Jiang_Luying_HW5

March 1, 2020

0.1 Conceptual: Cost functions for classification trees

1.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?

Gini index and cross-entropy measure the purity of the classification, while classification error measures the accuracy. Gini index and cross-entropy are quite similar numerically. Either the Gini index or the entropy are typically used to evaluate the quality of a particular split, since these two approaches are more sensitive to node purity than is the classification error rate. Classification error uses greedy approach which may lead to overfitting problem. Any of these three approaches might be used when pruning the tree, but the classification error rate is preferable if prediction accuracy of the final pruned tree is the goal.

0.2 Application: Predicting attitudes towards racist college professors

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import ElasticNet
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score, accuracy_score, roc_curve
```

[2]:
```
import warnings
warnings.filterwarnings('ignore')
```

[3]:
```
seed = 1234
```

0.2.1 Estimate the models

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

[4]:
```
gss_train = pd.read_csv("gss_train.csv")
gss_test = pd.read_csv("gss_test.csv")
X_train = gss_train.drop('colrac',axis=1)
X_test = gss_test.drop('colrac',axis=1)
y_train = gss_train['colrac']
y_test = gss_test['colrac']
```

[5]:
```
X_train.head()
```

[5]:

	age	attend	authoritarianism	black	born	childs	colath	colcom	colmil	\
0	21	0	4	0	0	0	1	0	1	
1	42	0	4	0	0	2	0	1	0	
2	70	1	1	1	0	3	0	1	0	
3	35	3	2	0	0	2	0	0	1	
4	24	3	6	0	1	3	1	0	0	

	colhomo	...	zodiac_GEMINI	zodiac_CANCER	zodiac_LEO	zodiac_VIRGO	\
0	1	...	0	0	0	0	
1	0	...	0	0	0	0	
2	0	...	0	0	0	0	
3	0	...	0	0	0	0	
4	0	...	0	0	0	0	

	zodiac_LIBRA	zodiac_SCORPIO	zodiac_SAGITTARIUS	zodiac_CAPRICORN	\
0	0	0	0	0	
1	0	0	0	0	
2	0	0	0	0	
3	0	1	0	0	
4	0	1	0	0	

	zodiac_AQUARIUS	zodiac_PISCES
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0

[5 rows x 77 columns]

[6]:
```
best_score = []
best_estimator = []
```

[7]:
```
# Logistic regression
LG = LogisticRegression()
grid_val = {}
lg_gscv = GridSearchCV(LG, grid_val, cv=10, refit=True, scoring='accuracy')
lg_gscv.fit(X_train, y_train)
best_score.append(lg_gscv.best_score_)
best_estimator.append(lg_gscv.best_estimator_)
```

```
print('Mean cross-validated score of the best_estimator: ', lg_gscv.best_score_)
print(lg_gscv.best_estimator_)
y_pred = lg_gscv.predict(X_test)
```

```
Mean cross-validated score of the best_estimator:  0.7974341661039838
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

[8]:
```
# Naive Bayes
NB = GaussianNB()
grid_val = {}
nb_gscv = GridSearchCV(NB, grid_val, cv=10, refit=True, scoring='accuracy')
nb_gscv.fit(X_train, y_train)
best_score.append(nb_gscv.best_score_)
best_estimator.append(nb_gscv.best_estimator_)
print('Mean cross-validated score of the best_estimator: ', nb_gscv.best_score_)
print(nb_gscv.best_estimator_)
y_pred = nb_gscv.predict(X_test)
```

```
Mean cross-validated score of the best_estimator:  0.7150573936529372
GaussianNB(priors=None, var_smoothing=1e-09)
```

