BC_DT_SVG

September 3, 2020

[2]:
```
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 roc_curve
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_curve
```

[3]:
```
#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'])
```

[4]:
```
#Checking for Missing Values/Null Values
BC_df.isna().values.any()
BC_df.isna().sum()
BC_df.isnull()
BC_df.isnull().sum()
```

[4]:
```
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
```

[5]:
```
#Removing the Missing rows
BC1_df = BC_df.dropna()
BC1_df.isna().sum()
```

[5]:
```
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
```

[6]:
```
#Response variable counts
BC1_df.Class.value_counts()
```

[6]:
```
no-recurrence-events    196
recurrence-events        81
Name: Class, dtype: int64
```

[7]:
```
#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
```

[7]:
```
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')
```

[8]:
```
#Final X/Predictors set
X = encoded_BC1_df[X_features]
X.iloc[0:5,]
```

[8]:
```
   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]
```

[9]:
```
#Final Response Set
Y = encoded_BC1_df['Class_recurrence-events']
Y.iloc[0:5,]
```

[9]:
```
0    0
1    0
2    0
3    0
4    0
Name: Class_recurrence-events, dtype: uint8
```

[10]:
```
#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)
```

[11]:
```
#Decision Trees using Gini
BC_DT_gini = DecisionTreeClassifier(criterion='gini',max_depth=3)
BC_DT_gini.fit(train_X,train_y)
```

[11]:
```
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')
```

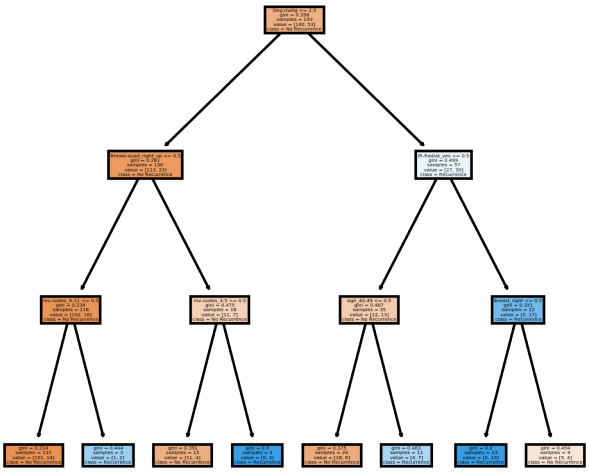
[12]:
```
#Predicting the class of the test set
BC_tree_gini_predict = BC_DT_gini.predict(test_X)
```

[13]:
```
#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]]
```

[14]:
```
#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")
```



[15]:
```
#Decision Trees using Entropy
BC_DT_entropy = DecisionTreeClassifier(criterion='entropy',max_depth=3)
BC_DT_entropy.fit(train_X,train_y)
```

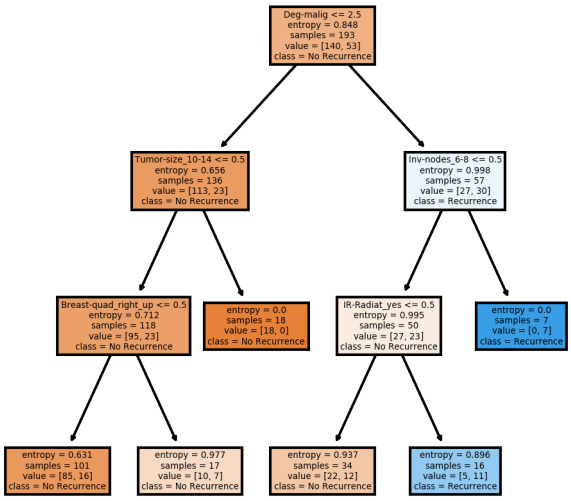
[15]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

[16]:
```
#Predicting the class of the test set
BC_tree_entropy_predict = BC_DT_entropy.predict(test_X)
```

[17]:
```
#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]]
```

[18]:
```
#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")
```



[19]:
```
#Random Forest Model
from sklearn.ensemble import RandomForestClassifier
BC_tree_RF = RandomForestClassifier(max_depth = 10, n_estimators=10)
```

[20]:
```
BC_tree_RF.fit(train_X,train_y)
```

[20]:
```
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)
```

[21]:
```
y_pred_rf = BC_tree_RF.predict(test_X)
y_pred_rf
```

[21]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 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, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0], dtype=uint8)
```

[22]:
```
print("AUC:\n",metrics.roc_auc_score(test_y,y_pred_rf))
```

