In [27]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
```

In [28]:
```
from sklearn.datasets import load_breast_cancer
dataset=load_breast_cancer()
X=dataset.data
y=dataset.target
```

In [29]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X,y,random_state=3)
```

In [30]:
```
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train,y_train)
knn.score(X_test,y_test)
```

Out[30]:
```
0.916083916083916
```

Bagging over KNN

In [31]:
```
bag_knn=BaggingClassifier(KNeighborsClassifier(n_neighbors=5),n_estimators=10,max_samples=0.5,bootstrap=
```

In [8]:
```
#Check out of bag score
bag_knn.fit(X_train, y_train)
bag_knn.oob_score_
```

Out[8]:
```
0.9295774647887324
```

In [9]:
```
bag_knn.score(X_test,y_test)
```

Out[9]:
```
0.9370629370629371
```

Pasting

In [10]:
```
pasting_knn=BaggingClassifier(KNeighborsClassifier(n_neighbors=5),n_estimators=10,max_samples=0.5,bootst
```

In [11]:
```
pasting_knn.fit(X_train,y_train)
pasting_knn.score(X_test,y_test)
```

Out[11]:
```
0.9300699300699301
```

Random Forests

In [18]:
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score
#from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
```

In [3]:
```
data = pd.read_csv("winequality_red.csv")
data
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

1599 rows × 12 columns

In [4]:
```
data.describe()
```

Out[4]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690

In [5]:
```
X = data.drop(columns = 'quality')
y = data['quality']
```

In [6]:
```
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size = 0.30, random_state= 355)
```

In [7]:
```
#let's first visualize the tree on the data without doing any pre processing
clf = DecisionTreeClassifier( min_samples_split= 2)
clf.fit(x_train,y_train)
```

Out[7]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [8]:
```
# accuracy of our classification tree
clf.score(x_test,y_test)
```

Out[8]:
```
0.6208333333333333
```

In [9]:
```
#let's first visualize the tree on the data without doing any pre processing
clf2 = DecisionTreeClassifier(criterion = 'entropy', max_depth =24, min_samples_leaf= 1)
clf2.fit(x_train,y_train)
```

Out[9]:
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=24)
```

In [10]:
```
clf2.score(x_test,y_test)
```

Out[10]: 0.64375

In [11]:
```
rand_clf = RandomForestClassifier(random_state=6)
```

In [12]:
```
rand_clf.fit(x_train,y_train)
```

Out[12]:
```
RandomForestClassifier
RandomForestClassifier(random_state=6)
```

In [13]:
```
rand_clf.score(x_test,y_test)
```

Out[13]: 0.6708333333333333

In [14]:
```
# we are tuning three hyperparameters right now, we are passing the different values for both parameters
grid_param = {
    "n_estimators" : [90,100,115,130],
    'criterion': ['gini', 'entropy'],
    'max_depth' : range(2,20,1),
    'min_samples_leaf' : range(1,10,1),
    'min_samples_split': range(2,10,1),
    'max_features' : ['auto','log2']
}
```

In [15]:
```
grid_search = GridSearchCV(estimator=rand_clf,param_grid=grid_param,cv=5,n_jobs =-1,verbose = 3)
```

In [ ]:
```
#grid_search.fit(x_train,y_train)
```

In [17]:
```
rand_clf = RandomForestClassifier(criterion= 'entropy',
 max_depth = 12,
 max_features = 'log2',
 min_samples_leaf = 1,
 min_samples_split= 5,
 n_estimators = 90,random_state=6)
```

In [18]:
```
rand_clf.fit(x_train,y_train)
```

Out[18]:
```
RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_depth=12, max_features='log2',
                       min_samples_split=5, n_estimators=90, random_state=6)
```

In [20]:
```
rand_clf.score(x_test,y_test)
```

Out[20]: 0.6604166666666667

In [21]:
```
# we are tuning three hyperparameters right now, we are passing the different values for both parameters
grid_param = {
    "n_estimators" : [90,100,115],
    'criterion': ['gini', 'entropy'],
    'min_samples_leaf' : [1,2,3,4,5],
    'min_samples_split': [4,5,6,7,8],
    'max_features' : ['auto','log2']
}
```

In [22]:
```
grid_search = GridSearchCV(estimator=rand_clf,param_grid=grid_param,cv=5,n_jobs =-1,verbose = 3)
```

In [23]:
```
grid_search.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 300 candidates, totalling 1500 fits
```

