Homework 5 - Tree-Based Inference

March 1, 2020

Student: Dimitrios Tanoglidis

[1]:
```python
#Import stuff
import numpy as np
import pandas as pd
import itertools
from sklearn import linear_model
from sklearn.metrics import mean_squared_error as MSE
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import rcParams
#import seaborn as sns
rcParams['font.family'] = 'serif'

# Adjust rc parameters to make plots pretty
def plot_pretty(dpi=200, fontsize=8):

    import matplotlib.pyplot as plt

    plt.rc("savefig", dpi=dpi)
    #plt.rc('text', usetex=True)
    plt.rc('font', size=fontsize)
    plt.rc('xtick', direction='in')
    plt.rc('ytick', direction='in')
    plt.rc('xtick.major', pad=10)
    plt.rc('xtick.minor', pad=5)
    plt.rc('ytick.major', pad=10)
    plt.rc('ytick.minor', pad=5)
    plt.rc('lines', dotted_pattern = [0.5, 1.1])

    return
plot_pretty()
```

0.1 Conceptual: Cost functions for classification trees

It is good to make a plot of the three quantities as a function of the purity of the first class, p_1. For two classes, $p_2 = 1 - p_1$.

Then, Error rate:

$$E = 1 - max(p, 1 - p) \tag{1}$$

Gini index is:

$$G = p(1 - p) + (1 - p)p = 2p(1 - p) \tag{2}$$

And Entropy:

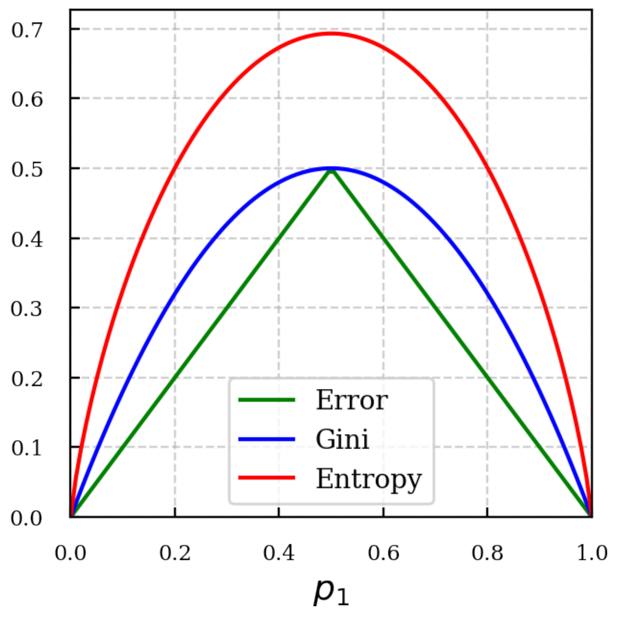
$$D = -p\log(p) - (1 - p)\log(1 - p) \tag{3}$$

[165]:
```
p = np.linspace(1e-9,0.9999999,100)
Error = np.zeros(100)
for i in range(100):
    Error[i] = 1.0 - max(p[i],1.0-p[i])

Gini = 2.0*p*(1.0-p)

Entropy = - p*np.log(p) - (1.-p)*np.log(1.0-p)

plt.figure(figsize=(3.5,3.5))
plt.plot(p,Error, c='green', label='Error')
plt.plot(p,Gini, c='blue', label='Gini')
plt.plot(p,Entropy, c='red', label='Entropy')
plt.grid(ls='--', alpha=0.6)
plt.xlim(0,1);plt.ylim(0,)
plt.legend(loc='lower center', fontsize=10)
plt.xlabel('$p_1$', fontsize=13)
plt.show()
```



When building a classification tree we can use the Gini index or the entropy when spliting each node, since they are more sensitive to node purity (slope of curves at $p_1 \sim 0$ or $p_1 \sim 1$).

When pruning the tree the classification error rate would be better if we want to predict the accuracy of the pruned tree, since error rate is actually 1- accuracy.

0.2 Application: Predicting attitudes towards racist college proffessors

Let's read the data first

[5]:
```
df_train = pd.read_csv('gss_train.csv') #Train data
df_test = pd.read_csv('gss_test.csv') #Test data

#df_train.head()
```

Get the labels, corresponding to the variable `colrac` and the fatures, corresponding to all the other columns of the dataframes.

[6]:
```
# Get the train/test target values
y_train = df_train['colrac'].values
y_test = df_test['colrac'].values
# Get the train/test features
X_train = df_train.loc[:, df_train.columns != 'colrac'].values
X_test = df_test.loc[:, df_test.columns != 'colrac'].values
```

0.2.1 Estimate the Models

Estimate the following models, predicting `colrac` using the training set (the `training .csv`) with 10-fold CV:

Logistic Regression This is a very simple model, without parameters to tune actually.

