BC_DT_SVG

August 15, 2020

[61]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

[62]:
```
#Import data file: Breast Cancer
BC_df = pd.read_csv("C:/Venu/UCI DataSets/breast-cancer.data",delimiter=',',␣
 ↪header=None,names=['Class', 'Age', 'Menopause', 'Tumor-size', 'Inv-nodes',␣
 ↪'Node-caps', 'Deg-malig', 'Breast', 'Breast-quad', 'IR-Radiat'])
```

[63]:
```
#Checking for Missing Values/Null Values
BC_df.isna().values.any()
BC_df.isna().sum()
BC_df.isnull()
BC_df.isnull().sum()
```

[63]:
```
Class          0
Age            0
Menopause      0
Tumor-size     0
Inv-nodes      0
Node-caps      8
Deg-malig      0
Breast         0
Breast-quad    1
IR-Radiat      0
dtype: int64
```

[64]:
```
#Removing the Missing rows
BC1_df = BC_df.dropna()
BC1_df.isna().sum()
```

```
[64]: Class          0
      Age            0
      Menopause      0
      Tumor-size     0
      Inv-nodes      0
      Node-caps      0
      Deg-malig      0
      Breast         0
      Breast-quad    0
      IR-Radiat      0
      dtype: int64
```

```
[65]: #Response variable counts
      BC1_df.Class.value_counts()
```

```
[65]: no-recurrence-events    196
      recurrence-events        81
      Name: Class, dtype: int64
```

```
[66]: #Converting the categorical features as dummy variables
      encoded_BC1_df = pd.get_dummies(BC1_df,drop_first=True)
      X_features = list(encoded_BC1_df.columns)
      X_features
      X_features.remove('Class_recurrence-events')
      X_features
      encoded_BC1_df[X_features].columns
```

```
[66]: Index(['Deg-malig', 'Age_30-39', 'Age_40-49', 'Age_50-59', 'Age_60-69',
             'Age_70-79', 'Menopause_lt40', 'Menopause_premeno', 'Tumor-size_10-14',
             'Tumor-size_15-19', 'Tumor-size_20-24', 'Tumor-size_25-29',
             'Tumor-size_30-34', 'Tumor-size_35-39', 'Tumor-size_40-44',
             'Tumor-size_45-49', 'Tumor-size_5-9', 'Tumor-size_50-54',
             'Inv-nodes_12-14', 'Inv-nodes_15-17', 'Inv-nodes_24-26',
             'Inv-nodes_3-5', 'Inv-nodes_6-8', 'Inv-nodes_9-11', 'Node-caps_yes',
             'Breast_right', 'Breast-quad_left_low', 'Breast-quad_left_up',
             'Breast-quad_right_low', 'Breast-quad_right_up', 'IR-Radiat_yes'],
            dtype='object')
```

```
[67]: #Final X/Predictors set
      X = encoded_BC1_df[X_features]
      X.iloc[0:5,]
```

```
[67]:    Deg-malig  Age_30-39  Age_40-49  Age_50-59  Age_60-69  Age_70-79  \
      0          3          1          0          0          0          0
      1          2          0          1          0          0          0
      2          2          0          1          0          0          0
      3          2          0          0          0          1          0
```

```
4          2          0          1          0          0          0

   Menopause_lt40  Menopause_premeno  Tumor-size_10-14  Tumor-size_15-19  ...  \
0               0                  1                 0                 0  ...
1               0                  1                 0                 0  ...
2               0                  1                 0                 0  ...
3               0                  0                 0                 1  ...
4               0                  1                 0                 0  ...

   Inv-nodes_3-5  Inv-nodes_6-8  Inv-nodes_9-11  Node-caps_yes  Breast_right  \
0              0              0               0              0             0
1              0              0               0              0             1
2              0              0               0              0             0
3              0              0               0              0             1
4              0              0               0              0             1

   Breast-quad_left_low  Breast-quad_left_up  Breast-quad_right_low  \
0                     1                    0                      0
1                     0                    0                      0
2                     1                    0                      0
3                     0                    1                      0
4                     0                    0                      1

   Breast-quad_right_up  IR-Radiat_yes
0                     0              0
1                     1              0
2                     0              0
3                     0              0
4                     0              0

[5 rows x 31 columns]
```

```
[68]: #Final Response Set
      Y = encoded_BC1_df['Class_recurrence-events']
      Y.iloc[0:5,]
```

```
[68]: 0    0
      1    0
      2    0
      3    0
      4    0
      Name: Class_recurrence-events, dtype: uint8
```

```
[69]: #Splitting into training and testing sets
      train_X,test_X,train_y,test_y = train_test_split(X,Y,train_size=0.
       ↪7,random_state=42)
```

