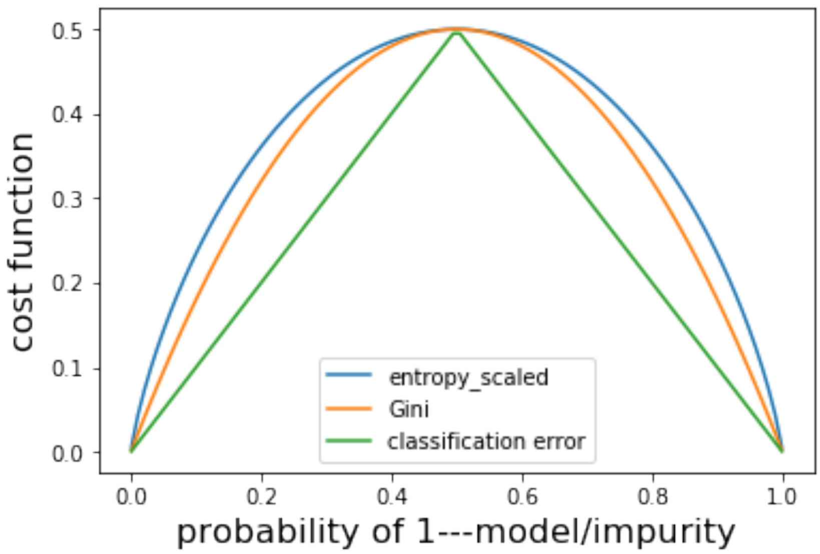
Question 1

```
p = np.linspace(0.001,0.999,100)
entropy_scaled = 0.5*(- p*np.log2(p) - (1 - p)*np.log2((1 - p)))
gini = (p)*(1 - (p)) + (1 - p)*(1 - (1-p))
classification_error = 1 - np.max([p, 1 - p], axis = 0)
plt.plot(p,entropy_scaled,label = 'entropy_scaled')
plt.plot(p,gini, label = 'Gini')
plt.plot(p,classification_error, label = 'classification error')
plt.legend()
plt.xlabel('probability of 1---model/impurity',size = 16)
plt.ylabel('cost function',size = 16)
plt.show()
```



When we are training a decision tree, we use the cost function to decide

1. whether to use this feature to split the node;
2. which value to use to split it
3. whether to stop splitting.

During the process of training, we need to decide whether to stop training, and we often would set a threshold of cost function for decision. I think Gini and Cross are more suitable as criterion than classification error. Gini and entropy are more sensitive to the impurity of data, so that a given threshold will naturally mean a lower impurity of data after classifying by the model which uses the Gini or entropy as cost function compared with that uses classification error.

When pruning the tree, we would cut the tree so that the new cost function(C_new) of the model will become: cost function(C) + alpha * the complexity of model(|T|). For a given alpha, we would search around the subtree to decide an optimal tree that can minimize the C_new. Using the criterion of classification error can help us to search around the model space more efficiently as it naturally represent the impurity of data and it can adjust the impurity larger for a given decrease of cost function when the impurity is low(after the training).

--- Have discussed it with Shengwenxin Ni whose suggestions are really helpful.
---- https://www.bogotobogo.com/python/scikit-learn/scikt_machine_learning_Decision_Tree_Learning_Informatioin_Gain_IG_Impurity_Entropy_Gini_Classification_Error.php

Question 2

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

from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV

from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier

from sklearn.metrics import auc
from sklearn.metrics import roc_curve
import copy
```

```
from sklearn import preprocessing
```

```
df_train = pd.read_csv('gss_train.csv')
df_test = pd.read_csv('gss_test.csv')
data_train = copy.copy(df_train)
data_test = copy.copy(df_test)
Y_train = df_train['colrac']
Y_test = df_test['colrac']
data_train.drop(['colrac'],axis = 1, inplace=True)
data_test.drop(['colrac'],axis = 1, inplace=True)
X_train = np.array(data_train)
X_scaled_train = preprocessing.scale(X_train)
X_test = np.array(data_test)
X_scaled_test = preprocessing.scale(X_test)
```

```
df_train.shape
```

```
(1476, 56)
```

```
?LogisticRegression
```

```
indice = np.random.permutation(np.array(list(range(len(X_train)))))
# logistic regression; without hyperparameter
kf = KFold(n_splits=10)
error_rate = []
for train_indice, test_indice in kf.split(indice):
    x_train = X_scaled_train[train_indice]
    y_train = Y_train[train_indice]
    x_test = X_scaled_train[test_indice]
    y_test = Y_train[test_indice]
    clf = LogisticRegression(random_state=0).fit(x_train, y_train)
    error_rate.append(clf.score(x_test, y_test))
```