[61]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

LR_class = LogisticRegression(solver='newton-cg')# Define the classifier

LR_class.fit(X_train,y_train) # Fit
colrac_pr = LR_class.predict(X_train) # Predict
```

Naive Bayes Similarly, for this model we don't have parameters to tune.

[69]:
```
from sklearn.naive_bayes import GaussianNB

GN_class = GaussianNB()
```

Elastic Net Of course this is a classification problem, so I will use a stochastic gradient descent with Elastic Net regularization.

For this classifier, I will tune the mixing (hyper)parameter `l1_ratio` that takes values in the range $[0, 1]$.

[62]:
```
from sklearn.linear_model import SGDClassifier

Elastic_params = {'l1_ratio': [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]}
Elastic_clf = SGDClassifier(penalty='elasticnet')

Elastic_grid = GridSearchCV(Elastic_clf, Elastic_params,cv=10,␣
 ↪scoring='accuracy')
Elastic_grid.fit(X_train, y_train)
print(Elastic_grid.best_params_)
```

```
{'l1_ratio': 1.0}
```

Decision Tree For this classifier I will tune the maximum depth of the tree (pruning).

[31]:
```
from sklearn.tree import DecisionTreeClassifier

DT_param_grid = {'max_depth':[1,2,4,6,8,10,12,15]} # Here I will tune the␣
 ↪maximum depth of the tree
```

```
# There are some other hyperparameters, but this is the most important

DT_class = DecisionTreeClassifier()
DT_grid = GridSearchCV(DT_class, DT_param_grid,cv=10, scoring='accuracy')
DT_grid.fit(X_train, y_train)
print(DT_grid.best_params_)
```

```
{'max_depth': 4}
```

Bagging For this estimator I will tune two hyperparameters:

The number of base estimators (Decision Trees) to use (this is equal to B using the notiation we used in class) and the number of samples, given as a fraction of the total number of instances used for the classification.

[40]:
```
from sklearn.ensemble import BaggingClassifier

bag_param_grid = {'n_estimators': [10,20,50,80,100],
                 'max_samples': [0.5,0.6,0.7,0.8]}

bag_clf = BaggingClassifier(base_estimator=DecisionTreeClassifier())
bag_grid = GridSearchCV(bag_clf, bag_param_grid,cv=10, scoring='accuracy')
bag_grid.fit(X_train, y_train)
print(bag_grid.best_params_)
```

```
{'n_estimators': 80, 'max_samples': 0.6}
```

Random Forest Here I will tune the following hyperparameters:

- Number of trees: `n_estimators`.
- Number of features at each split: `max_features` (given as a fraction of the total num of features).
- Min observation in terminal nodes: `min_samples_leaf`.

There are other hyperparameters that can be trained as well, but this would be time consuming, so I will keep just these three.

[45]:
```
from sklearn.ensemble import RandomForestClassifier

RF_param_grid = {'n_estimators': [50,100,150,200],
                'max_features': [0.3,0.4,0.5,0.6,0.7],
                'min_samples_leaf': [2,5,10,20,50]}

RF_clf = RandomForestClassifier()
RF_grid = GridSearchCV(RF_clf, RF_param_grid,cv=10, scoring='accuracy')
RF_grid.fit(X_train, y_train)
print(RF_grid.best_params_)
```

```
{'max_features': 0.3, 'n_estimators': 150, 'min_samples_leaf': 5}
```

Boosting For this model I will tune the following hyperparameters:

- Number of trees, B, given by the sklearn parameter: `n_estimators`.
- Learning rate, λ, given by the parameter: `learning_rate`.
- Number of splits at each tree, d, given by the parameter: `min_samples_leaf`.

[50]:
```
from sklearn.ensemble import GradientBoostingClassifier as GBCl

GB_param_grid = {'n_estimators': [50,100,150,200],
                'learning_rate': [0.01, 0.05, 0.1, 0.2, 0.3],
                'min_samples_leaf': [1,2,5,10]}

GB_clf = GBCl()
GB_grid = GridSearchCV(GB_clf, GB_param_grid,cv=10, scoring='accuracy')
GB_grid.fit(X_train, y_train)
print(GB_grid.best_params_)
```

```
{'n_estimators': 100, 'learning_rate': 0.1, 'min_samples_leaf': 5}
```

0.2.2 Evaluate the models

Now I will compare and present each model's training performance based on:

- Cross-validated error rate
- ROC/AUC

[84]:
```
from sklearn.metrics import zero_one_loss as Error_Rate
from sklearn.model_selection import cross_val_score as CV_score
from sklearn.metrics import roc_curve, auc
```

Logistic Regression

[94]:
```
# Define the classifier
LR_clf = LogisticRegression(solver='newton-cg')
# Get cross-validation scores
scores_LR = CV_score(LR_clf, X_train, y_train, cv=10, scoring='accuracy')
# Cross-Validation error rate
CV_err_LR = 1.0 - np.mean(scores_LR)
print('Cross validation error:',CV_err_LR)
```

