Perspective on Computational Modeling

Problem Set 5
Tianyue Niu

In [164]:
```
#import necessary packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression,ElasticNet, SGDClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier, GradientBoosting
Classifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score, roc_curve, auc

#disable future warning
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [12]:
```
train = pd.read_csv('gss_train.csv')
test = pd.read_csv('gss_test.csv')
```

1.(15 points) Consider the Gini index, classification error, and cross-entropy in simple classification settings with two classes. Of these three possible cost functions, which would be best to use when growing a decision tree? Which would be best to use when pruning a decision tree? Why?

```
When growing a decision tree, we should use cross entropy because it allows us to continue s
plitting even if all end nodes from a branch have the same classification result. Further sp
lit is worth it becasue we get different node purities for these two nodes, so we will be ab
le to predict/label the outcome with different certainties.

We can use classification error for pruning because in the end, we want our tree to be able
 to maximize prediction accuracy.
```

2.(35 points; 5 points/model) Estimate the following models, predicting colrac using the training set with 10-fold CV:

- Logistic regression
- Naive Bayes
- Elastic net regression
- Decision tree (CART)
- Bagging
- Random forest
- Boosting

In [13]:
```
X_train = train.drop(columns="colrac")
y_train = train["colrac"]
```

Logistic Regression

In [39]:
```
#Logistic Regression Model
lr = LogisticRegressiaon()
lr_error = 1-np.mean(cross_val_score(lr, X_train, y_train, cv=10, scoring='accuracy'))
print("Logistic regression 10-fold cross-validated classification error:", lr_error)
lr = lr.fit(X_train,y_train)
```

```
Logistic regression 10-fold cross-validated classification error: 0.20731955760718945
```

Naive Bayes Classifier

In [49]:
```
#Naive Bayes
gnb = GaussianNB()
nb_error = 1-np.mean(cross_val_score(gnb, X_train, y_train, cv=10, scoring='accuracy'))
print("Naive Bayes 10-fold cross-validated classification error:", nb_error)
gnb = gnb.fit(X_train, y_train)
```

```
Naive Bayes 10-fold cross-validated classification error: 0.26555250977590383
```

Elastic Net Regression

(Assumed this means Elastic Net Logistic Regression because otherwise it doesn't make sense)

In [80]:
```
#Elastic Net Logistic Regression
#define a function to find the best lambda and alpha
def find_best_para():
    min_error = 10000
    best_model = None
    best_lambda = None
    best_alpha = None

    for i in np.arange(0.1,1,0.1):
        for j in np.arange(0.1,1,0.1):
            elastic = SGDClassifier(loss="log", penalty="elasticnet", l1_ratio=i, alpha=
j)
            error = 1-np.mean(cross_val_score(elastic, X_train, y_train, cv=10, scoring=
'accuracy'))
            if error < min_error:
                min_error = error
                best_model = elastic
                best_lambda = i
                best_alpha = j

    return best_lambda, best_alpha, min_error
```

In [81]:
```
lam, alph, elastic_net_error = find_best_para()
```

In [84]:
```
"The best model has lambda = {}, alplha ={}, \
and the 10-fold cross validated MSE of this model is equal to {}".format(lam, alph, elas
tic_net_error)
```

Out[84]:
```
'The best model has lambda = 0.1, alplha =0.1, and the 10-fold cross validated MSE of t
his model is equal to 0.23985323496349387'
```

In [85]:
```
elastic = ElasticNet(l1_ratio=0.1, alpha=0.1).fit(X_train, y_train)
```

Decision Tree

In [86]:
```
dt = tree.DecisionTreeClassifier()
dt_error = 1-np.mean(cross_val_score(dt, X_train, y_train, cv=10, scoring='accuracy'))
print("Decision Tree 10-fold cross-validated classification error:", dt_error)
dt = dt.fit(X_train, y_train)
```

```
Decision Tree 10-fold cross-validated classification error: 0.27102698507300615
```

Bagging

In [114]:
```
def find_best_bootstrap_size():
    min_error = 10000
    best_model = None
    best_size = None

    for size in np.arange(0.1,1.0,0.1):
        bagging = BaggingClassifier(max_samples=size, random_state=666)
        error = 1-np.mean(cross_val_score(bagging, X_train, y_train, cv=10, scoring='acc
uracy'))

        if error < min_error:
            min_error = error
            best_model = bagging
            best_size = size

    return best_model, best_size, min_error
```

In [115]:
```
bagging, size, bagging_error = find_best_bootstrap_size()
```

In [116]:
```
print("Best bootstrap sample size is:", size, "with error of", bagging_error)
```

```
Best bootstrap sample size is: 0.4 with error of 0.22426213365810677
```

In [132]:
```
bagging = BaggingClassifier(max_samples=size, random_state = 666).fit(X_train, y_train)
```

In [133]:
```
bagging
```

Out[133]:
```
BaggingClassifier(base_estimator=None, bootstrap=True, bootstrap_features=False,
                  max_features=1.0, max_samples=0.4, n_estimators=10,
                  n_jobs=None, oob_score=False, random_state=666, verbose=0,
                  warm_start=False)
```

Random Forest

In [127]:
```
def find_best_para_rf():
    min_error = 10000
    best_model = None
    best_feature = None

    for feature in np.arange(0.1,1.0,0.1):
        rf = RandomForestClassifier(max_features=feature, random_state=666)
        error = 1-np.mean(cross_val_score(rf, X_train, y_train, cv=10, scoring='accurac
y'))
        if error < min_error:
            min_error = error
            best_model = bagging
            best_feature = feature

    return best_model, best_feature, min_error
```

In [130]:
```
rf, feature, rf_error = find_best_para_rf()
```

In [131]:
```
print("Best feature size is:", feature, "with error of",rf_error)
```

```
Best feature size is: 0.5 with error of 0.21684784016998793
```

In [134]:
```
rf = RandomForestClassifier(max_features=feature, random_state=666).fit(X_train, y_train
)
```

Boosting

In [140]:
```
def find_best_para_boost():
    min_error = 10000
    best_model = None
    best_shrinkage = None

    for s in np.arange(0.1,1.0,0.1):
        boosting = GradientBoostingClassifier(learning_rate=s, random_state=666)
        error = 1-np.mean(cross_val_score(boosting, X_train, y_train, cv=10, scoring='ac
curacy'))
        if error < min_error:
            min_error = error
            best_model = boosting
            best_shrinkage = s

    return best_model, best_shrinkage, min_error
```

In [141]:
```
boosting, rate, boosting_error = find_best_para_boost()
```

In [142]:
```
print("Best shrinkage parameter is:", rate, "with error of",boosting_error)
```

```
Best shrinkage parameter is: 0.1 with error of 0.19855391276771905
```

In [143]:
```
boosting = GradientBoostingClassifier(learning_rate=rate, random_state=666).fit(X_train,
y_train)
```

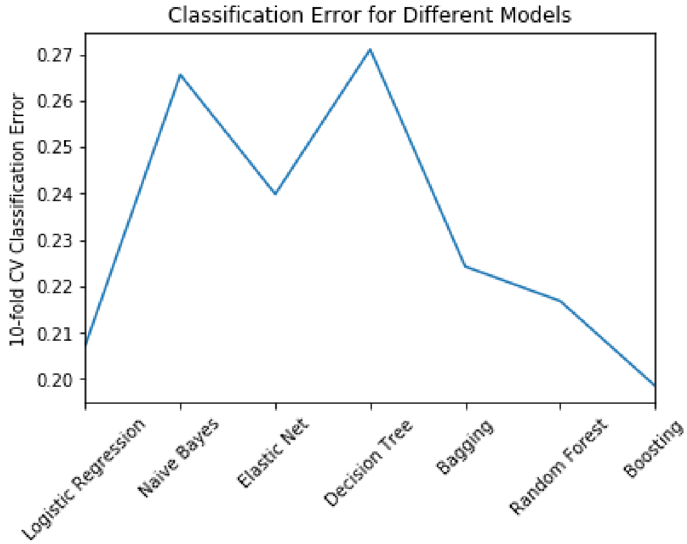
3.(20 points) Compare and present each model's (training) performance based on:

- *Cross-validated error rate*
- *ROC/AUC*

In [170]:
```
labels = ['Logistic Regression', 'Naive Bayes', 'Elastic Net', "Decision Tree" ,
          'Bagging','Random Forest','Boosting']
errors = [lr_error, nb_error, elastic_net_error, dt_error, bagging_error, rf_error, boos
ting_error]
print(pd.DataFrame(errors, labels))
```