```
np.mean(error_rate)
```

```
0.79470950542379115
```

```
# naive bayesian model
kf = KFold(n_splits=10)
error_rate = []
for train_indice, test_indice in kf.split(indice):
    x_train = X_scaled_train[train_indice]
    y_train = Y_train[train_indice]
    x_test = X_scaled_train[test_indice]
    y_test = Y_train[test_indice]
    clf = GaussianNB().fit(x_train, y_train)
    error_rate.append(clf.score(x_test, y_test))
```

```
np.mean(error_rate)
```

```
0.7350983636697922
```

```
#elastic net regression with grid search
elst = ElasticNet()
parameters = {'alpha':[0.1,0.5,1,10,100], 'l1_ratio':np.linspace(0.1,1,11)}
clf = GridSearchCV(elst, parameters, cv = 10)
clf.fit(X_scaled_train,Y_train)
```

```
GridSearchCV(cv=10, error_score=nan,
             estimator=ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True,
                                  l1_ratio=0.5, max_iter=1000, normalize=False,
                                  positive=False, precompute=False,
                                  random_state=None, selection='cyclic',
                                  tol=0.0001, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [0.1, 0.5, 1, 10, 100],
                         'l1_ratio': array([ 0.1 ,  0.19,  0.28,  0.37,  0.46,
0.55,  0.64,  0.73,  0.82,
        0.91,  1.  ])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
clf.best_estimator_
```

```
ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True,
           l1_ratio=0.10000000000000001, max_iter=1000, normalize=False,
           positive=False, precompute=False, random_state=None,
           selection='cyclic', tol=0.0001, warm_start=False)
```

```
elst = ElasticNet(alpha=0.1,l1_ratio=0.1)
clf.fit(X_scaled_train,Y_train)
y_predict = clf.predict(x_test)
y_predict[y_predict>0.5] = 1
y_predict[y_predict<=0.5] = 0
error_rate = sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test)
```

```
error_rate
```

```
0.18367346938775511
```

```
# decision tree
Tree = DecisionTreeClassifier()
#n_components = list(range(1,X.shape[1]+1,1))
criterion = ['gini', 'entropy']
splitter = ["best", "random"]
max_depth = [4,6,8,12]
parameters = {'criterion':criterion, 'splitter':splitter,
'max_depth':max_depth}
clf = GridSearchCV(Tree, parameters, cv = 10)
clf.fit(X_scaled_train,Y_train)
```

```
GridSearchCV(cv=10, error_score=nan,
             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'),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [4, 6, 8, 12],
                         'splitter': ['best', 'random']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
clf.best_estimator_
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=4, 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='random')
```

```
clf.best_score_
```

```
0.78656462585034015
```

```
# Bagging
from sklearn.ensemble import BaggingClassifier
Bagging = BaggingClassifier()
n_estimators = [1,10,20,30,40]
max_samples = [4,6,8,12]
bootstrap = [True, False]
#bootstrap_features = [True, False]
#oob_score = [True,]
warm_start = [True, False]
parameters =
{'n_estimators':n_estimators,'max_samples':max_samples,'bootstrap':bootstrap,\
              'warm_start':warm_start}
clf = GridSearchCV(Bagging, parameters, cv = 10)
clf.fit(X_scaled_train,Y_train)
```

```
GridSearchCV(cv=10, error_score=nan,
             estimator=BaggingClassifier(base_estimator=None, bootstrap=True,
                                         bootstrap_features=False,
                                         max_features=1.0, max_samples=1.0,
                                         n_estimators=10, n_jobs=None,
                                         oob_score=False, random_state=None,
```

```
                                    verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'bootstrap': [True, False],
                         'max_samples': [4, 6, 8, 12],
                         'n_estimators': [1, 10, 20, 30, 40],
                         'warm_start': [True, False]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
clf.best_estimator_
```

```
BaggingClassifier(base_estimator=None, bootstrap=False,
                  bootstrap_features=False, max_features=1.0, max_samples=12,
                  n_estimators=40, n_jobs=None, oob_score=False,
                  random_state=None, verbose=0, warm_start=True)
```

```
clf.best_score_
```

```
0.770279463136606
```

```
#random forest
RF = RandomForestClassifier()
n_estimators = [10,20,50,100,150]
criterion = ['gini', 'entropy']
max_features = ["auto","sqrt","log2", X_scaled_train.shape[1]]
max_samples = [4,6,8,12]
bootstrap = [True, False]
#oob_score = [True, False]
#warm_start = [True, False]
parameters =
{'n_estimators':n_estimators,'max_samples':max_samples,'max_features':max_featu
res,\
              'bootstrap':bootstrap,'criterion':criterion}
clf = GridSearchCV(RF, parameters, cv = 10)
clf.fit(X_scaled_train,Y_train)
```

```
GridSearchCV(cv=10, 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={'bootstrap': [True, False],
                         'criterion': ['gini', 'entropy'],
                         'max_features': ['auto', 'sqrt', 'log2', 55],
                         'max_samples': [4, 6, 8, 12],
                         'n_estimators': [10, 20, 50, 100, 150]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
clf.best_estimator_
```

```
RandomForestClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=6,
                       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)
```

```
clf.best_score_
```

```
0.8082505975363117 9
```

```python
# Boosting

Boosting = AdaBoostClassifier()
n_estimators = [10,20,50,70,80]
learning_rate = np.linspace(0.1,1,10)
algorithm = ['SAMME', 'SAMME.R']
parameters = {'n_estimators':n_estimators, 'learning_rate':learning_rate,
'algorithm':algorithm}
clf = GridSearchCV(Boosting, parameters, cv = 10)
clf.fit(X_scaled_train,Y_train)
```

```
GridSearchCV(cv=10, error_score=nan,
             estimator=AdaBoostClassifier(algorithm='SAMME.R',
                                          base_estimator=None,
                                          learning_rate=1.0, n_estimators=50,
                                          random_state=None),
             iid='deprecated', n_jobs=None,
             param_grid={'algorithm': ['SAMME', 'SAMME.R'],
                         'learning_rate': array([ 0.1,  0.2,  0.3,  0.4,  0.5,
0.6,  0.7,  0.8,  0.9,  1. ]),
                         'n_estimators': [10, 20, 50, 70, 80]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```python
clf.best_estimator_
```

```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
                   learning_rate=0.40000000000000002, n_estimators=80,
                   random_state=None)
```

```python
clf.best_score_
```

```
0.8048722191579335
```

Question 3 & Question 4

```
import matplotlib.pyplot as plt
```

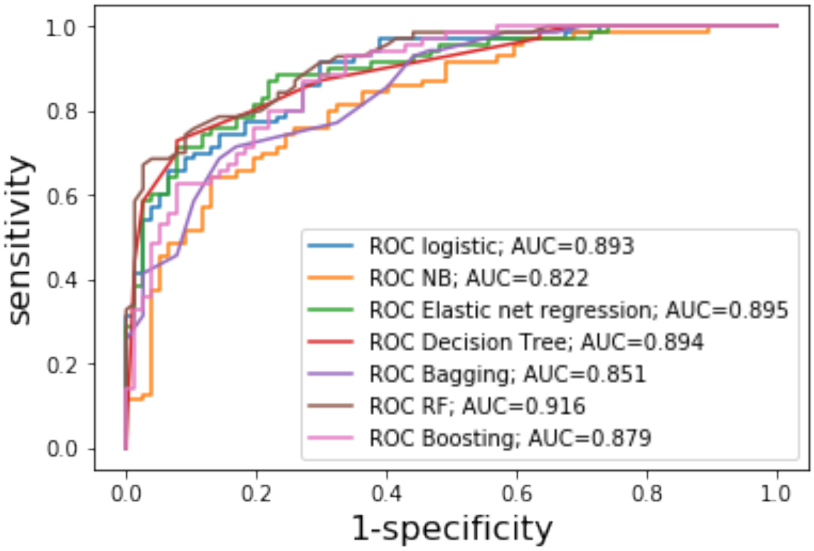
```
from sklearn.metrics import auc
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
```

```
model_error_rate = {}
# logistic error rate
kf = KFold(n_splits=10)
error_rate = []
for train_indice, test_indice in kf.split(indice):
    x_train = X_scaled_train[train_indice]
    y_train = Y_train[train_indice]
    x_test = X_scaled_train[test_indice]
    y_test = Y_train[test_indice]
    clf = LogisticRegression(random_state=0).fit(x_train, y_train)
    y_predict = clf.predict(x_test)
    error_rate.append(sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test))
y_predict_prob = clf.predict_proba(x_test)[:,1]
fpr, tpr,_ = roc_curve(y_test, y_predict_prob)
auc = roc_auc_score(y_test,  y_predict_prob)
plt.plot(fpr,tpr,label = 'ROC logistic; AUC={}'.format(round(auc,3)))
model_error_rate['logistic'] = np.mean(error_rate)