```
('Cross validation error:', 0.20255765284528482)
```

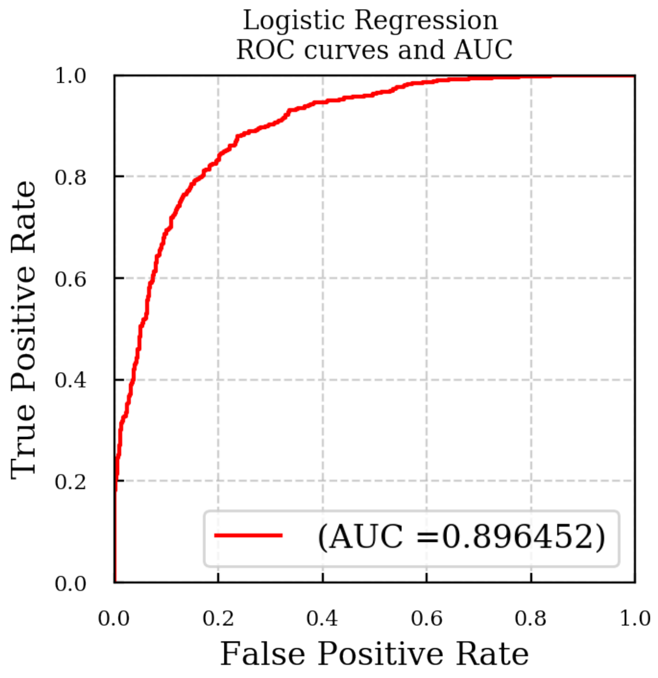
[96]:
```
# Predict probabilities
LR_clf.fit(X_train,y_train)
y_proba_LR = LR_clf.predict_proba(X_train)[:,1] #Logistic Regression
```

```
# fpr, tpr for the model in order to plot the ROC curve
fpr_LR, tpr_LR, thr = roc_curve(y_train, y_proba_LR)
# ======================================================
# Calcuate the AUC
AUC_LR = auc(fpr_LR,tpr_LR) # Logistic Regression
```

[105]:
```
plt.figure(figsize=(3.5,3.5))

plt.plot(fpr_LR, tpr_LR, c='red', label=' (AUC ={0:3f})'.format(AUC_LR))
plt.title('Logistic Regression \nROC curves and AUC')
plt.grid(ls='--', alpha=0.6)
plt.xlabel('False Positive Rate',fontsize=12);plt.ylabel('True Positive␣
 ↪Rate',fontsize=12)
plt.xlim(0.0,1.0);plt.ylim(0.0,1.0)

plt.legend(loc='lower right', fontsize=12)
plt.show()
```



Naive Bayes

[86]:
```
# Define the classifier
NB_clf = GaussianNB()
# Get cross-validation scores
scores_NB = CV_score(NB_clf, X_train, y_train, cv=10, scoring='accuracy')
```

```
# Cross-Validation error rate
CV_err_NB = 1.0 - np.mean(scores_NB)
print('Cross validation error:',CV_err_NB)
```

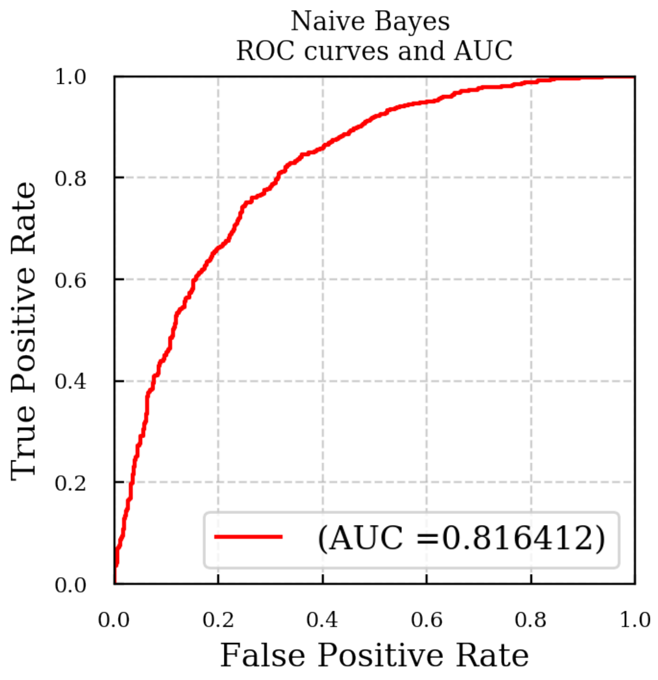
```
('Cross validation error:', 0.26555250977590383)
```

[108]:
```
# Predict probabilities
NB_clf.fit(X_train,y_train)
y_proba_NB = NB_clf.predict_proba(X_train)[:,1] #Naive Bayes
# fpr, tpr for the model in order to plot the ROC curve
fpr_NB, tpr_NB, thr = roc_curve(y_train, y_proba_NB)
# =====================================================
# Calcuate the AUC
AUC_NB = auc(fpr_NB,tpr_NB) # Naive Bayes
```

[109]:
```
plt.figure(figsize=(3.5,3.5))

plt.plot(fpr_NB, tpr_NB, c='red', label=' (AUC ={0:3f})'.format(AUC_NB))
plt.title('Naive Bayes \nROC curves and AUC')
plt.grid(ls='--', alpha=0.6)
plt.xlabel('False Positive Rate',fontsize=12);plt.ylabel('True Positive␣
 ↪Rate',fontsize=12)
plt.xlim(0.0,1.0);plt.ylim(0.0,1.0)

plt.legend(loc='lower right', fontsize=12)
plt.show()
```



Elastic Net

```
[87]: # Define the classifier
      Elastic_clf = SGDClassifier(penalty='elasticnet', l1_ratio=1.0)
      # Get cross-validation scores
      scores_Elastic = CV_score(Elastic_clf, X_train, y_train, cv=10,␣
       ↪scoring='accuracy')
      # Cross-Validation error rate
      CV_err_Elastic = 1.0 - np.mean(scores_Elastic)
      print('Cross validation error:', CV_err_Elastic)
```

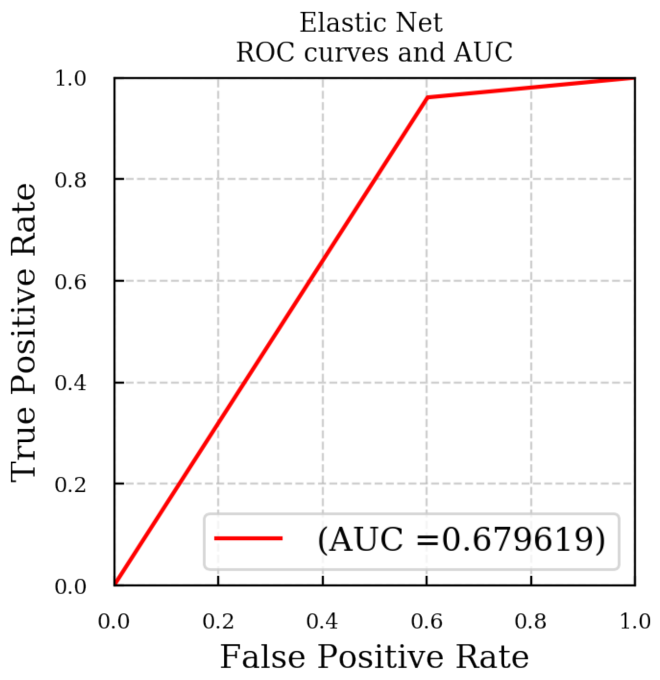
```
('Cross validation error:', 0.279093786979693)
```

```
[113]: # Predict probabilities
       Elastic_clf = SGDClassifier(loss='modified_huber',penalty='elasticnet',␣
        ↪l1_ratio=1.0)
       Elastic_clf.fit(X_train,y_train)
       y_proba_Elastic = Elastic_clf.predict_proba(X_train)[:,1] #Elastic Net
       # fpr, tpr for the model in order to plot the ROC curve
       fpr_Elastic, tpr_Elastic, thr = roc_curve(y_train, y_proba_Elastic)
       # =====================================================
       # Calcuate the AUC
       AUC_Elastic = auc(fpr_Elastic,tpr_Elastic) # Elastic Net
```

```
[114]: plt.figure(figsize=(3.5,3.5))

       plt.plot(fpr_Elastic, tpr_Elastic, c='red', label=' (AUC ={0:3f})'.
        ↪format(AUC_Elastic))
       plt.title('Elastic Net \nROC curves and AUC')
       plt.grid(ls='--', alpha=0.6)
       plt.xlabel('False Positive Rate',fontsize=12);plt.ylabel('True Positive␣
        ↪Rate',fontsize=12)
       plt.xlim(0.0,1.0);plt.ylim(0.0,1.0)

       plt.legend(loc='lower right', fontsize=12)
       plt.show()
```



Decision Tree

[88]:
```
# Define the classifier
DT_class = DecisionTreeClassifier(max_depth=4)
# Get cross-validation scores
scores_DT = CV_score(DT_class, X_train, y_train, cv=10, scoring='accuracy')
# Cross-Validation error rate
CV_err_DT = 1.0 - np.mean(scores_DT)
print('Cross validation error:',CV_err_DT)
```

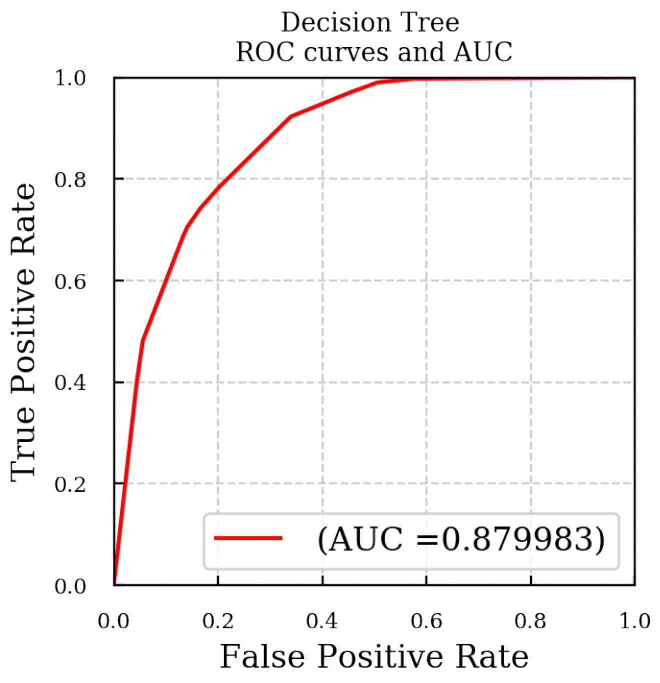
```
('Cross validation error:', 0.21618607718799476)
```

[118]:
```
DT_class.fit(X_train,y_train)
y_proba_DT = DT_class.predict_proba(X_train)[:,1] #Decision Tree
# fpr, tpr for the model in order to plot the ROC curve
fpr_DT, tpr_DT, thr = roc_curve(y_train, y_proba_DT)
# =====================================================
# Calcuate the AUC
AUC_DT = auc(fpr_DT,tpr_DT) # Decsion Tree
```

[117]:
```
plt.figure(figsize=(3.5,3.5))

plt.plot(fpr_DT, tpr_DT, c='red', label=' (AUC ={0:3f})'.format(AUC_DT))
plt.title('Decision Tree \nROC curves and AUC')
plt.grid(ls='--', alpha=0.6)
plt.xlabel('False Positive Rate',fontsize=12);plt.ylabel('True Positive␣
↪Rate',fontsize=12)
plt.xlim(0.0,1.0);plt.ylim(0.0,1.0)

plt.legend(loc='lower right', fontsize=12)
plt.show()
```



Bagging

[89]:
```
# Define the classifier
bag_clf =␣
↪BaggingClassifier(base_estimator=DecisionTreeClassifier(),n_estimators=80,max_samples=0.
↪6)
# Get cross-validation scores
scores_bag = CV_score(bag_clf, X_train, y_train, cv=10, scoring='accuracy')
# Cross-Validation error rate
CV_err_bag = 1.0 - np.mean(scores_bag)
print('Cross validation error:',CV_err_bag)
```

```
('Cross validation error:', 0.20530625818257653)
```

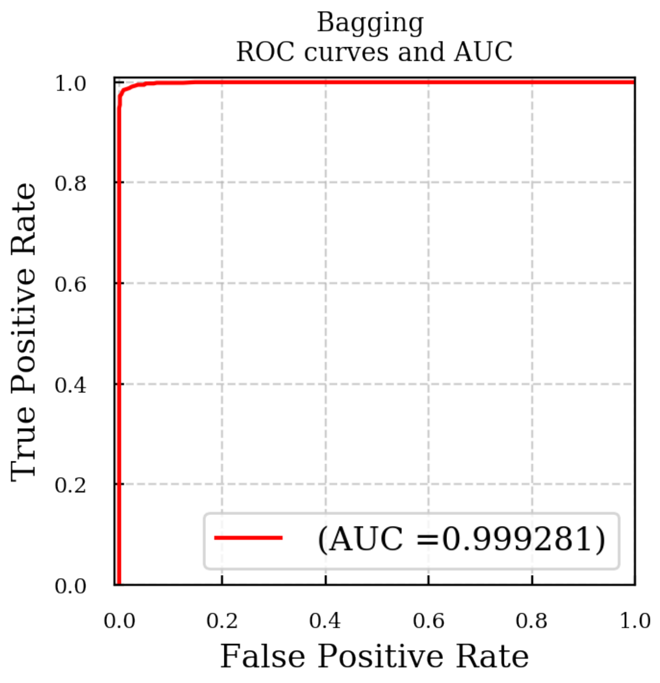
[119]:
```
bag_clf.fit(X_train,y_train)
y_proba_bag = bag_clf.predict_proba(X_train)[:,1] #Bagging
# fpr, tpr for the model in order to plot the ROC curve
```

```
fpr_bag, tpr_bag, thr = roc_curve(y_train, y_proba_bag)
# =====================================================
# Calcuate the AUC
AUC_bag = auc(fpr_bag,tpr_bag) # Bagging
```

[123]:
```
plt.figure(figsize=(3.5,3.5))

plt.plot(fpr_bag, tpr_bag, c='red', label=' (AUC ={0:3f})'.format(AUC_bag))
plt.title('Bagging \nROC curves and AUC')
plt.grid(ls='--', alpha=0.6)
plt.xlabel('False Positive Rate',fontsize=12);plt.ylabel('True Positive␣
 ↪Rate',fontsize=12)
plt.xlim(-0.01,1.0);plt.ylim(0.0,1.01)

plt.legend(loc='lower right', fontsize=12)
plt.show()
```



Random Forest

[90]:
```
# Define the classifier
RF_clf = RandomForestClassifier(n_estimators=150,max_features=0.
 ↪3,min_samples_leaf=5)
# Get cross-validation scores
scores_RF = CV_score(RF_clf, X_train, y_train, cv=10, scoring='accuracy')
```

```
# Cross-Validation error rate
CV_err_RF = 1.0 - np.mean(scores_RF)
print('Cross validation error:',CV_err_RF)
```

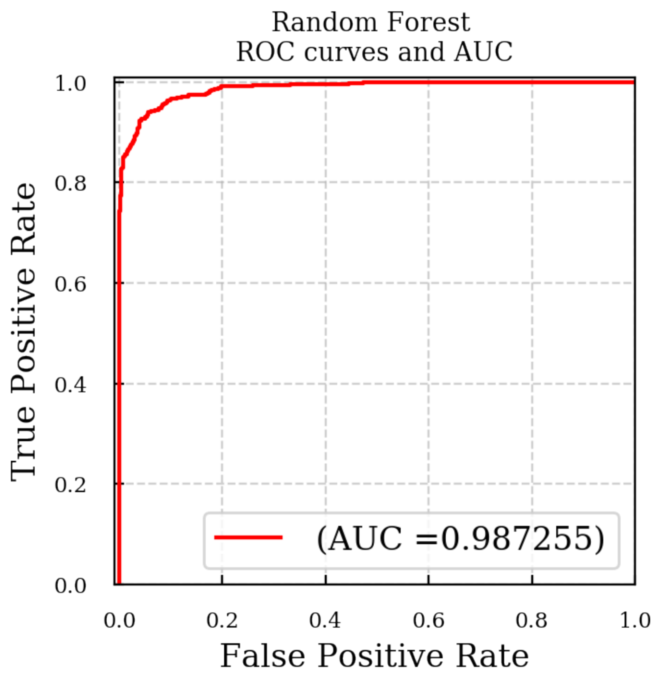
```
('Cross validation error:', 0.20530166174940856)
```

[125]:
```
RF_clf.fit(X_train,y_train)
y_proba_RF = RF_clf.predict_proba(X_train)[:,1] #Random Forest
# fpr, tpr for the model in order to plot the ROC curve
fpr_RF, tpr_RF, thr = roc_curve(y_train, y_proba_RF)
# ========================================================
# Calcuate the AUC
AUC_RF = auc(fpr_RF,tpr_RF) # Random Forest
```

[126]:
```
plt.figure(figsize=(3.5,3.5))

plt.plot(fpr_RF, tpr_RF, c='red', label=' (AUC ={0:3f})'.format(AUC_RF))
plt.title('Random Forest \nROC curves and AUC')
plt.grid(ls='--', alpha=0.6)
plt.xlabel('False Positive Rate',fontsize=12);plt.ylabel('True Positive␣
 ↪Rate',fontsize=12)
plt.xlim(-0.01,1.0);plt.ylim(0.0,1.01)