[9]:
```
# Elastic net regression
EN = ElasticNet()
grid_val = {"max_iter": [1, 5, 10],"alpha": [0.0001, 0.001, 0.01, 0.1, 1, 10,␣
 ↪100],
               "l1_ratio": np.arange(0.0, 1.0, 0.1)}
en_gscv = GridSearchCV(EN, grid_val, cv = 10, refit=True, scoring=␣
 ↪'neg_mean_squared_error')
en_gscv.fit(X_train, y_train)
best_score.append(en_gscv.best_score_)
best_estimator.append(en_gscv.best_estimator_)
print('Mean cross-validated score of the best_estimator: ', en_gscv.best_score_)
print(en_gscv.best_estimator_)
y_pred = en_gscv.predict(X_test)
```

```
Mean cross-validated score of the best_estimator:  -0.14713768187577483
ElasticNet(alpha=0.01, copy_X=True, fit_intercept=True,
           l1_ratio=0.30000000000000004, max_iter=10, normalize=False,
           positive=False, precompute=False, random_state=None,
           selection='cyclic', tol=0.0001, warm_start=False)
```

[10]:
```
# Decision Tree
DT = DecisionTreeClassifier()
grid_val = {'criterion': ['gini', 'entropy'], "min_samples_split": [2, 10, 20],
            "min_samples_leaf": [1, 5, 10], 'max_depth': np.arange(3, 10)}
dt_gscv = GridSearchCV(DT, grid_val, cv = 10, refit=True,scoring = 'accuracy')
dt_gscv.fit(X_train, y_train)
best_score.append(dt_gscv.best_score_)
best_estimator.append(dt_gscv.best_estimator_)
print('Mean cross-validated score of the best_estimator: ', dt_gscv.best_score_)
print(dt_gscv.best_estimator_)
y_pred = dt_gscv.predict(X_test)
```

```
Mean cross-validated score of the best_estimator:  0.775151924375422
DecisionTreeClassifier(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=5, min_samples_split=20,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

[11]:
```
# Bagging
BG = BaggingClassifier()
grid_val = {'n_estimators': range(10,50,5)}
bg_gscv = GridSearchCV(BG, grid_val, cv = 10, refit=True, scoring = 'accuracy')
bg_gscv.fit(X_train, y_train)
best_score.append(bg_gscv.best_score_)
best_estimator.append(bg_gscv.best_estimator_)
print('Mean cross-validated score of the best_estimator: ', bg_gscv.best_score_)
print(bg_gscv.best_estimator_)
y_pred = bg_gscv.predict(X_test)
```

```
Mean cross-validated score of the best_estimator:  0.7960837272113437
BaggingClassifier(base_estimator=None, bootstrap=True, bootstrap_features=False,
                  max_features=1.0, max_samples=1.0, n_estimators=40,
                  n_jobs=None, oob_score=False, random_state=None, verbose=0,
                  warm_start=False)
```

[12]:
```
# Random forest
RF = RandomForestClassifier()
grid_val = {"n_estimators": [10, 25, 50, 100], "max_depth": [5, 10, 20, 30],
            "min_samples_leaf": [1,2,4]}
rf_gscv = GridSearchCV(RF, grid_val, cv = 10, refit=True, scoring = 'accuracy')
rf_gscv.fit(X_train, y_train)
best_score.append(rf_gscv.best_score_)
best_estimator.append(rf_gscv.best_estimator_)
print('Mean cross-validated score of the best_estimator: ', rf_gscv.best_score_)
print(rf_gscv.best_estimator_)
```

```
y_pred = rf_gscv.predict(X_test)
```

```
Mean cross-validated score of the best_estimator:  0.8021607022282242
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=20, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=4, 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)
```

[13]:
```
# Boosting
BS = GradientBoostingClassifier()
grid_val = {'loss': ['deviance', 'exponential'], 'learning_rate': np.linspace(.
 ↪1,1,11)}
bs_gscv = GridSearchCV(BS, grid_val, cv = 10, refit=True, scoring = 'accuracy')
bs_gscv.fit(X_train, y_train)
best_score.append(bs_gscv.best_score_)
best_estimator.append(bs_gscv.best_estimator_)
print('Mean cross-validated score of the best_estimator: ', bs_gscv.best_score_)
print(bs_gscv.best_estimator_)
y_pred = bs_gscv.predict(X_test)
```

```
Mean cross-validated score of the best_estimator:  0.8014854827819041
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='exponential', 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, 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)
```

0.2.2 Evaluate the models

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

- Cross-validated error rate
- ROC/AUC

[19]:
```
model = ['Logistic Regression', 'Naive Bayes', 'Elastic Net', 'Decision tree␣
 ↪(CART)',
         'Bagging', 'Random forest', 'Boosting']
error_rate = {}
```

```
for i in range(len(best_score)):
    error_rate[model[i]] = 1 - best_score[i]
error_rate['Elastic Net'] = 1 - accuracy_score(y_train, en_gscv.
 ↪predict(X_train) >= 0.5)
sorted_x = sorted(error_rate.items(), key=lambda kv: kv[1])
sorted_x
```

[19]:
```
[('Elastic Net', 0.18433490884537473),
 ('Random forest', 0.19783929777177578),
 ('Boosting', 0.19851451721809588),
 ('Logistic Regression', 0.20256583389601623),
 ('Bagging', 0.20391627278865632),
 ('Decision tree (CART)', 0.22484807562457798),
 ('Naive Bayes', 0.2849426063470628)]
```

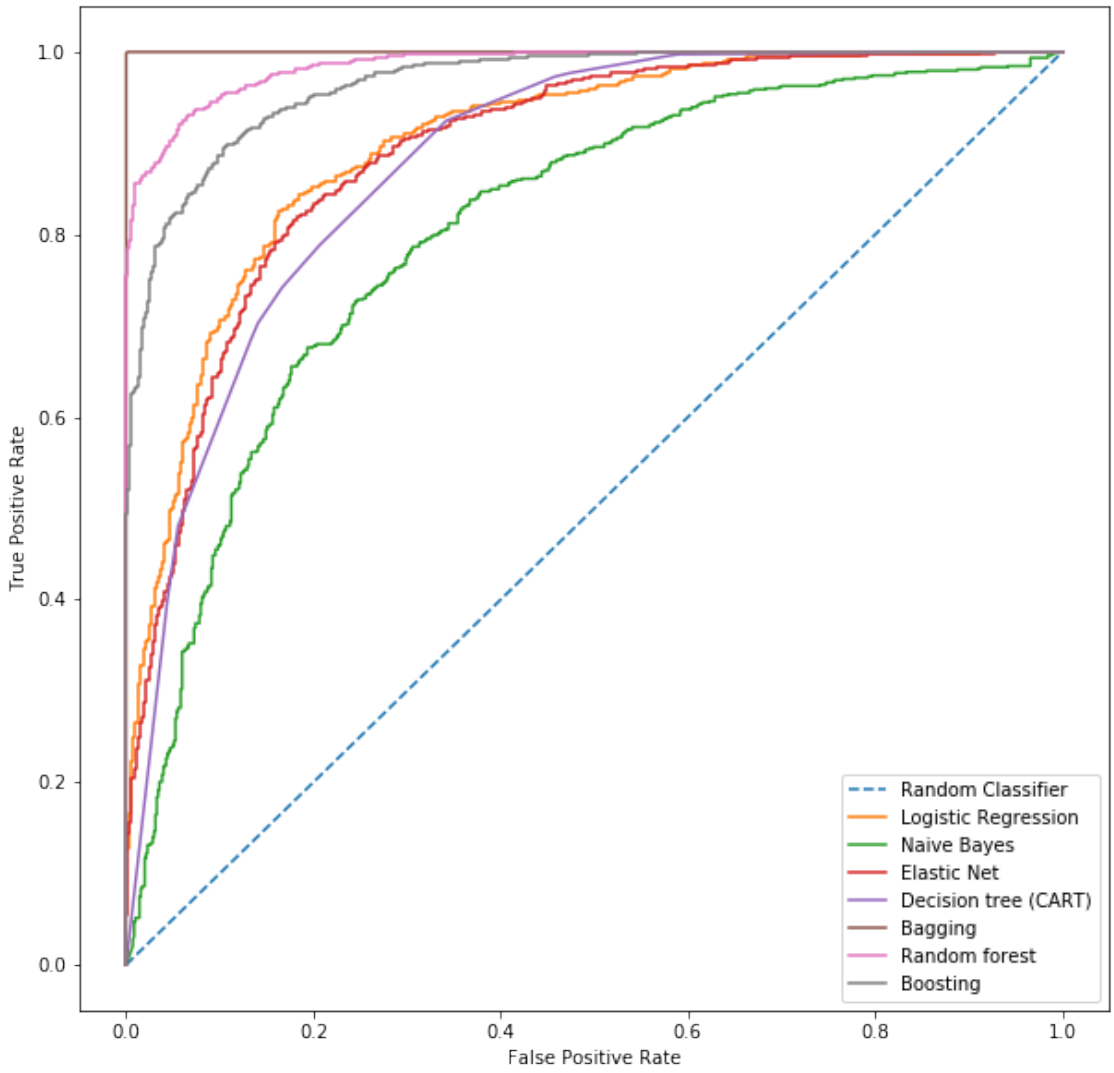
Elastic Net Regression has the lowest Cross-Validated Error rate of 0.1843.

[24]:
```
plt.figure(figsize=(10, 10))
rand_probs = [0] * len(y_train)
rand_fpr, rand_tpr, _ = roc_curve(y_train, rand_probs)
plt.plot(rand_fpr, rand_tpr, linestyle='--', label='Random Classifier')
auc_dict = {}
def roc_auc(model, name):
    if name == 'Elastic Net':
        probs = model.predict(X_train)
    else:
        probs = model.predict_proba(X_train)[:, 1]
    auc = roc_auc_score(y_train, probs)
    auc_dict[name] = ': ROC/AUC = %.9f' % (auc)
    fpr, tpr, _ = roc_curve(y_train, probs)
    plt.plot(fpr, tpr, label = name)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc = 'lower right')