```
AUC:
 0.5535714285714286
```

[23]:
```
print("Confusion Matrix\n", (confusion_matrix(test_y,y_pred_rf)))
```

```
Confusion Matrix
 [[50  6]
 [22  6]]
```

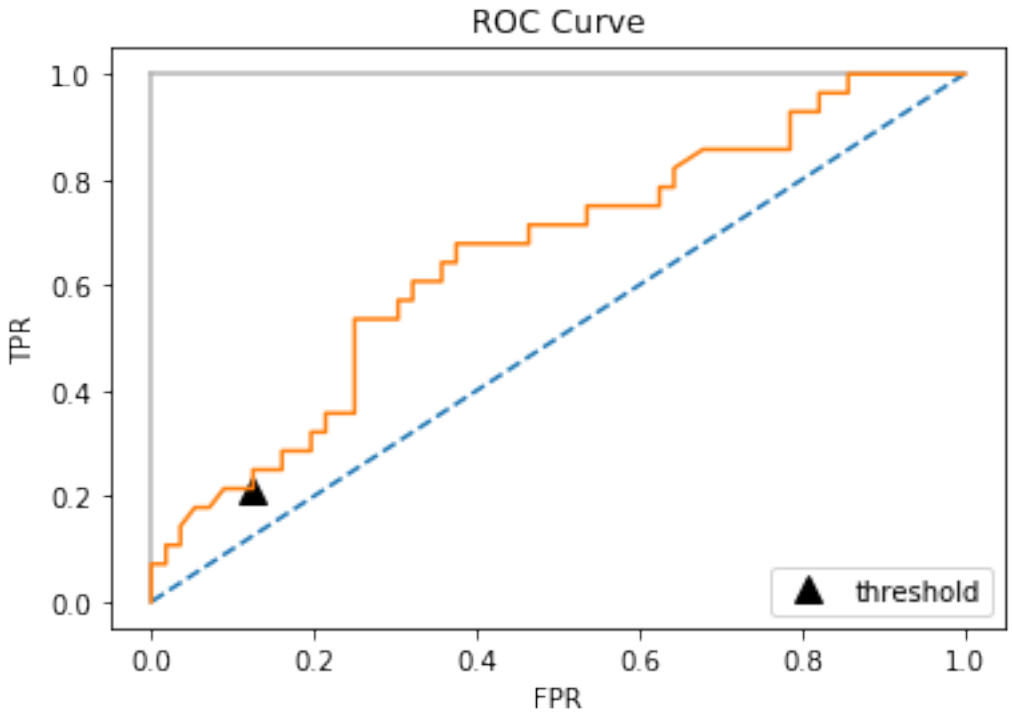
[24]:
```
print("Accuracy:\n",(accuracy_score(test_y,y_pred_rf)*100))
```

```
Accuracy:
 66.66666666666666
```

[25]:
```
print("Classification Report\n", (classification_report(test_y, y_pred_rf)))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.69      0.89      0.78        56
           1       0.50      0.21      0.30        28

