample_bytree = colsamplevalue),
             param_grid = param_test3,
             scoring = 'roc_auc',
             n_jobs = 4,
             iid = False,
             cv = 5)

# Fit the train dataset
gridSearch.fit(X_train, y_train)

# Print scores for each parameters.
# REMEMBER the scores are based on train dataset only and NOT on test dataset
gridSearch.grid_scores_, gridSearch.best_params_, gridSearch.best_score_
```

Out[18]:
```
([mean: 0.71048, std: 0.12635, params: {'gamma': 0.0},
  mean: 0.71048, std: 0.12635, params: {'gamma': 0.01},
  mean: 0.71048, std: 0.12635, params: {'gamma': 0.001},
  mean: 0.70991, std: 0.12710, params: {'gamma': 0.2},
  mean: 0.71048, std: 0.12635, params: {'gamma': 0.002}],
```

```
 {'gamma': 0.0},
 0.710483840871772)
```

```
In [19]: gammavalue = gridSearch.best_params_['gamma']
```

0.0.5 Take the values of max_depth, min_child_weight, subsample, colsample_bytree, and gamma from previous steps and tune reg_alpha

```
In [20]: # Set range of parameters for reg_alpha
         param_test4 = {
          'reg_alpha':[0.9, 0.95, 1, 1.05]
         }

         # Build the XGBoost model for the range of reg_alpha values
         gridSearch = GridSearchCV(XGBClassifier(learning_rate = 0.01,
                                                 n_estimators = 150,
                                                 max_depth = maxdepthvalue,
                                                 min_child_weight = minchildvalue,
                                                 gamma = gammavalue,
                                                 subsample = subsamplevalue,
                                                 colsample_bytree = colsamplevalue),
                      param_grid = param_test4,
                      scoring = 'roc_auc',
                      n_jobs = 4,
                      iid = False,
                      cv = 5)

         # Fit the train dataset
         gridSearch.fit(X_train, y_train)

         # Print scores for each parameters.
         # REMEMBER the scores are based on train dataset only and NOT on test dataset
         gridSearch.grid_scores_, gridSearch.best_params_, gridSearch.best_score_
```

```
Out[20]: ([mean: 0.71053, std: 0.13149, params: {'reg_alpha': 0.9},
           mean: 0.71110, std: 0.13070, params: {'reg_alpha': 0.95},
           mean: 0.70986, std: 0.12873, params: {'reg_alpha': 1},
           mean: 0.70993, std: 0.12945, params: {'reg_alpha': 1.05}],
          {'reg_alpha': 0.95},
          0.7111005373936409)
```

```
In [21]: regalphavalue = gridSearch.best_params_['reg_alpha']
```

0.0.6 Combine all the tuned parameters and train the model

```
In [22]: # Set final parameters
         finalParams = {
             "max_depth": maxdepthvalue,
             "min_child_weight": minchildvalue,
```

```
        "gamma": gammavalue,
        "subsample": subsamplevalue,
        "colsample_bytree": colsamplevalue,
        "reg_alpha": regalphavalue,
        "learning_rate": 0.01
    }

    # Create XGB matrix on train dataset
    xgbTrain = xgb.DMatrix(X_train, label=y_train)

    cvResult = xgb.cv(finalParams,
                      xgbTrain,
                      num_boost_round = 500,
                      nfold = 5,
                      metrics = 'auc',
                      early_stopping_rounds=25,
                      verbose_eval=False
                     )

    cvResult.shape[0]
```

```
Out[22]: 60
```

```
In [23]: estimatorvalue = cvResult.shape[0]
```

```
In [24]: finalModel = XGBClassifier(learning_rate = 0.01,
                                    n_estimators = estimatorvalue,
                                    max_depth = maxdepthvalue,
                                    min_child_weight= minchildvalue,
                                    gamma = gammavalue,
                                    subsample = subsamplevalue,
                                    colsample_bytree = colsamplevalue,
                                    reg_alpha = regalphavalue,
                                    silent=True
                                   )
```

```
In [25]: finalModel.fit(X_train, y_train)

         # Make predictions for test data
         y_pred = finalModel.predict(X_test)  # Array into list

         print(y_pred[0:25])
```

```
[0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

```
In [26]: # Evaluate predictions
         accuracy = accuracy_score(y_test, y_pred)
         print("XGBoost model accuracy after parameter tuning: %.2f%%" % (100 * accuracy))
```

```
XGBoost model accuracy after parameter tuning: 74.42%
```

```
In [27]: pd.crosstab(y_test, y_pred, margins=True)
```

```
Out[27]: col_0   0   1  All
         row_0
         0      57   3   60
         1      19   7   26
         All    76  10   86
```

Reference: https://jessesw.com/XG-Boost/