```
C:\Users\Personal\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter as it is also the default value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
```

Out[23]:

GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier

In [24]:
```
#Let's see the best parameters as per our grid search
grid_search.best_params_
```

Out[24]:
```
{'criterion': 'entropy',
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 7,
 'n_estimators': 90}
```

In [25]:
```
rand_clf = RandomForestClassifier(criterion= 'entropy',
 max_features = 'sqrt',
 min_samples_leaf = 1,
 min_samples_split= 4,
 n_estimators = 115,random_state=6)
```

In [26]:
```
rand_clf.fit(x_train,y_train)
```

Out[26]:

RandomForestClassifier

```
RandomForestClassifier(criterion='entropy', min_samples_split=4,
                       n_estimators=115, random_state=6)
```

In [28]:
```
rand_clf.score(x_test,y_test)
```

Out[28]:
```
0.6729166666666667
```

Stacking (Stacked Generalization)

In [1]:
```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from sklearn.model_selection import train_test_split
import numpy as np
```

In [2]:
```
data = pd.read_csv("diabetes.csv")
data
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

In [3]:
```
X = data.drop(columns = 'Outcome')
y = data['Outcome']
```

In [4]:
```
# let's divide our dataset into training set and hold out set by 50%
train,val_train,test,val_test = train_test_split(X,y,test_size=0.5, random_state= 355)
```

In [5]:
```
# let's split the training set again into training and test dataset
x_train,x_test,y_train,y_test =  train_test_split(train,test,test_size=0.2, random_state= 355)
```

In [6]:
```
knn = KNeighborsClassifier()

knn.fit(x_train,y_train)
```

Out[6]:
```
KNeighborsClassifier()
```

In [7]:
```
knn.score(x_test,y_test)
```

Out[7]:
```
0.7402597402597403
```

In [8]:
```
# rand_clf = RandomForestClassifier()

# rand_clf.fit(x_train,y_train)


svm = SVC()
svm.fit(x_train,y_train)
```

Out[8]:
```
SVC()
```

In [9]:
```
#rand_clf.score(x_test,y_test)

svm.score(x_test,y_test)
```

Out[9]:
```
0.7402597402597403
```

In [10]:
```
predict_val1 = knn.predict(val_train)
predict_val2 = svm.predict(val_train)
#predict_val2 = rand_clf.predict(val_train)
```

In [11]:
```
predict_val = np.column_stack((predict_val1,predict_val2))
predict_val
```

Out[11]:
```
array([[0, 0],
       [0, 0],
       [1, 1],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [1, 0],
       [1, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [1, 1],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [1, 1],
       [1, 0],
       [0, 0],
       [1, 1],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [1, 0],
       [0, 0],
       [1, 1],
       [1, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [1, 1],
       [1, 1],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [1, 1],
       [1, 0],
       [1, 1],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [1, 1],
       [1, 0],
       [0, 0],
       [0, 0],
       [1, 1],
       [1, 0],
       [1, 1],
       [1, 1],
       [1, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [1, 0],
       [1, 1],
       [0, 1],
```

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

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

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

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

```
       [0, 0],
       [0, 0],
       [0, 0],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 0],
       [0, 0],
       [1, 0]], dtype=int64)
```

In [12]:
```
predict_test1 = knn.predict(x_test)
predict_test2 = svm.predict(x_test)
#predict_test2 = rand_clf.predict(x_test)
```

In [13]:
```
predict_test = np.column_stack((predict_test1,predict_test2))
predict_test
```