    accuracy                           0.67        84
   macro avg       0.60      0.55      0.54        84
weighted avg       0.63      0.67      0.62        84
```

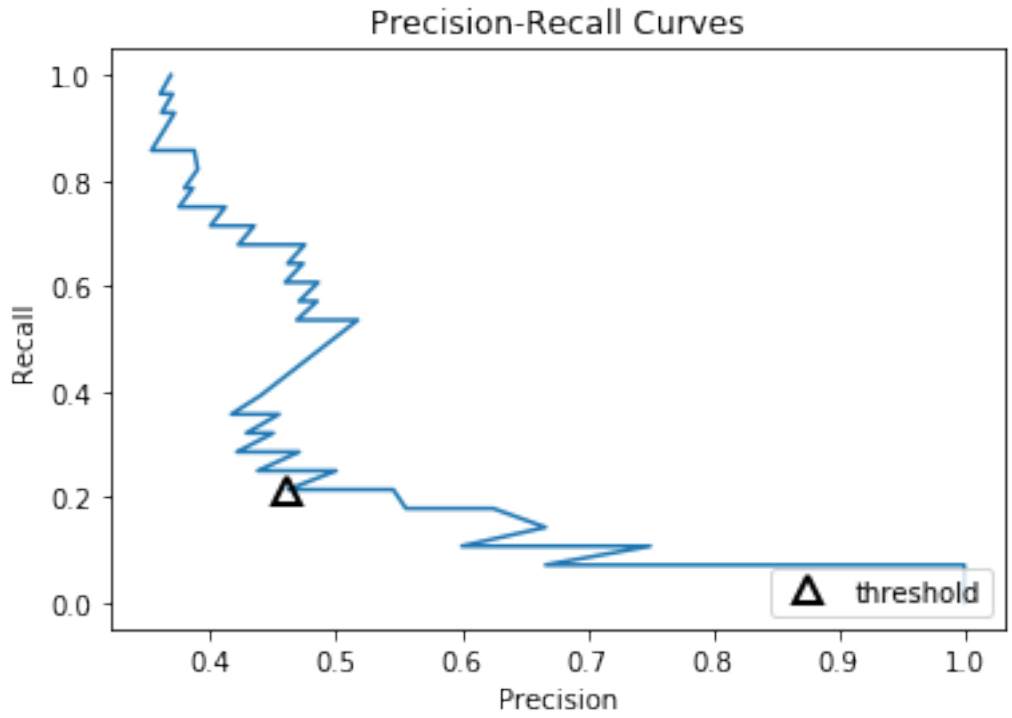
[62]:
```
# Plot ROC curve - Random Forest
fpr, tpr, thresholds = roc_curve(test_y,BC_tree_RF.predict_proba(test_X)[:,1])
plt.clf()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
close_default = np.argmin(np.abs(thresholds-0.5))
plt.
 ↪plot(fpr[close_default],tpr[close_default],"^",c='k',markersize=10,label='threshold')
plt.plot(fpr,tpr)
plt.legend(loc=4)
plt.show()
```



[78]:
```
# Plot Precision Recall curve - Random Forest
precision, recall, thresholds = precision_recall_curve(test_y,BC_tree_RF.
 ↪predict_proba(test_X)[:,1])
plt.plot(precision,recall)
close_default = np.argmin(np.abs(thresholds-0.5))
plt.
 ↪plot(precision[close_default],recall[close_default],"^",c='k',markersize=10,label='threshol
plt.title("Precision-Recall Curves")
```

```
plt.xlabel("Precision")
plt.ylabel("Recall")
plt.legend(loc=4)
```

[78]: <matplotlib.legend.Legend at 0x22a45afd8d0>



[27]:
```
#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)
```

[27]:
```
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)
```

[28]:
```
BC_Grid_clf.best_score_
```

[28]:
```
0.7613636363636365
```

[29]:
```
BC_Grid_clf.best_params_
```

[29]:
```
{'max_depth': 10, 'max_features': 'sqrt', 'n_estimators': 20}
```

[30]:
```
BC_bestTree =␣
↪RandomForestClassifier(max_depth=10,n_estimators=10,max_features=0.2)
BC_bestTree.fit(train_X,train_y)
```

[30]:
```
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=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

[31]:
```
y_pred_bestTree = BC_bestTree.predict(test_X)
```

[32]:
```
print("Confusion Matrix\n", (confusion_matrix(test_y,y_pred_bestTree)))
```

```
Confusion Matrix
 [[52  4]
 [19  9]]
```