```
                            0
Logistic Regression  0.207320
Naive Bayes          0.265553
Elastic Net          0.239853
Decision Tree        0.271027
Bagging              0.224262
Random Forest        0.216848
Boosting             0.198554
```

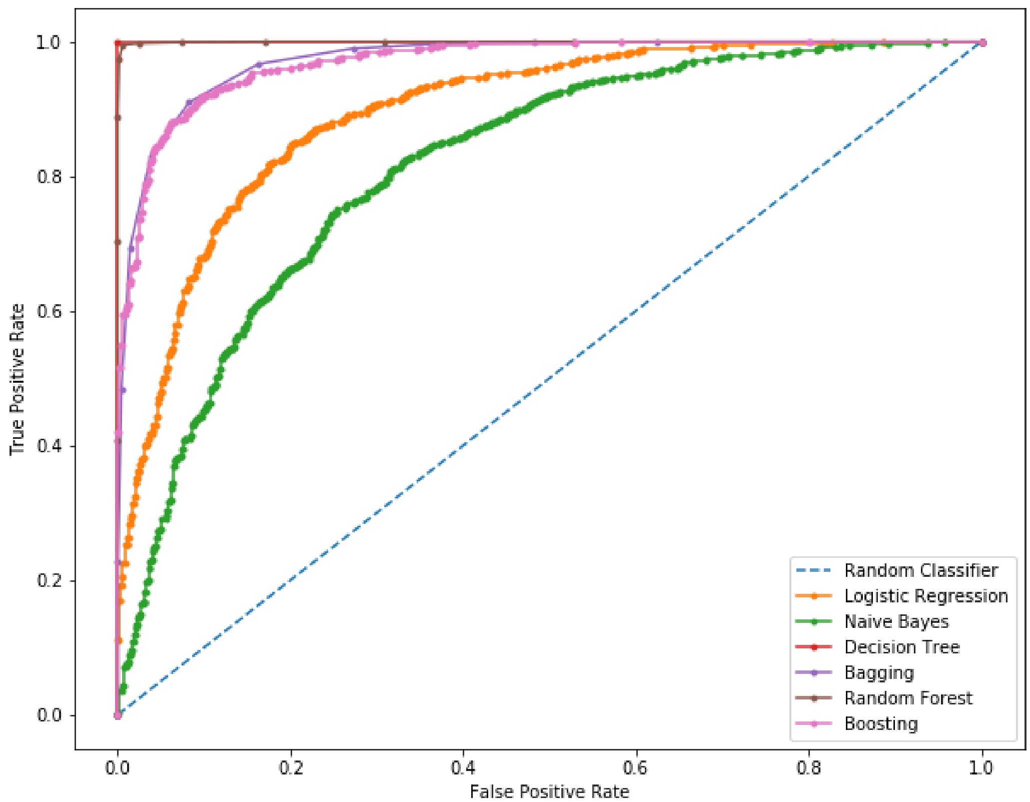
In [177]:
```
pd.DataFrame(errors, labels).plot(legend=False)
plt.xticks(rotation=45)
plt.ylabel('10-fold CV Classification Error')
plt.title('Classification Error for Different Models');
```



In [165]:
```
def get_roc_auc(model, label):
    # predict probabilities
    model_probs = model.predict_proba(X_train)
    # keep probabilities for the positive outcome only
    model_probs = model_probs[:, 1]
    #calcualte auc score
    model_auc = roc_auc_score(y_train, model_probs)
    # summarize scores
    print(label+ ': AUC = %.3f' % (model_auc))
    # calculate roc curves
    model_fpr, model_tpr, _ = roc_curve(y_train, model_probs)
    # plot the roc curve for the model
    plt.plot(model_fpr, model_tpr, marker='.', label=label)
    # axis labels
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    # show the legend
    plt.legend()

def plot_random_classifier():
    plt.figure(figsize=(10,8))
    # generate a random prediction line
    random_probs = [0 for _ in range(len(y_train))]
    # calculate scores
    random_auc = roc_auc_score(y_train, random_probs)
    random_fpr, random_tpr, _ = roc_curve(y_train, random_probs)
    plt.plot(random_fpr, random_tpr, linestyle='--', label='Random Classifier')
```

In [167]:
```
plot_random_classifier()

models = [lr, gnb, dt, bagging, rf, boosting]
labels = ['Logistic Regression', 'Naive Bayes', "Decision Tree" ,
          'Bagging','Random Forest','Boosting']

count = 0
for model in models:
    get_roc_auc(model, labels[count])
    count+=1
```