[70]:

```
#Decision Trees using Gini
BC_DT_gini = DecisionTreeClassifier(criterion='gini',max_depth=3)
BC_DT_gini.fit(train_X,train_y)
```

[70]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

[71]:

```
#Predicting the class of the test set
BC_tree_gini_predict = BC_DT_gini.predict(test_X)
```

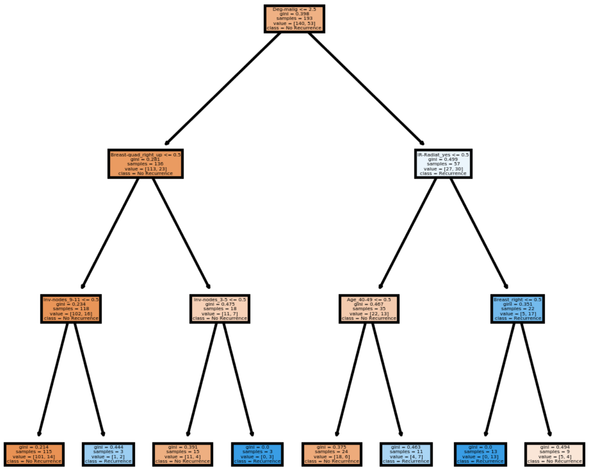
[72]:

```
#AUC, Classification Report and Confusion Matrix
print("AUC: \n", metrics.roc_auc_score(test_y,BC_tree_gini_predict))
print("ROC Curve:\n",metrics.roc_curve(test_y,BC_tree_gini_predict))
print("Classification Report: \n", metrics.
 ↪classification_report(test_y,BC_tree_gini_predict))
print("Confusion Metrics:\n",metrics.
 ↪confusion_matrix(test_y,BC_tree_gini_predict))
```

```
AUC:
 0.5625
ROC Curve:
 (array([0.        , 0.01785714, 1.        ]), array([0.        , 0.14285714, 1.
]), array([2, 1, 0]))
Classification Report:
               precision    recall  f1-score   support

           0       0.70      0.98      0.81        56
           1       0.80      0.14      0.24        28

    accuracy                           0.70        84
   macro avg       0.75      0.56      0.53        84
weighted avg       0.73      0.70      0.62        84

Confusion Metrics:
 [[55  1]
 [24  4]]
```

[73]:

```
#Ploting the decision tree based on Gini's index
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
fn = encoded_BC1_df[X_features].columns
cn = ["No Recurrence","Recurrence"]
tree.plot_tree(BC_DT_gini,feature_names = fn, class_names=cn,filled = True)
fig.savefig("BC_DT_tree_gini.png")
```



[74]:
```
#Decision Trees using Entropy
BC_DT_entropy = DecisionTreeClassifier(criterion='entropy',max_depth=3)
BC_DT_entropy.fit(train_X,train_y)
```

[74]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

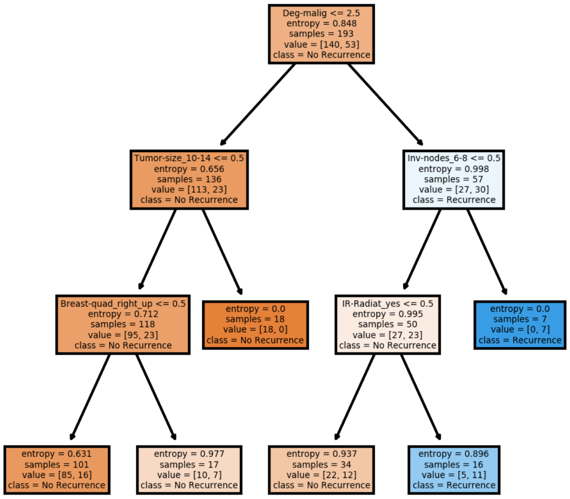
[75]:
```
#Predicting the class of the test set
BC_tree_entropy_predict = BC_DT_entropy.predict(test_X)
```

[76]:
```
#AUC, Classification Report and Confusion Matrix
print("AUC:\n",metrics.roc_auc_score(test_y,BC_tree_entropy_predict))
print("ROC Curve:\n",metrics.roc_curve(test_y,BC_tree_entropy_predict))
print("Classification Report:\n",metrics.
 ↪classification_report(test_y,BC_tree_entropy_predict))
print("Confusion Metrics:\n",metrics.
 ↪confusion_matrix(test_y,BC_tree_entropy_predict))
```

```
AUC:
 0.5357142857142857
ROC Curve:
 (array([0.        , 0.07142857, 1.        ]), array([0.        , 0.14285714, 1.
]), array([2, 1, 0]))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.68      0.93      0.79        56
           1       0.50      0.14      0.22        28

    accuracy                           0.67        84
   macro avg       0.59      0.54      0.51        84
weighted avg       0.62      0.67      0.60        84

Confusion Metrics:
 [[52  4]
 [24  4]]
```

```
[77]: #Ploting the decision tree based on Entropy
      fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
      tree.plot_tree(BC_DT_entropy,feature_names = fn, class_names=cn,filled = True);
      fig.savefig("BC_DT_tree_entropy.png")
```



```
[78]: #Random Forest Model
      from sklearn.ensemble import RandomForestClassifier
      BC_tree_RF = RandomForestClassifier(max_depth = 10, n_estimators=10)
```

```
[79]: BC_tree_RF.fit(train_X,train_y)
```

```
[79]: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                             criterion='gini', max_depth=10, max_features='auto',
                             max_leaf_nodes=None, max_samples=None,
                             min_impurity_decrease=0.0, min_impurity_split=None,
                             min_samples_leaf=1, min_samples_split=2,
                             min_weight_fraction_leaf=0.0, n_estimators=10,
                             n_jobs=None, oob_score=False, random_state=None,
                             verbose=0, warm_start=False)
```

[80]:

```
y_pred_rf = BC_tree_RF.predict(test_X)
y_pred_rf
```

[80]:

```
array([0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 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, 0, 0], dtype=uint8)
```

[81]:

```
print("AUC:\n",metrics.roc_auc_score(test_y,y_pred_rf))
```

```
AUC:
 0.5714285714285714
```

[82]:

```
print("Confusion Matrix\n", (confusion_matrix(test_y,y_pred_rf)))
```

```
Confusion Matrix
 [[50  6]
 [21  7]]
```

[83]:

```
print("Accuracy:\n",(accuracy_score(test_y,y_pred_rf)*100))
```

```
Accuracy:
 67.85714285714286
```

[84]:

```
print("Classification Report\n", (classification_report(test_y, y_pred_rf)))
```

```
Classification Report
               precision    recall  f1-score   support

           0       0.70      0.89      0.79        56
           1       0.54      0.25      0.34        28

    accuracy                           0.68        84
   macro avg       0.62      0.57      0.56        84
weighted avg       0.65      0.68      0.64        84
```

[85]:

```
#GRID Search
from sklearn.model_selection import GridSearchCV
tuned_parameters = [{'max_depth': [10,15],'n_estimators':␣
 ↪[10,20],'max_features':['sqrt',0.2]}]
```

```
BC_tree_clf = RandomForestClassifier()
BC_Grid_clf = GridSearchCV(BC_tree_clf,tuned_parameters,cv=5,scoring='roc_auc')
BC_Grid_clf.fit(train_X,train_y)
```

```
[85]: GridSearchCV(cv=5, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini', max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators=100, n_jobs=None,
                                                    oob_score=False,
                                                    random_state=None, verbose=0,
                                                    warm_start=False),
                   iid='deprecated', n_jobs=None,
                   param_grid=[{'max_depth': [10, 15], 'max_features': ['sqrt', 0.2],
                                'n_estimators': [10, 20]}],
                   pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                   scoring='roc_auc', verbose=0)
```

```
[86]: BC_Grid_clf.best_score_
```

```
[86]: 0.748538961038961
```

```
[87]: BC_Grid_clf.best_params_
```

```
[87]: {'max_depth': 10, 'max_features': 0.2, 'n_estimators': 20}
```

```
[88]: BC_bestTree = RandomForestClassifier(max_depth=10,n_estimators=20,max_features=0.2)
      BC_bestTree.fit(train_X,train_y)
```

```
[88]: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                             criterion='gini', max_depth=10, max_features=0.2,
                             max_leaf_nodes=None, max_samples=None,
                             min_impurity_decrease=0.0, min_impurity_split=None,
                             min_samples_leaf=1, min_samples_split=2,
                             min_weight_fraction_leaf=0.0, n_estimators=20,
                             n_jobs=None, oob_score=False, random_state=None,
                             verbose=0, warm_start=False)
```