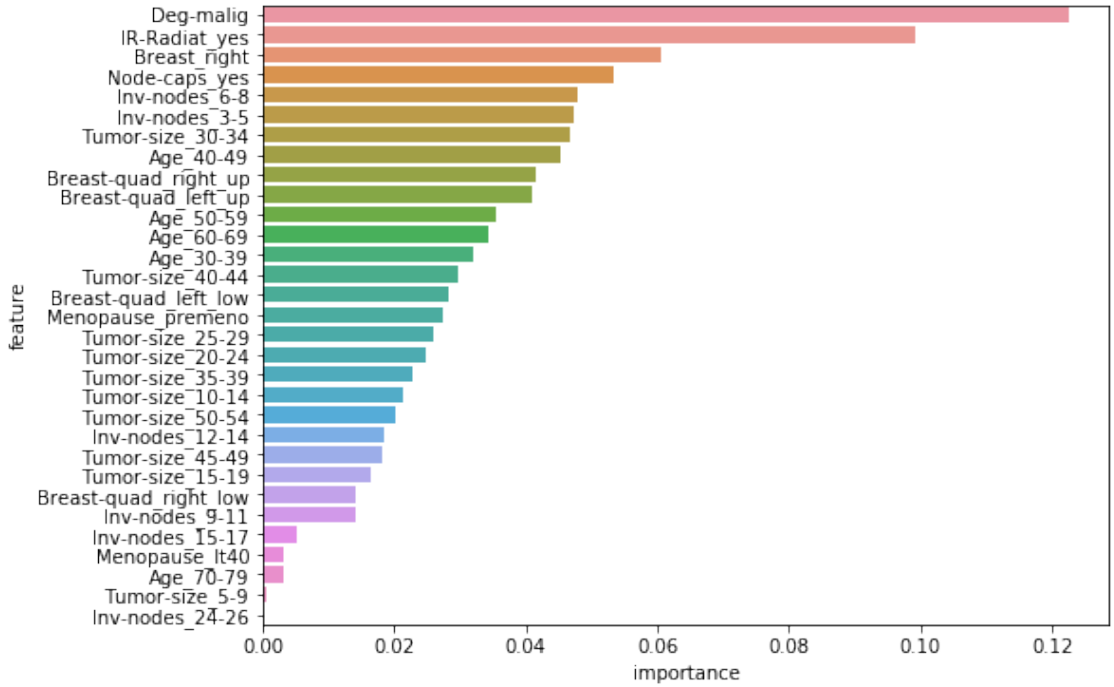
[33]:
```
print("Accuracy:\n",(accuracy_score(test_y,y_pred_bestTree)*100))
```

```
Accuracy:
 72.61904761904762
```

[34]:
```
print("Classification Report\n", (classification_report(test_y,␣
 ↪y_pred_bestTree)))
```

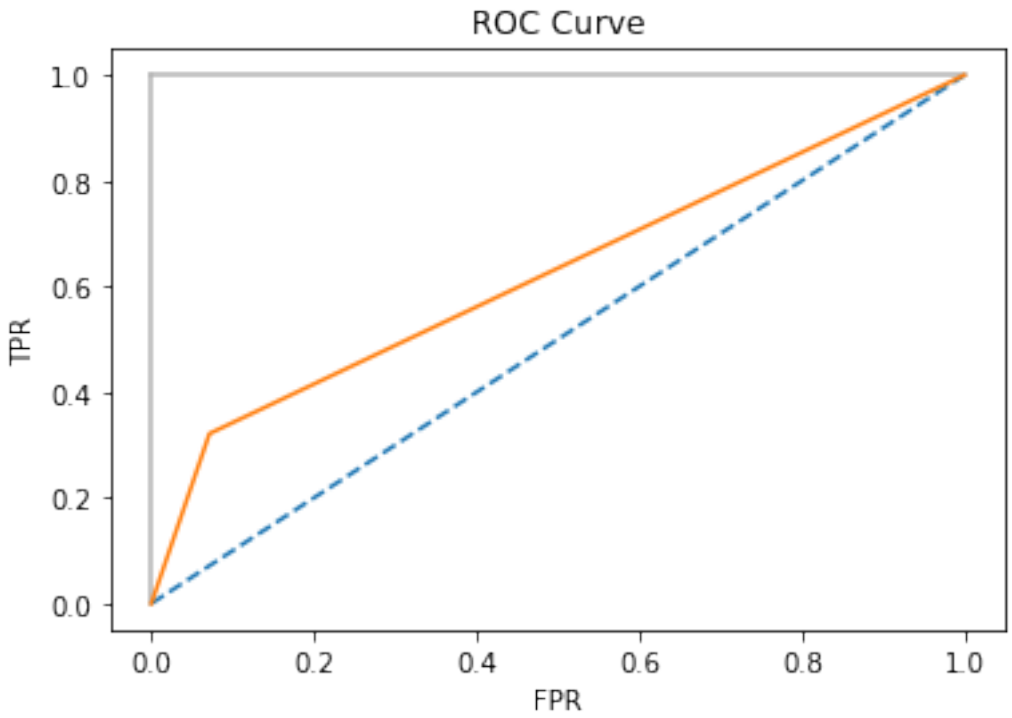
```
Classification Report
              precision    recall  f1-score   support

           0       0.73      0.93      0.82        56
           1       0.69      0.32      0.44        28

    accuracy                           0.73        84
   macro avg       0.71      0.62      0.63        84
weighted avg       0.72      0.73      0.69        84
```

[35]:
```
#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);
```



[36]:
```
# Plot ROC curve
fpr, tpr, thresholds = roc_curve(test_y,y_pred_bestTree)
plt.clf()
```

```
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.plot(fpr,tpr)
plt.show()
```