```
Logistic Regression: AUC = 0.896
Naive Bayes: AUC = 0.816
Decision Tree: AUC = 1.000
Bagging: AUC = 0.973
Random Forest: AUC = 1.000
Boosting: AUC = 0.970
```



4.(15 points) Which is the best model? Defend your choice.

From the above graph we see that the best models are Rnadom Forest and Decision Tree. This is very surprising. It could be that, because I didn't limit decision tree's depth, it has tremendously overfitted, and so on the training data set it can perfectly predict every outcome. Random Forest has the same problem, because I did not limit the max-depth we might have overfitted.

We see that following the 'perfect' models, bagging and boosting also did pretty well as well (around 0.97). In reality, I would probably choose boosting over all the other models because it has the lowest 10-fold cross-validated classification error. This error rate should be a more realistic estimate of the model's performance on a testing data set.

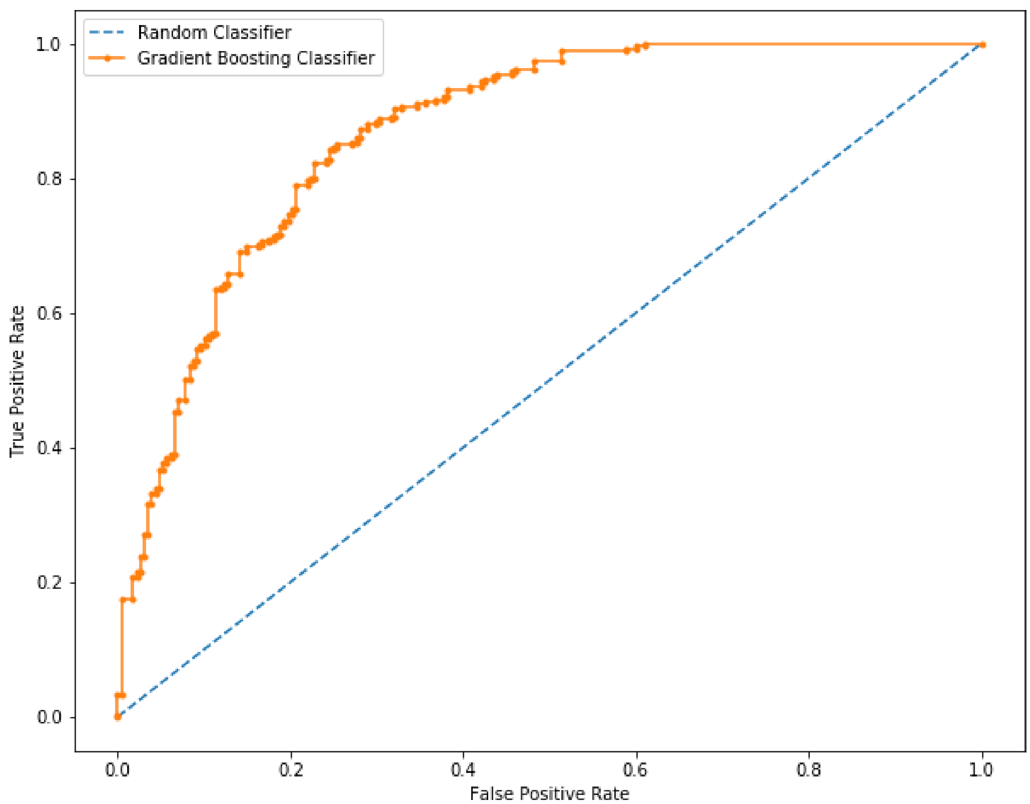
*5.(15 points) Evaluate the final, best model's (selected in 4) performance on the test set (the test .csv) by calculating and presenting the classification error rate and AUC. Compared to the fit evaluated on the training set in questions 3-4, does the "best" model generalize well? Why or why not? How do you know?**

In [178]:
```
X_test = test.drop(columns="colrac")
y_test = test["colrac"]
```

In [181]:
```
plot_random_classifier()
boosting_prob = boosting.predict_proba(X_test)
boosting_prob = boosting_prob[:,1]
boosting_auc = roc_auc_score(y_test, boosting_prob)
print("Gradient Boosting Classifier Test Data Performance", boosting_auc)
model_fpr, model_tpr, _ = roc_curve(y_test, boosting_prob)
plt.plot(model_fpr, model_tpr, marker='.', label="Gradient Boosting Classifier")
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
```

```
Gradient Boosting Classifier Test Data Performance 0.8702250910294604
```

Out[181]:
```
<matplotlib.legend.Legend at 0x1a2572f128>
```




```
Seeing from the above graph, I would say that the model generalized pretty well with an AUC
 = 0.87. The estimated test error was 0.199, which is slighly lower than the actual error. H
owever, the cross-validated error rate is already a pretty good estimation of the true class
ification error.
```