9) Apply Ensemble learning and evaluate the prediction

Task 1 - Deep Study of Ensemble Learning and Random Forest, Understand Begging and Stacking

Bagging (Bootstrap Aggregation)

In [1]:

```python
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
```

In [2]:

```python
from sklearn.datasets import load_breast_cancer
dataset = load_breast_cancer()
X = dataset.data
y = dataset.target
```

In [3]:

```python
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 [4]:

```python
# K-NeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

Out[4]:

```
▾ KNeighborsClassifier ⓘ ?
KNeighborsClassifier()
```

In [5]:

```python
knn.score(X_test, y_test)
```

Out[5]:

```
0.916083916083916
```

In [6]:

```python
bag_knn = BaggingClassifier(KNeighborsClassifier(n_neighbors=5),
                            n_estimators=10, max_samples=0.5,
                            bootstrap=True, random_state=3,oob_score=True)
```

In [7]:

```python
#Let's check the out of bag score
bag_knn.fit(X_train, y_train)
bag_knn.oob_score_
```

Out[7]:

```
0.9295774647887324
```

In [8]:

```python
bag_knn.score(X_test, y_test)
```

Out[8]:

```
0.9370629370629371
```

Pasting

In [9]:

```
pasting_knn = BaggingClassifier(KNeighborsClassifier(n_neighbors=5),
                                n_estimators=10, max_samples=0.5,
                                bootstrap=False, random_state=3)
```

In [10]:

```
pasting_knn.fit(X_train, y_train)
pasting_knn.score(X_test, y_test)
```

Out[10]:

```
0.9300699300699301
```

Stacking (Stacked Generalization)

In [3]:

```
import pandas as pd
import numpy as np
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
```

In [12]:

```
data = pd.read_csv("diabetes.csv")
data.head()
```

Out[12]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeF
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

In [13]:

```
data.describe()
```

Out[13]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000

In [14]:

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

In [15]:

```
# 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
```

In [16]:

```
# 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, rand
```

In [17]:

```
knn = KNeighborsClassifier()
knn.fit(x_train,y_train)
```

Out[17]:

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [18]:

```
knn.score(x_test,y_test)
```

Out[18]:

```
0.7402597402597403
```

In [19]:

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

Out[19]:

```
▾ SVC
SVC()
```

In [20]:

```
svm.score(x_test,y_test)
```

Out[20]:

```
0.7402597402597403
```

In [21]:

```
predict_val1 = knn.predict(val_train)
predict_val2 = svm.predict(val_train)
#predict_val2 = rand_clf.predict(val_train)
```

In [22]:

```
predict_val = np.column_stack((predict_val1,predict_val2))
```

```
predict_val
```

```
       [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 [23]:

```
predict_test1 = knn.predict(x_test)
predict_test2 = svm.predict(x_test)
#predict_test2 = rand_clf.predict(x_test)
```

In [24]:

```
predict_test = np.column_stack((predict_test1,predict_test2))
predict_test
```

```
Out[24]: 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 [25]:

```
svm = SVC()
svm.fit(predict_val,val_test)
```

Out[25]:

```
▾ SVC ⓘ ?

SVC()
```

In [26]:

```
svm.score(predict_test,y_test)
```

Out[26]:

```
0.7402597402597403
```

In [27]:

```
rand_clf = RandomForestClassifier()
rand_clf.fit(predict_val,val_test)
```

Out[27]:

```
▾ RandomForestClassifier ⓘ ?

RandomForestClassifier()
```

In [28]:

```
rand_clf.score(predict_test,y_test)
```

Out[28]:

```
0.7402597402597403
```

In [29]:

```
# we are tuning three hyperparameters right now, we are passing the different va
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 [30]:

```
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(estimator=rand_clf,param_grid=grid_param,cv=5,n_jobs
```

In [31]:

```
grid_search.fit(predict_val,val_test)
```

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

```
C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
\model_selection\_validation.py:547: FitFailedWarning:
750 fits failed out of a total of 1500.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_sc
ore='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
476 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages
\sklearn\model_selection\_validation.py", line 895, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages
\sklearn\base.py", line 1467, in wrapper
    estimator._validate_params()
  File "C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages
\sklearn\base.py", line 666, in _validate_params
    validate_parameter_constraints(
  File "C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages
\sklearn\utils\_param_validation.py", line 95, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' paramet
er of RandomForestClassifier must be an int in the range [1, inf), a float in the
range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got 'auto' instead.