Out[13]:
```
array([[1, 0],
       [0, 0],
       [1, 1],
       [1, 0],
       [0, 0],
       [1, 1],
       [1, 1],
       [1, 1],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [1, 0],
       [0, 0],
       [1, 1],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [1, 1],
       [1, 0],
       [0, 0],
       [0, 0],
       [1, 0],
       [0, 1],
       [1, 1],
       [1, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [1, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [1, 1],
       [1, 1],
       [0, 0],
       [1, 1],
       [1, 0],
       [1, 0],
       [0, 0],
       [0, 0],
       [1, 0],
       [0, 0],
       [0, 0],
       [1, 1],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [1, 0],
       [0, 0],
       [0, 0],
       [1, 0],
       [0, 0],
       [1, 0],
       [1, 0],
       [1, 1],
       [0, 0],
       [0, 0],
       [1, 0],
       [0, 0],
       [0, 0],
       [1, 1],
```

```
       [0, 0],
       [1, 1]], dtype=int64)
```

In [14]:
```
rand_clf = RandomForestClassifier()

rand_clf.fit(predict_val,val_test)
```

Out[14]:
```
RandomForestClassifier

RandomForestClassifier()
```

In [15]:
```
rand_clf.score(predict_test,y_test)
```

Out[15]:
```
0.7402597402597403
```

In [16]:
```
# we are tuning three hyperparameters right now, we are passing the different values for both parameters
grid_param = {
    "n_estimators" : [90,100,115],
    'criterion': ['gini', 'entropy'],
    'min_samples_leaf' : [1,2,3,4,5],
    'min_samples_split': [4,5,6,7,8],
    'max_features' : ['auto','log2']
}
```

In [19]:
```
grid_search = GridSearchCV(estimator=rand_clf,param_grid=grid_param,cv=5,n_jobs =-1,verbose = 3)
```

In [22]:
```
grid_search.fit(predict_val,val_test)
```

```
Fitting 5 folds for each of 300 candidates, totalling 1500 fits
C:\Users\Personal\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter as it is also the default value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
```

Out[22]:
```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

In [23]:
```
grid_search.best_params_
```

Out[23]:
```
{'criterion': 'gini',
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 4,
 'n_estimators': 90}
```

In [24]:
```
rand_clf = RandomForestClassifier( criterion='gini',max_features = 'auto',min_samples_leaf =1,min_sample
```

In [25]:
```
rand_clf.fit(predict_val,val_test)
```

```
C:\Users\Personal\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter as it is also the default value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
```

Out[25]:
```
RandomForestClassifier

RandomForestClassifier(max_features='auto', min_samples_split=4,
                       n_estimators=90)
```

In [26]:
```
rand_clf.score(predict_test,y_test)
```

Out[26]:
```
0.7402597402597403
```

Random Forest (Rice Classification)

In [1]:
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score
#from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
```

In [2]:
```
data = pd.read_csv("riceClassification.csv")
data
```

Out[2]:

	id	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	EquivDiameter	Extent	Perin
0	1	4537	92.229316	64.012769	0.719916	4677	76.004525	0.657536	27
1	2	2872	74.691881	51.400454	0.725553	3015	60.471018	0.713009	20
2	3	3048	76.293164	52.043491	0.731211	3132	62.296341	0.759153	21
3	4	3073	77.033628	51.928487	0.738639	3157	62.551300	0.783529	21
4	5	3693	85.124785	56.374021	0.749282	3802	68.571668	0.769375	23
...	...	...	...	...	...	...	...	...	
18180	18181	5853	148.624571	51.029281	0.939210	6008	86.326537	0.498594	33
18181	18182	7585	169.593996	58.141659	0.939398	7806	98.272692	0.647461	38
18182	18183	6365	154.777085	52.908085	0.939760	6531	90.023162	0.561287	34
18183	18184	5960	151.397924	51.474600	0.940427	6189	87.112041	0.492399	34
18184	18185	6134	153.081981	51.590606	0.941500	6283	88.374495	0.489975	33

18185 rows × 12 columns

In [3]:
```
data.describe()
```

Out[3]:

	id	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	EquivDiameter
count	18185.000000	18185.000000	18185.000000	18185.000000	18185.000000	18185.000000	18185.000000
mean	9093.000000	7036.492989	151.680754	59.807851	0.915406	7225.817872	94.132952
std	5249.701658	1467.197150	12.376402	10.061653	0.030575	1502.006571	9.906250
min	1.000000	2522.000000	74.133114	34.409894	0.676647	2579.000000	56.666658
25%	4547.000000	5962.000000	145.675910	51.393151	0.891617	6125.000000	87.126656
50%	9093.000000	6660.000000	153.883750	55.724288	0.923259	6843.000000	92.085696
75%	13639.000000	8423.000000	160.056214	70.156593	0.941372	8645.000000	103.559146
max	18185.000000	10210.000000	183.211434	82.550762	0.966774	11008.000000	114.016559