plt.legend(loc='lower right', fontsize=12)
plt.show()
```



Boosting

[91]:
```
# Define the classifier
GB_clf = GBCl(n_estimators=100,learning_rate=0.1,min_samples_leaf=5)
# Get cross-validation scores
scores_GB = CV_score(GB_clf, X_train, y_train, cv=10, scoring='accuracy')
# Cross-Validation error rate
CV_err_GB = 1.0 - np.mean(scores_GB)
print('Cross validation error:',CV_err_GB)
```

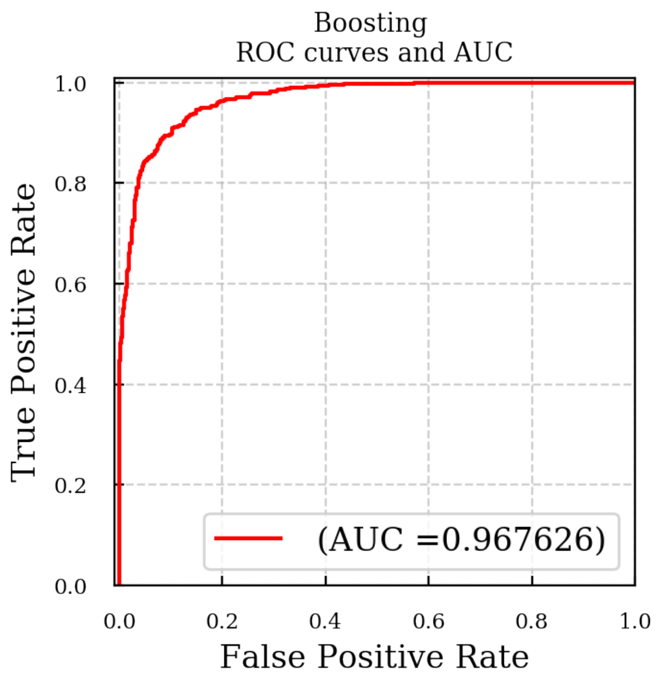
```
('Cross validation error:', 0.19447234181174744)
```

[127]:
```
GB_clf.fit(X_train,y_train)
y_proba_GB = GB_clf.predict_proba(X_train)[:,1] #Gradient Boosting
# fpr, tpr for the model in order to plot the ROC curve
fpr_GB, tpr_GB, thr = roc_curve(y_train, y_proba_GB)
# =====================================================
# Calcuate the AUC
AUC_GB = auc(fpr_GB,tpr_GB) # Gradient Boosting
```

[128]:
```
plt.figure(figsize=(3.5,3.5))

plt.plot(fpr_GB, tpr_GB, c='red', label=' (AUC ={0:3f})'.format(AUC_GB))
plt.title('Boosting \nROC curves and AUC')
plt.grid(ls='--', alpha=0.6)
plt.xlabel('False Positive Rate',fontsize=12);plt.ylabel('True Positive␣
 ↪Rate',fontsize=12)
plt.xlim(-0.01,1.0);plt.ylim(0.0,1.01)

plt.legend(loc='lower right', fontsize=12)
plt.show()
```



Best Model We see that the maximum AUC belongs to the Bagging model (AUC = 0.99928) followed by that of the Random Forest classifier (AUC = 0.98725). The Gradient Boosting model has an AUC = 0.96762.

In terms of the (cross validated) error rate, the minimum Error Rate belongs to the Boosting model:

- Error Rate (Boosting): 0.1944

Followed by that of the Random Forest Classifier:

- Error Rate (RF) : 0.205301

And of the Bagging:

- Error Rate (Bagging): 0.205306

Given that Boosting and Bagging models have comparable Error rates (difference ~ 0.01) but Bagging has significantly higher AUC (difference ~ 0.03), I will use the Bagging model as the best one.

0.2.3 Evaluate the *best* model

Here I will evaluate the Bagging model (the best model, according to the analysis above) and test that on the test set (classification error rate and AUC).

[134]:
```
# Predict on the test set
y_pred_bag = bag_clf.predict(X_test)

# Calculate Error Rate
ER_test = Error_Rate(y_pred_bag,y_test)
print(ER_test)
```

```
0.20486815415821502
```

The Error rate on the test set is ~ 0.204, and is actually less than the Cross Validation Error Rate.

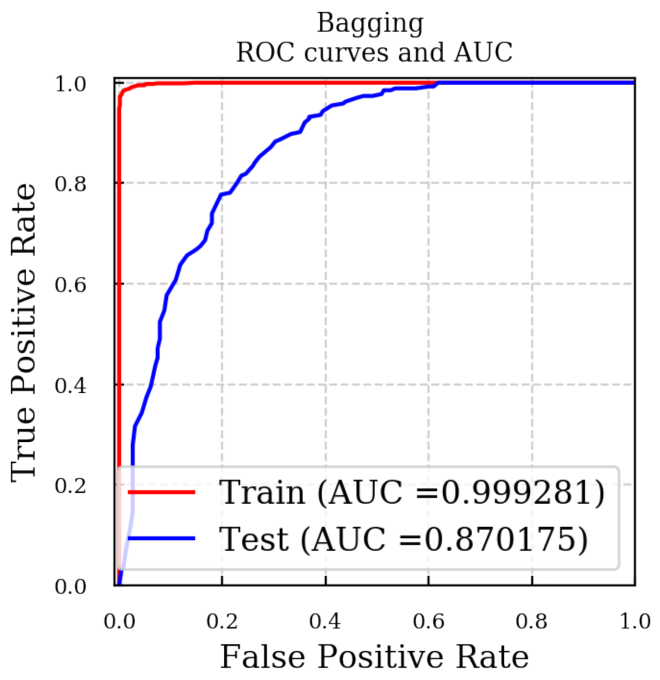
Let's compare the ROC curves and the AUCs.

[136]:
```
y_test_proba_bag = bag_clf.predict_proba(X_test)[:,1] #Bagging
# fpr, tpr for the model in order to plot the ROC curve
fpr_test_bag, tpr_test_bag, thr = roc_curve(y_test, y_test_proba_bag)
# =====================================================
# Calcuate the AUC
AUC_test_bag = auc(fpr_test_bag,tpr_test_bag) # Bagging
```

[138]:
```
plt.figure(figsize=(3.5,3.5))