--------------------------------------------------------------------------------
274 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages
\sklearn\model_selection\_validation.py", line 895, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages
\sklearn\base.py", line 1467, in wrapper
    estimator._validate_params()
  File "C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages
\sklearn\base.py", line 666, in _validate_params
    validate_parameter_constraints(
  File "C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages
\sklearn\utils\_param_validation.py", line 95, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' paramet
er of RandomForestClassifier must be an int in the range [1, inf), a float in the
range (0.0, 1.0], a str among {'sqrt', 'log2'} or None. Got 'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
\model_selection\_search.py:1051: UserWarning: One or more of the test scores are
non-finite: [       nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032
 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032 0.74231032]
  warnings.warn(
```

Out[31]:

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [32]:

```
grid_search.best_params_
```

Out[32]:

```
{'criterion': 'gini',
 'max_features': 'log2',
 'min_samples_leaf': 1,
 'min_samples_split': 4,
 'n_estimators': 90}
```

In [33]:

```
rand_clf = RandomForestClassifier(criterion='gini',max_features = 'log2',
                                  min_samples_leaf =1,min_samples_split= 4,
                                n_estimators =90)
rand_clf.fit(predict_val,val_test)
```

Out[33]:

```
▾                          RandomForestClassifier
RandomForestClassifier(max_features='log2', min_samples_split=4,
                       n_estimators=90)
```

Random Forests on winequality_red.csv

In [34]:

```
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
#from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
```

In [35]:

```
data = pd.read_csv("winequality-red.csv")
data
```

Out[35]:

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

1599 rows × 12 columns

In [36]:

```
data.describe()
```

Out[36]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	

In [37]:

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

In [38]:

```
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size = 0.30, random_st
```

In [39]:

```
#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[39]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [40]:

```
# accuracy of our classification tree
clf.score(x_test,y_test)
```

Out[40]:

```
0.6166666666666667
```

In [41]:

```
#let's first visualize the tree on the data without doing any pre processing
clf2 = DecisionTreeClassifier(criterion = 'entropy', max_depth =24, min_samples_
clf2.fit(x_train,y_train)
```

Out[41]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=24)
```

In [42]:

```
clf2.score(x_test,y_test)
```

Out[42]:

```
0.61875
```

In [43]:

```
rand_clf = RandomForestClassifier(random_state=6)
rand_clf.fit(x_train,y_train)
```

Out[43]:

```
▾ RandomForestClassifier

RandomForestClassifier(random_state=6)
```

In [44]:

```
rand_clf.score(x_test,y_test)
```

Out[44]:

```
0.6708333333333333
```

In [45]:

```
# we are tuning three hyperparameters right now, we are passing the different va
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 [46]:

```
grid_search = GridSearchCV(estimator=rand_clf,param_grid=grid_param,cv=5,n_jobs
```

In [47]:

```
#grid_search.fit(x_train,y_train)
```

In [48]:

```
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 [49]:

```
rand_clf.fit(x_train,y_train)
```

Out[49]:

```
▾ RandomForestClassifier

RandomForestClassifier(criterion='entropy', max_depth=12, max_features
='log2',
                       min_samples_split=5, n_estimators=90, random_sta
te=6)
```

In [50]:

```
rand_clf.score(x_test,y_test)
```

Out[50]:

```
0.6604166666666667
```

In [51]:

```
# we are tuning three hyperparameters right now, we are passing the different va
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 [52]:

```
grid_search = GridSearchCV(estimator=rand_clf,param_grid=grid_param,cv=5,n_jobs
```

In [53]:

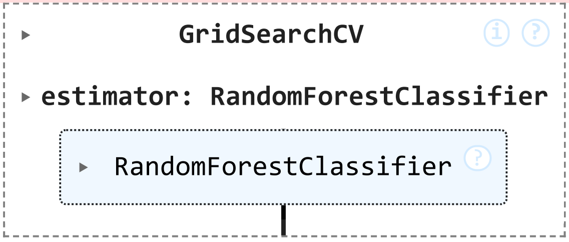
```
grid_search.fit(x_train,y_train)
```