[89]:
```
y_pred_bestTree = BC_bestTree.predict(test_X)
```

[90]:
```
print("Confusion Matrix\n", (confusion_matrix(test_y,y_pred_bestTree)))
```

```
Confusion Matrix
 [[52  4]
 [18 10]]
```

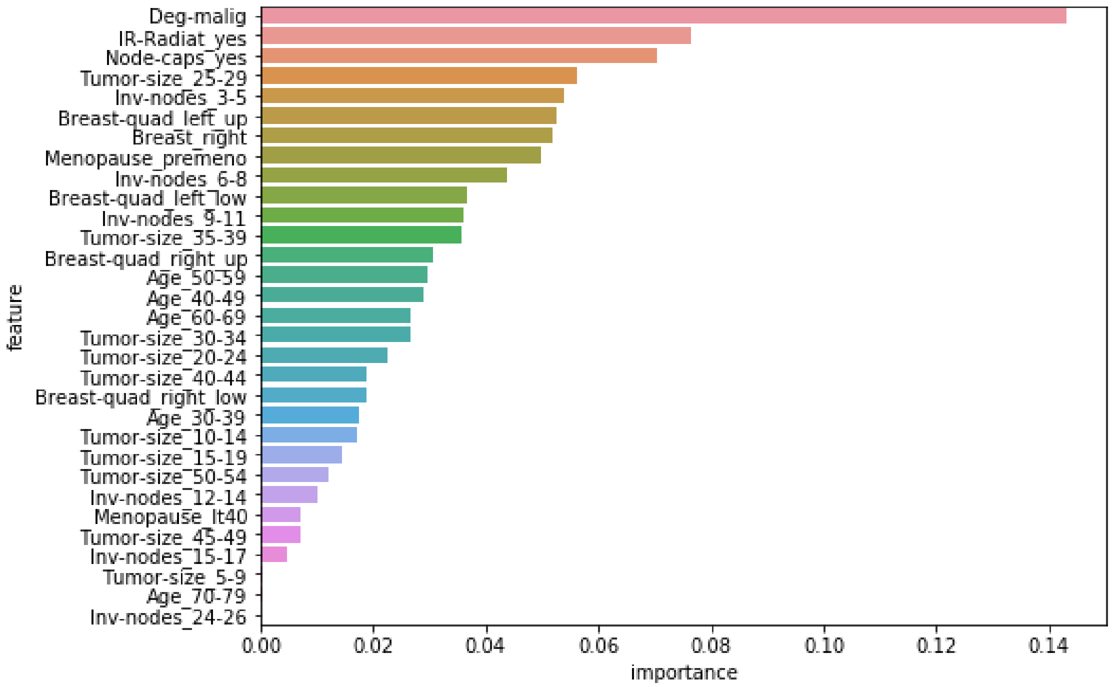
[91]:
```
print("Accuracy:\n",(accuracy_score(test_y,y_pred_bestTree)*100))
```

```
Accuracy:
 73.80952380952381
```

[92]:
```
print("Classification Report\n", (classification_report(test_y,␣
 ↪y_pred_bestTree)))
```

```
Classification Report
               precision    recall  f1-score   support

           0       0.74      0.93      0.83        56
           1       0.71      0.36      0.48        28

    accuracy                           0.74        84
   macro avg       0.73      0.64      0.65        84
weighted avg       0.73      0.74      0.71        84
```