# Naive Bayes
kf = KFold(n_splits=10)
error_rate = []
for train_indice, test_indice in kf.split(indice):
    x_train = X_scaled_train[train_indice]
    y_train = Y_train[train_indice]
    x_test = X_scaled_train[test_indice]
    y_test = Y_train[test_indice]
    clf = GaussianNB().fit(x_train, y_train)
    y_predict = clf.predict(x_test)
    error_rate.append(sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test))
y_predict_prob = clf.predict_proba(x_test)[:,1]
fpr, tpr,_ = roc_curve(y_test, y_predict_prob)
auc = roc_auc_score(y_test,  y_predict_prob)
plt.plot(fpr,tpr,label = 'ROC NB; AUC={}'.format(round(auc,3)))
model_error_rate['Naive Bayes'] = np.mean(error_rate)

#Elastic net regression
kf = KFold(n_splits=10)
```

```
error_rate = []
for train_indice, test_indice in kf.split(indice):
    x_train = X_scaled_train[train_indice]
    y_train = Y_train[train_indice]
    x_test = X_scaled_train[test_indice]
    y_test = Y_train[test_indice]
    clf = ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True,
           l1_ratio=0.10000000000000001, max_iter=1000, normalize=False,
           positive=False, precompute=False, random_state=None,
           selection='cyclic', tol=0.0001, warm_start=False).fit(x_train,
y_train)
    y_predict = clf.predict(x_test)
    y_predict[y_predict>0.5] = 1
    y_predict[y_predict<=0.5] = 0
    error_rate.append(sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test))
y_predict_prob = clf.predict(x_test)
fpr, tpr,_ = roc_curve(y_test, y_predict_prob)
auc = roc_auc_score(y_test,  y_predict_prob)
plt.plot(fpr,tpr,label = 'ROC Elastic net regression; AUC=
{}'.format(round(auc,3)))
model_error_rate['Elastic net regression'] = np.mean(error_rate)

#Decision tree (CART)
kf = KFold(n_splits=10)
error_rate = []
for train_indice, test_indice in kf.split(indice):
    x_train = X_scaled_train[train_indice]
    y_train = Y_train[train_indice]
    x_test = X_scaled_train[test_indice]
    y_test = Y_train[test_indice]
    clf = DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
criterion='gini',
                       max_depth=4, 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='random').fit(x_train,
y_train)
    y_predict = clf.predict(x_test)
    error_rate.append(sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test))
y_predict_prob = clf.predict_proba(x_test)[:,1]
fpr, tpr,_ = roc_curve(y_test, y_predict_prob)
auc = roc_auc_score(y_test,  y_predict_prob)
plt.plot(fpr,tpr,label = 'ROC Decision Tree; AUC={}'.format(round(auc,3)))
model_error_rate['Decision tree'] = np.mean(error_rate)

# Bagging
kf = KFold(n_splits=10)
error_rate = []
```

```
for train_indice, test_indice in kf.split(indice):
    x_train = X_scaled_train[train_indice]
    y_train = Y_train[train_indice]
    x_test = X_scaled_train[test_indice]
    y_test = Y_train[test_indice]
    clf = BaggingClassifier(base_estimator=None, bootstrap=True,
bootstrap_features=False,
                  max_features=1.0, max_samples=12, n_estimators=40,
                  n_jobs=None, oob_score=False, random_state=None, verbose=0,
                  warm_start=True).fit(x_train, y_train)
    y_predict = clf.predict(x_test)
    error_rate.append(sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test))
y_predict_prob = clf.predict_proba(x_test)[:,1]
fpr, tpr,_ = roc_curve(y_test, y_predict_prob)
auc = roc_auc_score(y_test,  y_predict_prob)
plt.plot(fpr,tpr,label = 'ROC Bagging; AUC={}'.format(round(auc,3)))
model_error_rate['Bagging'] = np.mean(error_rate)