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

```
C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
\model_selection\_validation.py:547: FitFailedWarning: 
750 fits failed out of a total of 1500.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_sc
ore='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
471 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages
\sklearn\model_selection\_validation.py", line 895, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages
\sklearn\base.py", line 1467, in wrapper
    estimator._validate_params()
  File "C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages
\sklearn\base.py", line 666, in _validate_params
    validate_parameter_constraints(
  File "C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages
\sklearn\utils\_param_validation.py", line 95, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' paramet
er of RandomForestClassifier must be an int in the range [1, inf), a float in the
range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got 'auto' instead.

--------------------------------------------------------------------------------
279 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages
\sklearn\model_selection\_validation.py", line 895, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages
\sklearn\base.py", line 1467, in wrapper
    estimator._validate_params()
  File "C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages
\sklearn\base.py", line 666, in _validate_params
    validate_parameter_constraints(
  File "C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages
\sklearn\utils\_param_validation.py", line 95, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' paramet
er of RandomForestClassifier must be an int in the range [1, inf), a float in the
range (0.0, 1.0], a str among {'sqrt', 'log2'} or None. Got 'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\nayan\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
\model_selection\_search.py:1051: UserWarning: One or more of the test scores are
non-finite: [       nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
```

```
       nan        nan        nan        nan        nan        nan
       nan        nan        nan 0.65683056 0.6559417  0.65595372
0.66220372 0.66040999 0.66310058 0.65595372 0.65685058 0.6586443
0.65325112 0.65234225 0.65324712 0.66489029 0.66578716 0.66132687
0.66131086 0.66131486 0.65863629 0.67202514 0.67114029 0.66935058
0.66669202 0.67115231 0.66133088 0.65862028 0.66220372 0.66130285
0.65505685 0.66131486 0.66132287 0.65683857 0.65951714 0.65862028
0.65683857 0.65951714 0.65862028 0.65683857 0.65951714 0.65862028
0.65595772 0.65596573 0.65684257 0.66130285 0.66398142 0.66309657
0.64970772 0.65328716 0.65060458 0.64970772 0.65328716 0.65060458
0.64970772 0.65328716 0.65060458 0.64970772 0.65328716 0.65060458
0.64970772 0.65328716 0.65060458 0.65595372 0.65059657 0.65505685
0.65595372 0.65059657 0.65505685 0.65595372 0.65059657 0.65505685
0.65595372 0.65059657 0.65505685 0.65595372 0.65059657 0.65505685
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan 0.66399343 0.666668   0.66756887
0.67560058 0.67471172 0.67114029 0.66934257 0.67023543 0.66934257
0.67648943 0.6684417  0.666668   0.66130685 0.66577114 0.660414
0.66130685 0.66844571 0.67202915 0.66129484 0.6621877  0.66129484
0.6621877  0.67203315 0.6711443  0.66935058 0.67203315 0.66935058
0.65504484 0.65951313 0.660418   0.64879484 0.654164   0.65059257
0.64879484 0.654164   0.65059257 0.64879484 0.654164   0.65059257
0.65326714 0.64880285 0.64969971 0.65504484 0.65058056 0.65772341
0.65506086 0.65505685 0.65504484 0.65506086 0.65505685 0.65504484
0.65506086 0.65505685 0.65504484 0.65506086 0.65505685 0.65504484
0.65506086 0.65505685 0.65504484 0.65059257 0.64611627 0.65415999
0.65059257 0.64611627 0.65415999 0.65059257 0.64611627 0.65415999
0.65059257 0.64611627 0.65415999 0.65059257 0.64611627 0.65415999]
  warnings.warn(
```

Out[53]:



In [54]:

```
#let's see the best parameters as per our grid search
grid_search.best_params_
```

Out[54]:

```
{'criterion': 'entropy',
 'max_features': 'log2',
 'min_samples_leaf': 1,
 'min_samples_split': 7,
 'n_estimators': 90}
```

In [55]:

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

In [56]:
```
rand_clf.fit(x_train,y_train)
```

Out[56]:
```
▾                      RandomForestClassifier

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

In [57]:
```
rand_clf.score(x_test,y_test)
```

Out[57]:
```
0.6729166666666667
```

Task 2 - Implementation of Random Forest Classifier on Rice Classification

In [58]:
```
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
#from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
```

In [59]:
```
data = pd.read_csv("riceClassification.csv")
data
```

Out[59]:

	id	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Equi
0	1	4537	92.229316	64.012769	0.719916	4677	
1	2	2872	74.691881	51.400454	0.725553	3015	
2	3	3048	76.293164	52.043491	0.731211	3132	
3	4	3073	77.033628	51.928487	0.738639	3157	
4	5	3693	85.124785	56.374021	0.749282	3802	
...	...	...	...	...	...	...	
18180	18181	5853	148.624571	51.029281	0.939210	6008	
18181	18182	7585	169.593996	58.141659	0.939398	7806	
18182	18183	6365	154.777085	52.908085	0.939760	6531	
18183	18184	5960	151.397924	51.474600	0.940427	6189	
18184	18185	6134	153.081981	51.590606	0.941500	6283	

18185 rows × 12 columns

In [60]:
```
data.describe()
```

Out[60]:

	id	Area	MajorAxisLength	MinorAxisLength	Eccentricity	
count	18185.000000	18185.000000	18185.000000	18185.000000	18185.000000	1
mean	9093.000000	7036.492989	151.680754	59.807851	0.915406	
std	5249.701658	1467.197150	12.376402	10.061653	0.030575	
min	1.000000	2522.000000	74.133114	34.409894	0.676647	
25%	4547.000000	5962.000000	145.675910	51.393151	0.891617	
50%	9093.000000	6660.000000	153.883750	55.724288	0.923259	
75%	13639.000000	8423.000000	160.056214	70.156593	0.941372	
max	18185.000000	10210.000000	183.211434	82.550762	0.966774	1

In [61]:
```
data.describe()
```

Out[61]:

	id	Area	MajorAxisLength	MinorAxisLength	Eccentricity	
count	18185.000000	18185.000000	18185.000000	18185.000000	18185.000000	1
mean	9093.000000	7036.492989	151.680754	59.807851	0.915406	
std	5249.701658	1467.197150	12.376402	10.061653	0.030575	
min	1.000000	2522.000000	74.133114	34.409894	0.676647	
25%	4547.000000	5962.000000	145.675910	51.393151	0.891617	
50%	9093.000000	6660.000000	153.883750	55.724288	0.923259	
75%	13639.000000	8423.000000	160.056214	70.156593	0.941372	
max	18185.000000	10210.000000	183.211434	82.550762	0.966774	1

In [62]:

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

In [63]:

```
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size = 0.30, random_st
```

In [64]:

```
#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[64]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [65]:

```
# accuracy of our classification tree
clf.score(x_test,y_test)
```

Out[65]:

```
0.999633431085044
```

In [66]:

```
#let's first visualize the tree on the data without doing any pre processing
clf2 = DecisionTreeClassifier(criterion = 'entropy', max_depth =24, min_samples_
clf2.fit(x_train,y_train)
```

Out[66]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=24)
```

In [67]:

```
clf2.score(x_test,y_test)
```

Out[67]:

```
0.999633431085044
```

In [68]:

```
rand_clf = RandomForestClassifier(random_state=6)
rand_clf.fit(x_train,y_train)
```

Out[68]:

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=6)
```

In [69]:

```
rand_clf.score(x_test,y_test)
```

Out[69]:

```
1.0
```

In [70]:

```
# we are tuning three hyperparameters right now, we are passing the different va
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 [71]:

```
grid_search = GridSearchCV(estimator=rand_clf,
                           param_grid=grid_param,cv=5,n_jobs =-1,verbose = 3)
```

In [72]:

```
#grid_search.fit(x_train,y_train)
```

In [73]:

```
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 [74]:

```
rand_clf.fit(x_train,y_train)
```

Out[74]:

▾ RandomForestClassifier

```
RandomForestClassifier(criterion='entropy', max_depth=12, max_features
='log2',
                       min_samples_split=5, n_estimators=90, random_sta
te=6)
```

In [75]:

```
rand_clf.score(x_test,y_test)
```

Out[75]:

```
1.0
```

In [76]:

```
# we are tuning three hyperparameters right now, we are passing the different va
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 [ ]:

```
grid_search.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 20736 candidates, totalling 103680 fits
```

In [ ]:

```
#let's see the best parameters as per our grid search
grid_search.best_params_
```

```
 '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_featur
es='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 RandomForestClassi
fiers 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
```