[93]:
```
#Important Features for the bestTree
import seaborn as sn
feature_rank = pd.DataFrame({'feature': train_X.columns,'importance':
 ↪BC_bestTree.feature_importances_})
feature_rank = feature_rank.sort_values('importance', ascending = False)
plt.figure(figsize=(8,6))
sn.barplot(y = 'feature',x = 'importance',data = feature_rank);
```



[94]:

```
#Adaboost Classifier
from sklearn.ensemble import AdaBoostClassifier
carseats_adaboost = DecisionTreeClassifier()
BC_tree_Adaboost = AdaBoostClassifier(carseats_adaboost, n_estimators=50)
BC_tree_Adaboost.fit(train_X,train_y)
```

[94]:

```
AdaBoostClassifier(algorithm='SAMME.R',
                   base_estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                                         class_weight=None,
                                                         criterion='gini',
                                                         max_depth=None,
                                                         max_features=None,
                                                         max_leaf_nodes=None,
min_impurity_decrease=0.0,
min_impurity_split=None,
                                                         min_samples_leaf=1,
                                                         min_samples_split=2,
min_weight_fraction_leaf=0.0,
                                                         presort='deprecated',
                                                         random_state=None,
                                                         splitter='best'),
                   learning_rate=1.0, n_estimators=50, random_state=None)
```

```
[95]: y_pred_Adaboost = BC_tree_Adaboost.predict(test_X)
      y_pred_Adaboost
```

```
[95]: array([0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
             0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
             1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0], dtype=uint8)
```

```
[96]: print("AUC:\n",metrics.roc_auc_score(test_y,y_pred_Adaboost))
```

```
AUC:
 0.5892857142857143
```

```
[97]: print("Confusion Matrix\n", (confusion_matrix(test_y,y_pred_Adaboost)))
```

```
Confusion Matrix
 [[46 10]
 [18 10]]
```

```
[98]: print("Accuracy:\n",(accuracy_score(test_y,y_pred_Adaboost)*100))
```

```
Accuracy:
 66.66666666666666
```

```
[99]: print("Classification Report\n", (classification_report(test_y,␣
       ↪y_pred_Adaboost)))
```

```
Classification Report
               precision    recall  f1-score   support

           0       0.72      0.82      0.77        56
           1       0.50      0.36      0.42        28

    accuracy                           0.67        84
   macro avg       0.61      0.59      0.59        84
weighted avg       0.65      0.67      0.65        84
```

```
[100]: #Gradient Boosting Classifier
       from sklearn.ensemble import GradientBoostingClassifier
       BC_tree_gradient = GradientBoostingClassifier(n_estimators = 500, max_depth =␣
        ↪10)
       BC_tree_gradient.fit(train_X,train_y)
```

```
[100]: GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                                  learning_rate=0.1, loss='deviance', max_depth=10,
                                  max_features=None, max_leaf_nodes=None,
                                  min_impurity_decrease=0.0, min_impurity_split=None,
                                  min_samples_leaf=1, min_samples_split=2,
                                  min_weight_fraction_leaf=0.0, n_estimators=500,
                                  n_iter_no_change=None, presort='deprecated',
                                  random_state=None, subsample=1.0, tol=0.0001,
                                  validation_fraction=0.1, verbose=0,
                                  warm_start=False)
```

```
[101]: y_pred_gradient = BC_tree_gradient.predict(test_X)
```

```
[102]: print("Confusion Matrix\n", (confusion_matrix(test_y,y_pred_gradient)))
```

```
Confusion Matrix
 [[45 11]
 [20  8]]
```

```
[103]: print("Accuracy:\n",(accuracy_score(test_y,y_pred_gradient)*100))
```

```
Accuracy:
 63.095238095238095
```

```
[104]: print("Classification Report\n", (classification_report(test_y,␣
        ↪y_pred_gradient)))
```

```
Classification Report
               precision    recall  f1-score   support

           0       0.69      0.80      0.74        56
           1       0.42      0.29      0.34        28

    accuracy                           0.63        84
   macro avg       0.56      0.54      0.54        84
weighted avg       0.60      0.63      0.61        84
```

```
[105]: print("AUC:\n",metrics.roc_auc_score(test_y,y_pred_gradient))
```

```
AUC:
 0.5446428571428571
```

```
[ ]:
```