for i, m in enumerate(best_estimator):
    roc_auc(m, model[i])

sorted_x = sorted(auc_dict.items(), key=lambda kv: kv[1], reverse=True)
sorted_x
```

[24]:
```
[('Bagging', ': ROC/AUC = 0.999999086'),
 ('Random forest', ': ROC/AUC = 0.985597897'),
 ('Boosting', ': ROC/AUC = 0.965800261'),
 ('Logistic Regression', ': ROC/AUC = 0.899099343'),
 ('Elastic Net', ': ROC/AUC = 0.890469417'),
 ('Decision tree (CART)', ': ROC/AUC = 0.879281595'),
 ('Naive Bayes', ': ROC/AUC = 0.804613537')]
```



Bagging has the highest ROC/AUC of 0.999999086.

4.Which is the best model? Defend your choice.

After comparing both error rate and ROC/AUC, Random forest is the best model. It has the second lowest error rate at 0.1978 and the second highest ROC/AUC score at 0.9856. As for other models, Bagging performs well in having the ROC/AUC, while it got the median Cross-Validated error rate. Elastic Net has the lowest Cross-Validated error rate, while it has the fifth large ROC/AUC. Actually, Boosting is the second best model, because it has the third lowest Cross-Validated error rate and the third highest ROC/AUC.

0.2.3 Evaluate the best model

5.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?

[27]:
```
rf_pred = best_estimator[-2].predict(X_test)
rf_err = 1 - accuracy_score(y_test, rf_pred)
rf_auc = roc_auc_score(y_test, rf_pred)
print("Random Forest", '\nError Rate', rf_err, '\nROC/AUC', rf_auc)
```

```
Random Forest
Error Rate 0.2089249492900609
ROC/AUC 0.7833085071168487
```

Compared with the training set result, Random Forest does not generalize well. Random Forest has a similar Cross-Validated Error rate which differs by 0.01 (testing error rate is higher). However, the testing AUC, 0.78, is much lower than that of the training set, which is around 0.99. This may indicates that the best model that I chose in 4 has the problem of overfitting.

[29]:
```
bs_pred = best_estimator[-1].predict(X_test)
bs_err = 1 - accuracy_score(y_test, bs_pred)
bs_auc = roc_auc_score(y_test, bs_pred)
print("Boosting", '\nError Rate', bs_err, '\nROC/AUC', bs_auc)
```

```
Boosting
Error Rate 0.19878296146044627
ROC/AUC 0.7939672293942404
```

Boosting is actually a better model when taking overfitting problem into considerdation.