plt.plot(fpr_bag, tpr_bag, c='red', label='Train (AUC ={0:3f})'.format(AUC_bag))
plt.plot(fpr_test_bag, tpr_test_bag, c='blue', label='Test (AUC ={0:3f})'.
 ↪format(AUC_test_bag))
plt.title('Bagging \nROC curves and AUC')
plt.grid(ls='--', alpha=0.6)
plt.xlabel('False Positive Rate',fontsize=12);plt.ylabel('True Positive␣
 ↪Rate',fontsize=12)
plt.xlim(-0.01,1.0);plt.ylim(0.0,1.01)

plt.legend(loc='lower right', fontsize=12)
plt.show()
```



The test AUC is much lower than that of the train set (0.87 vs 0.99), thus the "Best" model does not generalize well.

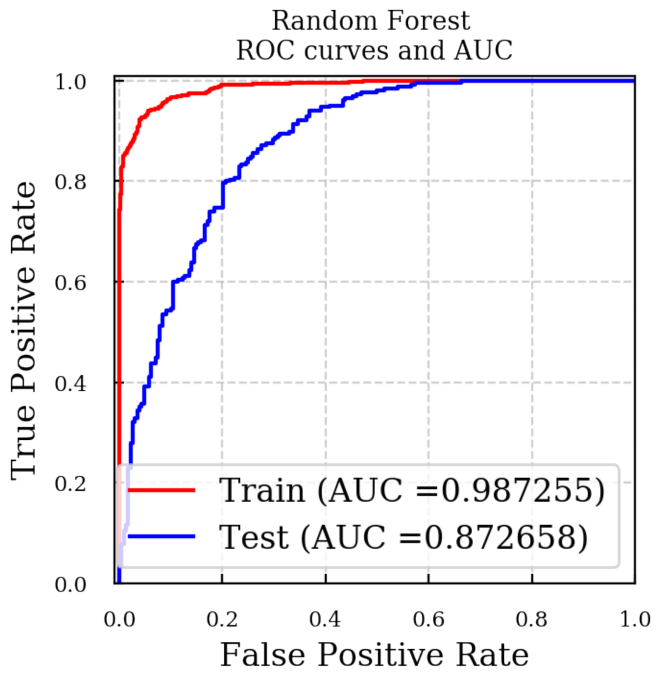
Let's check the Random Forest Classifier to see if it generalizes better.

[139]:
```
y_test_proba_RF = RF_clf.predict_proba(X_test)[:,1] #Random Forest
# fpr, tpr for the model in order to plot the ROC curve
fpr_test_RF, tpr_test_RF, thr = roc_curve(y_test, y_test_proba_RF)
# =====================================================
# Calcuate the AUC
AUC_test_RF = auc(fpr_test_RF,tpr_test_RF) # Random Forest
```

[142]:
```
plt.figure(figsize=(3.5,3.5))

plt.plot(fpr_RF, tpr_RF, c='red', label='Train (AUC ={0:3f})'.format(AUC_RF))
plt.plot(fpr_test_RF, tpr_test_RF, c='blue', label='Test (AUC ={0:3f})'.
 ↪format(AUC_test_RF))
plt.title('Random Forest \nROC curves and AUC')
plt.grid(ls='--', alpha=0.6)
plt.xlabel('False Positive Rate',fontsize=12);plt.ylabel('True Positive␣
 ↪Rate',fontsize=12)
plt.xlim(-0.01,1.0);plt.ylim(0.0,1.01)

plt.legend(loc='lower right', fontsize=12)
plt.show()
```



This model generalizes better (Train and Test AUCs agree better) however it is still quite bad (the two AUCs do not agree very well).