[37]:
```
#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)
```

[37]:
```
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)
```

[38]:
```
y_pred_Adaboost = BC_tree_Adaboost.predict(test_X)
y_pred_Adaboost
```

[38]:
```
array([0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 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)
```

[39]:
```
print("AUC:\n",metrics.roc_auc_score(test_y,y_pred_Adaboost))
```

```
AUC:
 0.5714285714285714
```

[40]:
```
print("Confusion Matrix\n", (confusion_matrix(test_y,y_pred_Adaboost)))
```

```
Confusion Matrix
 [[46 10]
 [19  9]]
```

[41]:
```
print("Accuracy:\n",(accuracy_score(test_y,y_pred_Adaboost)*100))
```

```
Accuracy:
 65.47619047619048
```

[42]:
```
print("Classification Report\n", (classification_report(test_y,␣
 ↪y_pred_Adaboost)))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.71      0.82      0.76        56
           1       0.47      0.32      0.38        28

    accuracy                           0.65        84
   macro avg       0.59      0.57      0.57        84
weighted avg       0.63      0.65      0.63        84
```

[43]:
```
#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)
```

[43]:
```
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)
```

[44]:
```
y_pred_gradient = BC_tree_gradient.predict(test_X)
```

[45]:
```
print("Confusion Matrix\n", (confusion_matrix(test_y,y_pred_gradient)))
```

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

[46]:
```
print("Accuracy:\n",(accuracy_score(test_y,y_pred_gradient)*100))
```

```
Accuracy:
 63.095238095238095
```

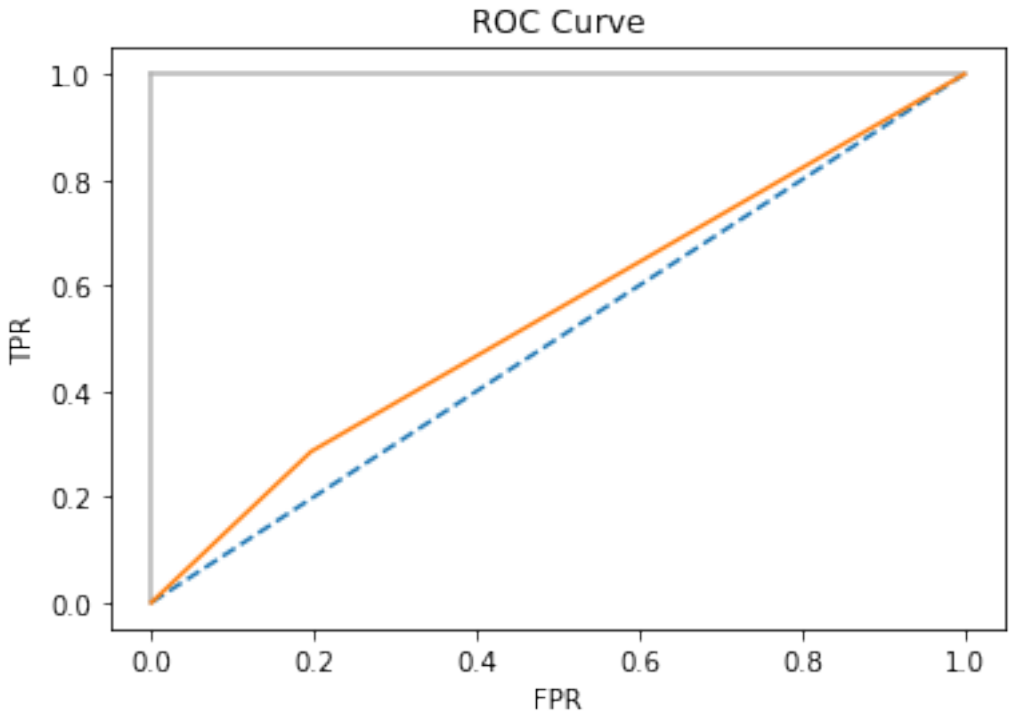
[47]:
```
print("Classification Report\n", (classification_report(test_y,␣
↪y_pred_gradient)))
```

```
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
```

[48]:
```
print("AUC:\n",metrics.roc_auc_score(test_y,y_pred_gradient))
```

```
AUC:
 0.5446428571428571
```

[49]:
```
# Plot ROC curve - Gradient Boosting
fpr, tpr, thresholds = roc_curve(test_y,y_pred_gradient)
plt.clf()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.plot(fpr,tpr)
plt.show()
```



[ ]: