Homework 5

YIMIN LI

1. Conceptual: Cost functions for classification trees

Classification error focuses more on the accuracy of the classification while the other two (cross entropy and Gini index) pays more attention to the purity of the classification. Thus, when growing a decision tree, Gini index and cross-entropy are better choices since they can control the variance and classification error is not sensitive enough for tree-growing.

On the other hand, when doing accuracy-based pruning, apparently, classification error is the best to use considering its character to maximize the accuracy.

Application: Predicting attitudes towards racist college professors

2.

In [57]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression, ElasticNet, Elasti
cNetCV
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier,
 GradientBoostingClassifier
```

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score, roc_curve, auc
```

In [58]:
```
df_train = pd.read_csv("https://raw.githubusercontent.com/macss-model2
0/problem-set-5/master/data/gss_train.csv")
df_test = pd.read_csv("https://raw.githubusercontent.com/macss-model20/
problem-set-5/master/data/gss_test.csv")
```

In [59]:
```
x_train = df_train.drop(['colrac'], axis=1)
y_train = df_train['colrac']
x_test = df_test.drop(['colrac'], axis=1)
y_test = df_test['colrac']
```

In [74]:
```
import warnings
warnings.filterwarnings('ignore')
# Logistic Regression
lr = LogisticRegression()
lr_error = 1 - np.mean(cross_val_score(lr, x_train, y_train, cv=10))
print("Logisitic Regression Test Error: " + str(lr_error))
lr = lr.fit(x_train, y_train)
```

```
Logisitic Regression Test Error: 0.20731955760718945
```

In [75]:
```
#Naive Bayes
gnb = GaussianNB()
gnb_error = 1 - np.mean(cross_val_score(gnb, x_train, y_train, cv=10))
print("Naive Bayes Test Error: " + str(gnb_error))
gnb = gnb.fit(x_train, y_train)
```

```
Naive Bayes Test Error: 0.26555250977590383
```

In [76]:
```
# ElasticNet
elastic = ElasticNetCV(cv=10, alphas = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6
, 0.7, 0.8, 0.9, 1]).fit(x_train, y_train)
elastic_ = ElasticNet(alpha=elastic.l1_ratio_, l1_ratio=elastic.alpha_)
elastic_error = -np.mean(cross_val_score(elastic_, x_train, y_train, cv
=10, scoring = 'neg_mean_squared_error'))
```

```
print("ElasticNet Error: " + str(elastic_error))
elastic_ = ElasticNet(alpha=elastic.l1_ratio_, l1_ratio=elastic.alpha_)
.fit(x_train, y_train)
```

```
ElasticNet Error: 0.16101423184432828
```

In [77]:

```
# Decision Tree
dt = DecisionTreeClassifier()
dt_error = 1 - np.mean(cross_val_score(dt, x_train, y_train, cv=10))
print("Decision Tree Error: " + str(dt_error))
dt.fit(x_train, y_train)
```

```
Decision Tree Error: 0.2669178355180273
```

Out[77]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=N
one,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=No
ne,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [78]:

```
# Bagging
bagging_error = 1
sample = 0
for i in np.arange(0.1, 1, 0.1):
    bagging = BaggingClassifier(max_samples=i)
    temp_error = 1 - np.mean(cross_val_score(bagging, x_train, y_train,
 cv=10))
    if temp_error < bagging_error:
        bagging_error = temp_error
        sample = i
bagging = BaggingClassifier(max_samples=sample)
print("Bagging Tree Error: " + str(bagging_error))
bagging.fit(x_train, y_train)
```

```
Bagging Tree Error: 0.21816248175308584
```

Out[78]:

```
BaggingClassifier(base_estimator=None, bootstrap=True, bootstrap_featur
```

```
es=False,
                          max_features=1.0, max_samples=0.4, n_estimators=10,
                          n_jobs=None, oob_score=False, random_state=None, verb
ose=0,
                          warm_start=False)
```

In [80]:

```
# Random Forest
forest_error = 1
feature = 0
for i in np.arange(0.1, 1, 0.1):
    forest = RandomForestClassifier(max_features=i)
    temp_error = 1 - np.mean(cross_val_score(forest, x_train, y_train,
cv=10))
    if temp_error < forest_error:
        forest_error = temp_error
        feature = i
forest = RandomForestClassifier(max_features=feature)
print("Random Forest Error: " + str(forest_error))
forest.fit(x_train, y_train)
```

```
Random Forest Error: 0.22291531704283374
```

Out[80]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gi
ni',
                       max_depth=None, max_features=0.3000000000000000
4,
                       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=10, n_jobs=None, oob_score=False,
                       random_state=None, verbose=0, warm_start=False)
```

In [81]:

```
# Boosting
boosting_error = 1
feature_boost = 0
for i in np.arange(0.1, 1, 0.1):
    boosting = GradientBoostingClassifier(max_features=i)
    temp_error = 1 - np.mean(cross_val_score(boosting, x_train, y_train
```

```
, cv=10))
    if temp_error < boosting_error:
        boosting_error = temp_error
        feature_boost = i
boosting = GradientBoostingClassifier(max_features=feature_boost)
print("Boosting Error: " + str(boosting_error))
boosting.fit(x_train, y_train)
```

```
Boosting Error: 0.19309794659746715
```

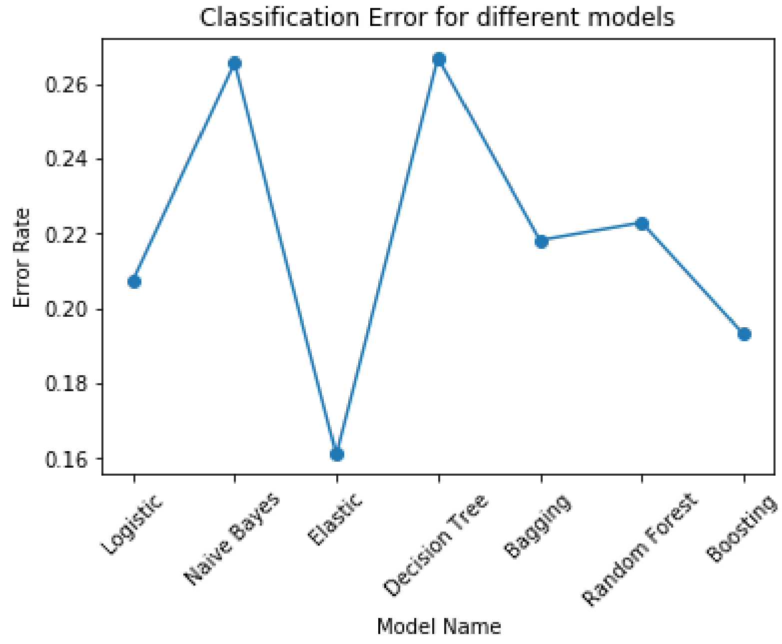
Out[81]:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_dept
h=3,
                           max_features=0.30000000000000004,
                           max_leaf_nodes=None, min_impurity_decrease=
0.0,
                           min_impurity_split=None, min_samples_leaf=1,
                           min_samples_split=2, min_weight_fraction_lea
f=0.0,
                           n_estimators=100, n_iter_no_change=None,
                           presort='auto', random_state=None, subsample
=1.0,
                           tol=0.0001, validation_fraction=0.1, verbose
=0,
                           warm_start=False)
```

3.

In [82]:

```
ls_names = ['Logistic', 'Naive Bayes', 'Elastic', 'Decision Tree', 'Bag
ging', 'Random Forest', 'Boosting']
ls_values = [lr_error, gnb_error, elastic_error, dt_error, bagging_erro
r, forest_error, boosting_error]
plt.plot(ls_names, ls_values, marker='o')
plt.xticks(rotation=45)
plt.xlabel('Model Name')
plt.ylabel('Error Rate')
plt.title('Classification Error for different models')
plt.show()
```



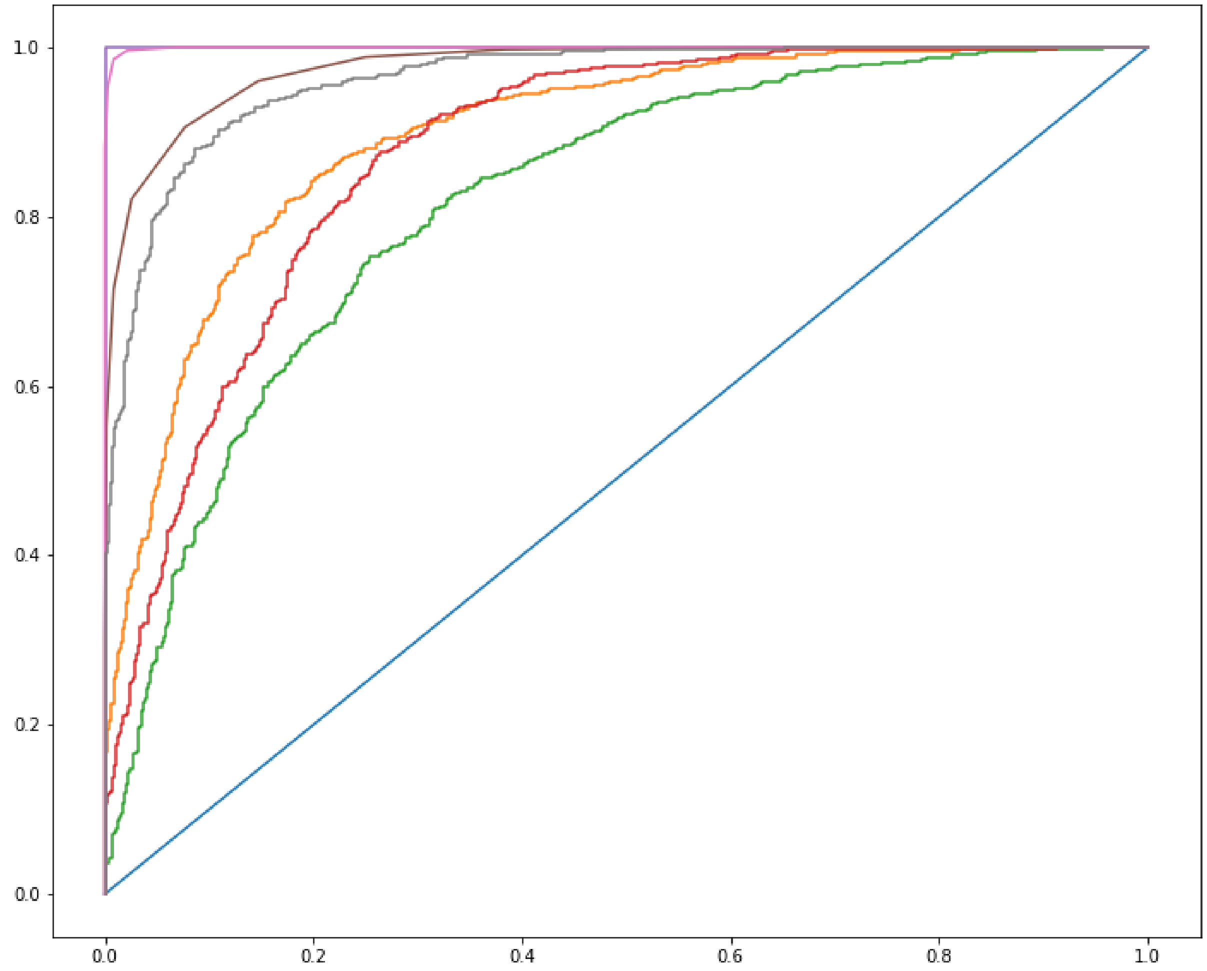
In [89]:
```
def roc_auc(model, name):
    if name == 'Elastic Net':
        probs = model.predict(x_train)
    else:
        probs = model.predict_proba(x_train)[:,1]
    model_auc = roc_auc_score(y_train, probs)
    print(name + ' : ' + str(model_auc))
    fpr, tpr, _ = roc_curve(y_train, probs)
    plt.plot(fpr, tpr, label=name)

plt.figure(figsize=(12,10))
random_probs = [0] * len(y_train)
random_fpr, random_tpr, _ = roc_curve(y_train, random_probs)
plt.plot(random_fpr, random_tpr, label='Random Classifier')

models = [lr, gnb, elastic_, dt, bagging, forest, boosting]
names = ['Logistic Regression', 'Naive Bayes', 'Elastic Net', 'Decision
 Tree', 'Bagging', 'Random Forest', 'Boosting']
```

```python
for i in range(len(models)):
    roc_auc(models[i], names[i])
```

```
Logistic Regression : 0.8958943444848373
Naive Bayes : 0.816411577930146
Elastic Net : 0.8742349638764898
Decision Tree : 1.0
Bagging : 0.9769766692743086
Random Forest : 0.9993833693801483
Boosting : 0.9615277713864986
```



4.

Considering AUC, Decision Tree, Bagging, Random forest and Boosting all have very high AUCs, all approached to 1 while in contrast, the other three is around 0.8-0.85. Taking error rate into consideration, boosting is the best model, becuase it ranks 2nd in the error rate and is in the first tier of the AUC rates. It is interesting to see that decision tree models rank 1st in AUC but rank last in error rate, and vice versa for Elastic Net model. So all in all, boosting is the best model in taking two factors into consideration and Random Forest and Bagging also performed well in both factors.

5.

In [90]:
```
boosting_pred = boosting.predict(x_test)
boosting_error_test = 1 - np.mean(cross_val_score(boosting, x_train, y_
train, cv=10))
boosting_auc = np.mean(cross_val_score(boosting, x_train, y_train, cv=1
0, scoring='roc_auc'))
print("Boosting Test Error Rate: " + str(boosting_error_test))
print("Boosting Test AUC: " + str(boosting_auc))
```

```
Boosting Test Error Rate: 0.20325159085945288
Boosting Test AUC: 0.8802773685067449
```

Compared to the training fit, boosting model fluctuates a little in terms of generalizing it into test data: raising the test error from 0.19 to 0.20, decreasing the AUC from 0.96 to 0.88. Generally speaking, this fluctuation is fully acceptable and hence this best model generalizes pretty well from training data to the test data.