# Random Forest
kf = KFold(n_splits=10)
error_rate = []
for train_indice, test_indice in kf.split(indice):
    x_train = X_scaled_train[train_indice]
    y_train = Y_train[train_indice]
    x_test = X_scaled_train[test_indice]
    y_test = Y_train[test_indice]
    clf = RandomForestClassifier(bootstrap=False, ccp_alpha=0.0,
class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=6,
                       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).fit(x_train, y_train)
    y_predict = clf.predict(x_test)
    error_rate.append(sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test))
y_predict_prob = clf.predict_proba(x_test)[:,1]
fpr, tpr,_ = roc_curve(y_test, y_predict_prob)
auc = roc_auc_score(y_test,  y_predict_prob)
plt.plot(fpr,tpr,label = 'ROC RF; AUC={}'.format(round(auc,3)))
model_error_rate['RF'] = np.mean(error_rate)

#Boosting
kf = KFold(n_splits=10)
error_rate = []
for train_indice, test_indice in kf.split(indice):
    x_train = X_scaled_train[train_indice]
    y_train = Y_train[train_indice]
```

```
    x_test = X_scaled_train[test_indice]
    y_test = Y_train[test_indice]
    clf = AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
                   learning_rate=0.4, n_estimators=80,
                   random_state=None).fit(x_train, y_train)
    y_predict = clf.predict(x_test)
    error_rate.append(sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test))
y_predict_prob = clf.predict_proba(x_test)[:,1]
fpr, tpr,_ = roc_curve(y_test, y_predict_prob)
auc = roc_auc_score(y_test,  y_predict_prob)
plt.plot(fpr,tpr,label = 'ROC Boosting; AUC={}'.format(round(auc,3)))
model_error_rate['Boosting'] = np.mean(error_rate)

plt.xlabel('1-specificity',size = 16)
plt.ylabel('sensitivity', size = 16)
plt.legend()
plt.show()
#
```




```
sorted(model_error_rate.items(), key = lambda x:x[1])
```

```
[('Boosting', 0.19446589446589446),
 ('RF', 0.19787185144328001),
 ('logistic', 0.20529049457620885),
 ('Elastic net regression', 0.20665563522706379),
 ('Decision tree', 0.22566648280933993),
 ('Bagging', 0.25138812281669426),
 ('Naive Bayes', 0.2649016363302078)]
```

Best model and defend my choice

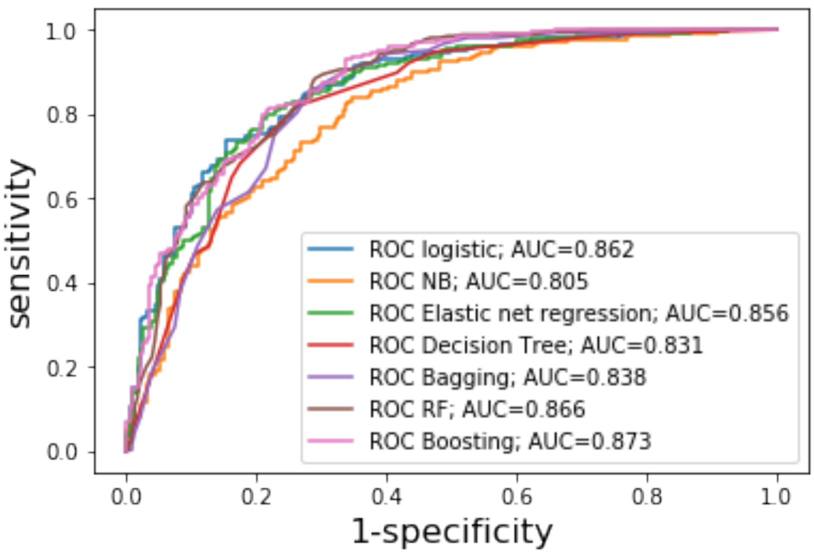
I think the Random Forest is the best model because it has the highest AUC, suggesting when the ture positive rate goes up, the false positive rate will go up relatively slower than other models, and produce a higher accuracy. Moreover, it has a low error rate compared with other models. Although the error rate is not the lowest, but the difference between boosting and random forest is not distinctive.

Question 5: model performance on test dataset

```
model_error_rate = {}

x_train = X_scaled_train
y_train = Y_train
x_test = X_scaled_test
y_test = Y_test
# logistic error rate

clf = LogisticRegression(random_state=0).fit(x_train, y_train)
y_predict = clf.predict(x_test)
error_rate = sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test)
y_predict_prob = clf.predict_proba(x_test)[:,1]
fpr, tpr,_ = roc_curve(y_test, y_predict_prob)
auc = roc_auc_score(y_test,  y_predict_prob)
plt.plot(fpr,tpr,label = 'ROC logistic; AUC={}'.format(round(auc,3)))
model_error_rate['logistic'] = error_rate