In [4]:
```
X = data.drop(columns = 'Class')
y = data['Class']
```

In [5]:
```
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size = 0.30, random_state= 355)
```

In [6]:
```
clf = DecisionTreeClassifier( min_samples_split= 2)
clf.fit(x_train,y_train)
```

Out[6]:
```
DecisionTreeClassifier()
```

In [7]:
```
# accuracy of our classification tree
clf.score(x_test,y_test)
```

Out[7]: 0.999633431085044

In [8]:
```
#Let's first visualize the tree on the data without doing any pre processing
clf2 = DecisionTreeClassifier(criterion = 'entropy', max_depth =24, min_samples_leaf= 1)
clf2.fit(x_train,y_train)
```

Out[8]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=24)
```

In [9]:
```
clf2.score(x_test,y_test)
```

Out[9]: 0.999633431085044

In [10]:
```
rand_clf = RandomForestClassifier(random_state=6)
```

In [11]:
```
rand_clf.fit(x_train,y_train)
```

Out[11]:
```
RandomForestClassifier(random_state=6)
```

In [12]:
```
rand_clf.score(x_test,y_test)
```

Out[12]: 1.0

In [13]:
```
# we are tuning three hyperparameters right now, we are passing the different values for both parameters
grid_param = {
 "n_estimators" : [90,100,115,130],
 'criterion': ['gini', 'entropy'],
 'max_depth' : range(2,20,1),
 'min_samples_leaf' : range(1,10,1),
 'min_samples_split': range(2,10,1),
 'max_features' : ['auto','log2']
}
```

In [14]:
```
grid_search = GridSearchCV(estimator=rand_clf,param_grid=grid_param,cv=5,n_jobs =-1,verbose = 3)
```

In [ ]:
```
#grid_search.fit(x_train,y_train)
```

In [16]:
```
rand_clf = RandomForestClassifier(criterion= 'entropy',
max_depth = 12,
max_features = 'log2',
min_samples_leaf = 1,
min_samples_split= 5,
n_estimators = 90,random_state=6)
```

In [17]:
```
rand_clf.fit(x_train,y_train)
```

Out[17]:
```
RandomForestClassifier(criterion='entropy', max_depth=12, max_features='log2',
                       min_samples_split=5, n_estimators=90, random_state=6)
```

In [18]:
```
rand_clf.score(x_test,y_test)
```

Out[18]: 1.0

In [19]:
```
# we are tuning three hyperparameters right now, we are passing the different values for both parameters
grid_param = {
 "n_estimators" : [90,100,115],
```

```
 'criterion': ['gini', 'entropy'],
 'min_samples_leaf' : [1,2,3,4,5],
 'min_samples_split': [4,5,6,7,8],
 'max_features' : ['auto','log2']
}
```

In [20]:
```
grid_search = GridSearchCV(estimator=rand_clf,param_grid=grid_param,cv=5,n_jobs =-1,verbose = 3)
```

In [21]:
```
grid_search.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 300 candidates, totalling 1500 fits
C:\Users\Personal\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter as it is also the default value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
```

Out[21]:

GridSearchCV

estimator: RandomForestClassifier

RandomForestClassifier

In [22]:
```
#Let's see the best parameters as per our grid search
grid_search.best_params_
```

Out[22]:
```
{'criterion': 'gini',
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 4,
 'n_estimators': 90}
```

In [23]:
```
rand_clf = RandomForestClassifier(criterion= 'entropy',
max_features = 'sqrt',
min_samples_leaf = 1,
min_samples_split= 4,
n_estimators = 115,random_state=6)
```

In [24]:
```
rand_clf.fit(x_train,y_train)
```

Out[24]:

RandomForestClassifier

```
RandomForestClassifier(criterion='entropy', min_samples_split=4,
                       n_estimators=115, random_state=6)
```

In [25]:
```
rand_clf.score(x_test,y_test)
```

Out[25]:
```
1.0
```