30100HW5

February 29, 2020

Yanjie Zhou

MACS 30100

Dr. Waggoner

2020 Feb 29th

1 1

Classification error measures the accuracy of the classification, while cross entropy and Gini index measure the purity of the classfication. Thus regarding growing a decision tree, the first objective is undoubtedly accuracy, so classification error is the optimal choice in this case. When pruning the tree, we need to cut off those unnecessary leaves based on variance. Hence in this case, cross entropy and Gini index are better to use to prune those leaves that make the tree too complicated than necessary. As regards the choice between cross entropy and Gini index, considering that Gini index is somewhat more strongly peaked at equal probabilities for two classes than cross entrop, I would prefer using Gini index.

2 2

[44]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import ElasticNet, LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier,␣
↪GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from tqdm import tqdm
from sklearn.metrics import roc_auc_score, accuracy_score, roc_curve
```

[20]:
```
train = pd.read_csv('C:/Users/zyj/gss_train.csv')
test = pd.read_csv('C:/Users/zyj/gss_test.csv')
x_train = train.drop('colrac', axis=1)
y_train = train.colrac
```

```
x_test = test.drop('colrac', axis=1)
y_test = test.colrac
```

[23]:
```
model_set = [
(LogisticRegression(), {}),
(GaussianNB(), {}),
(ElasticNet(), {'alpha': np.logspace(-4, 4, 10), 'l1_ratio': [.1, .5, .7, .9, .
 ↪95, .99, 1]}),
(DecisionTreeClassifier(), {'criterion': ['gini', 'entropy'], 'max_depth':␣
 ↪range(2, 20, 2)}),
(BaggingClassifier(), {'n_estimators': range(10, 50, 5)}),
(RandomForestClassifier(), {'n_estimators': range(100, 500, 25), 'criterion':␣
 ↪['gini', 'entropy']}),
(GradientBoostingClassifier(), {'learning_rate': np.logspace(-4, -0.3, 10),
'loss': ['deviance', 'exponential'], 'n_estimators': range(100, 500, 50)})
]
best_esti = {}
best_score = {}
for model, parameters in tqdm(model_set):
    gscv = GridSearchCV(model, parameters, cv=10, refit=True, n_jobs=-1)
    gscv.fit(x_train, y_train)
    best_score[model.__class__.__name__] = gscv.best_score_
    best_esti[model.__class__.__name__] = gscv.best_estimator_
```

```
  0%|
| 0/7 [00:00<?, ?it/s]S:\Python-64\lib\site-
packages\sklearn\linear_model\_logistic.py:938: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(


 14%|
| 1/7 [00:02<00:14,  2.39s/it]

 43%|
| 3/7 [00:03<00:07,  1.84s/it]

 57%|
```

```
| 4/7 [00:04<00:04,  1.45s/it]

 71%|
| 5/7 [00:07<00:04,  2.15s/it]

 86%|
| 6/7 [00:59<00:17, 17.11s/it]

100%|
     | 7/7 [05:43<00:00, 49.09s/it]
```

[25]:
```
best_esti
```

[25]:
```
{'LogisticRegression': LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                    warm_start=False),
 'GaussianNB': GaussianNB(priors=None, var_smoothing=1e-09),
 'ElasticNet': ElasticNet(alpha=0.005994842503189409, 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),
 'DecisionTreeClassifier': 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='best'),
 'BaggingClassifier': BaggingClassifier(base_estimator=None, bootstrap=True,
bootstrap_features=False,
                   max_features=1.0, max_samples=1.0, n_estimators=30,
                   n_jobs=None, oob_score=False, random_state=None, verbose=0,
                   warm_start=False),
 'RandomForestClassifier': 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=175,
                        n_jobs=None, oob_score=False, random_state=None,
                        verbose=0, warm_start=False),
 'GradientBoostingClassifier': GradientBoostingClassifier(ccp_alpha=0.0,
```

```
criterion='friedman_mse', init=None,
                             learning_rate=0.029286445646252372,
                             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=300, n_iter_no_change=None,
                             presort='deprecated', random_state=None,
                             subsample=1.0, tol=0.0001, validation_fraction=0.1,
                             verbose=0, warm_start=False)}
```

3 3

3.1 Cross-validated error rate

[31]:
```
# The default scoring of ElasticNet is R^2, but we need accuracy
best_score['ElasticNet'] = accuracy_score(y_train, best_esti['ElasticNet'].
 ↪predict(x_train) >= 0.5)
```

[43]:
```
best_error = {n: 1-s for n, s in best_score.items()}
best_error
```

[43]:
```
{'LogisticRegression': 0.2039069681926825,
 'GaussianNB': 0.2655773120058834,
 'ElasticNet': 0.18631436314363148,
 'DecisionTreeClassifier': 0.22022890237175952,
 'BaggingClassifier': 0.21477753263467547,
 'RandomForestClassifier': 0.19447049089906232,
 'GradientBoostingClassifier': 0.19580805295091008}
```

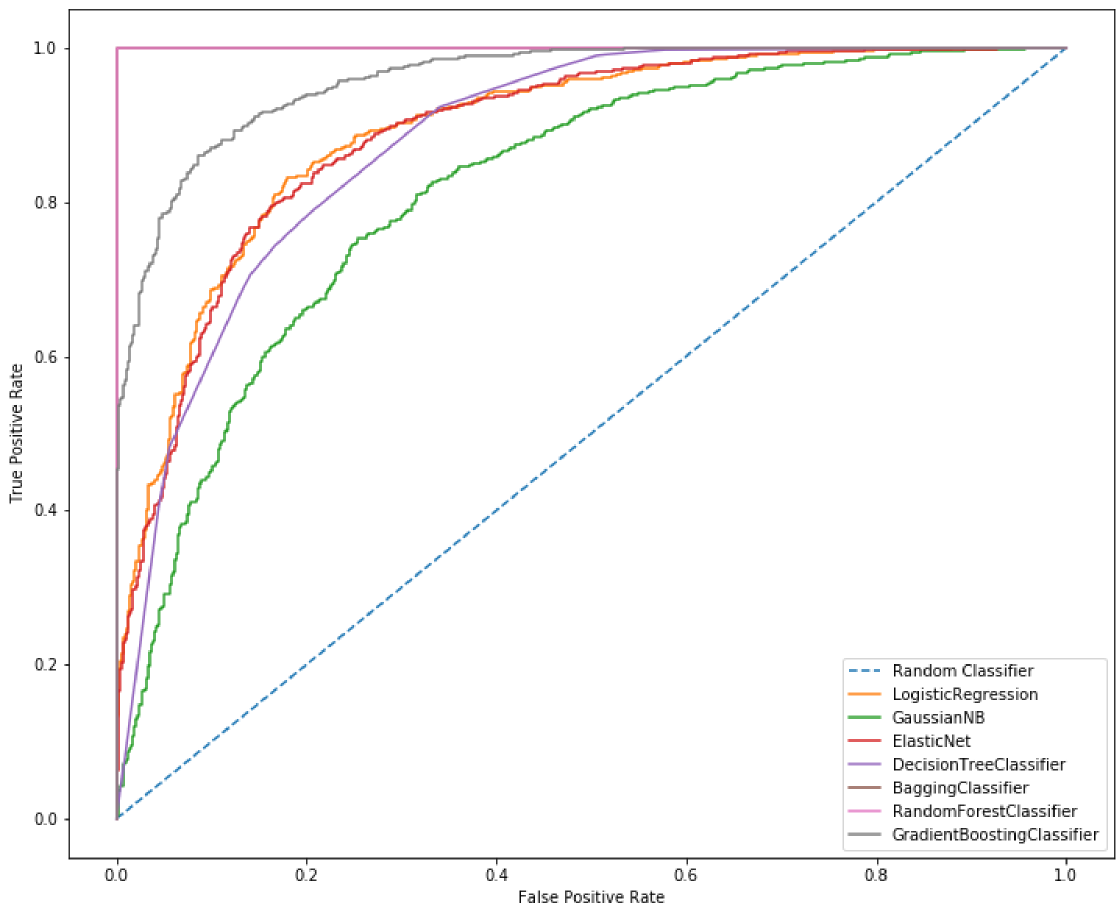
Above are the cross validated error rates for all the methods. ElasticNet makes the best performance with the cross validated error rate of 0.1863.

3.2 ROC/AUC

[36]:
```
def auc_roc(name, model):
    if name == 'ElasticNet':
        pred = model.predict(x_train)
    else:
        pred = model.predict_proba(x_train)[:, 1]
    auc = roc_auc_score(y_train, pred)
    print('AUC of '+ name + ': %f' % (auc))
    fpr, tpr, _ = roc_curve(y_train, pred)
    plt.plot(fpr, tpr, label=name)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend()
```

```
plt.figure(figsize=(12,10))
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')
for name, model in best_esti.items():
    auc_roc(name, model)
```

```
AUC of LogisticRegression: 0.893179
AUC of GaussianNB: 0.816412
AUC of ElasticNet: 0.889767
AUC of DecisionTreeClassifier: 0.879983
AUC of BaggingClassifier: 1.000000
AUC of RandomForestClassifier: 1.000000
AUC of GradientBoostingClassifier: 0.958783
```



4 4

Considering both accuracy and AUC, random forest is the best model because it has the second highest accuracy and the highest AUC. As for other models, bagging performs also pretty well in the term of AUC but it has a comparatively low accuracy. Elasticnet has the highest accuracy though it fails to achieve a high enough AUC. Gradient boosting is actually the optimal choice when taking the problem of overfitting into consideration.

5 5

[40]:
```
rf_pred = best_esti['RandomForestClassifier'].predict(x_test)
rf_accuracy = accuracy_score(y_test, rf_pred)
rf_auc = roc_auc_score(y_test, rf_pred)
print('Random forest', f'Accuracy: {rf_accuracy}', f'AUC: {rf_auc}', sep='\n')
```

```
Random forest
Accuracy: 0.7931034482758621
AUC: 0.7842767295597485
```

Compared with the result on the training set, it is obvious that the random forest classifier generalizes badly. It has the AUC of 0.7843 on the test set in contrast to 1.0000 on the training set, which suggests that this method leads to the problem of overfitting. Thus, we should instead choose the gradient boosting model, which has a good overall performance and in the meantime avoids the problem of overfitting by slowing down the learning rate of its alrogithm to 0.0293.