# Naive Bayes

clf = GaussianNB().fit(x_train, y_train)
y_predict = clf.predict(x_test)
error_rate = sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test)
y_predict_prob = clf.predict_proba(x_test)[:,1]
fpr, tpr,_ = roc_curve(y_test, y_predict_prob)
auc = roc_auc_score(y_test,  y_predict_prob)
plt.plot(fpr,tpr,label = 'ROC NB; AUC={}'.format(round(auc,3)))
model_error_rate['Naive Bayes'] = error_rate

#Elastic net regression

clf = ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True,
       l1_ratio=0.10000000000000001, max_iter=1000, normalize=False,
       positive=False, precompute=False, random_state=None,
       selection='cyclic', tol=0.0001, warm_start=False).fit(x_train, y_train)
```

```
y_predict = clf.predict(x_test)
y_predict[y_predict>0.5] = 1
y_predict[y_predict<=0.5] = 0
error_rate = sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test)
y_predict_prob = clf.predict(x_test)
fpr, tpr,_ = roc_curve(y_test, y_predict_prob)
auc = roc_auc_score(y_test,  y_predict_prob)
plt.plot(fpr,tpr,label = 'ROC Elastic net regression; AUC=
{}'.format(round(auc,3)))
model_error_rate['Elastic net regression'] = error_rate

#Decision tree (CART)

clf = DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
criterion='gini',
                       max_depth=4, 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='random').fit(x_train, y_train)
y_predict = clf.predict(x_test)
error_rate = sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test)
y_predict_prob = clf.predict_proba(x_test)[:,1]
fpr, tpr,_ = roc_curve(y_test, y_predict_prob)
auc = roc_auc_score(y_test,  y_predict_prob)
plt.plot(fpr,tpr,label = 'ROC Decision Tree; AUC={}'.format(round(auc,3)))
model_error_rate['Decision tree'] = error_rate

# Bagging

clf = BaggingClassifier(base_estimator=None, bootstrap=True,
bootstrap_features=False,
                  max_features=1.0, max_samples=12, n_estimators=40,
                  n_jobs=None, oob_score=False, random_state=None, verbose=0,
                  warm_start=True).fit(x_train, y_train)
y_predict = clf.predict(x_test)
error_rate = sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test)
y_predict_prob = clf.predict_proba(x_test)[:,1]
fpr, tpr,_ = roc_curve(y_test, y_predict_prob)
auc = roc_auc_score(y_test,  y_predict_prob)
plt.plot(fpr,tpr,label = 'ROC Bagging; AUC={}'.format(round(auc,3)))
model_error_rate['Bagging'] = error_rate

# Random Forest

clf = RandomForestClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=6,
                       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).fit(x_train, y_train)
y_predict = clf.predict(x_test)
error_rate = sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test)
y_predict_prob = clf.predict_proba(x_test)[:,1]
fpr, tpr,_ = roc_curve(y_test, y_predict_prob)
auc = roc_auc_score(y_test,  y_predict_prob)
plt.plot(fpr,tpr,label = 'ROC RF; AUC={}'.format(round(auc,3)))
model_error_rate['RF'] = error_rate

#Boosting

clf = AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
                   learning_rate=0.4, n_estimators=80,
                   random_state=None).fit(x_train, y_train)
y_predict = clf.predict(x_test)
error_rate = sum(np.ones(len(y_test))[y_predict!=y_test])/len(y_test)
y_predict_prob = clf.predict_proba(x_test)[:,1]
fpr, tpr,_ = roc_curve(y_test, y_predict_prob)
auc = roc_auc_score(y_test,  y_predict_prob)
plt.plot(fpr,tpr,label = 'ROC Boosting; AUC={}'.format(round(auc,3)))
model_error_rate['Boosting'] = error_rate

plt.xlabel('1-specificity',size = 16)
plt.ylabel('sensitivity', size = 16)
plt.legend()
plt.show()
#
```




```
sorted(model_error_rate.items(), key = lambda x:x[1])
```

```
[('RF', 0.19675456389452334),
 ('Elastic net regression', 0.20892494929006086),
 ('Boosting', 0.20892494929006086),
 ('Bagging', 0.21095334685598377),
 ('Decision tree', 0.21906693711967545),
 ('logistic', 0.2231237322515213),
 ('Naive Bayes', 0.26977687626774849)]
```

comments:

our best model: random forest in question 4 has a second highest AUC among the other models and the lowest error rate, suggesting the random forest can accomplish a good classification task with a good generalization ability.