Deng_Yehong_HW5

March 1, 2020

Conceptual: Cost functions for classification trees 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?

Among these three possible cost functions, Gini index and cross_entropy are better to grow a decison tree. The basic algorithm for growing a decision tree is: first, start at the root node as parent node; then, split the paent node at a feature which can minimize the sum of the child node impurities; then, assign training smaple to new child nodes and stop if leave nodes are pure or early stopping criteria is satisfied.

In other words, the decision tree algorithm aims to find the feature and splitting value that leads to maximum decrease of the average child node impurities over the parent node. In comparison with the classification error, since the average child node entropy and Gini index is not equal to those of the parent node, the splitting rule would continue until the child nodes are pure. Therefore, Gini index and cross-entropy are better than classification error in growing tree. Between cross-entropy and Gini index, cross-entropy is better because its maximum value is 1 while Gini index's maximum is o.5. This suggests that cross-entropy is more sensitive to the impurity.

In order to prune the decision tree, classification error is prefered because it is less sensitive to changes in the class probabilities of the nodes.

Application: Predicting attitudes towards racist college professors

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

[35]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import  ElasticNet
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier,␣
↪GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.metrics import roc_auc_score, accuracy_score
```

[36]:
```
train = pd.read_csv('gss_train.csv')
test = pd.read_csv('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']
import warnings
warnings.filterwarnings('ignore')
```

[18]:
```
#DecisionTreeClassifier referenced from https://scikit-learn.org/stable/modules/
 ↪generated/sklearn.tree.DecisionTreeClassifier.html
#BaggingClassifier referenced from https://scikit-learn.org/stable/modules/
 ↪generated/sklearn.ensemble.BaggingClassifier.html
#RandomForestClassifier referenced from https://scikit-learn.org/stable/modules/
 ↪generated/sklearn.ensemble.RandomForestClassifier.html
#GradientBoostingClassifier referenced from https://scikit-learn.org/stable/
 ↪modules/generated/sklearn.ensemble.RandomForestClassifier.html
#GridSearchCV referenced from https://scikit-learn.org/stable/modules/generated/
 ↪sklearn.model_selection.GridSearchCV.html
models = [(LogisticRegression(), {}),
          (GaussianNB(), {}),
          (ElasticNet(), {'l1_ratio': np.linspace(0.1,1,10), 'alpha': np.
 ↪linspace(0,0.01,11)}),
          (DecisionTreeClassifier(), {'criterion': ['gini', 'entropy']}),
          (BaggingClassifier(), {'n_estimators': range(10,50,5)}),
          (RandomForestClassifier(), {'n_estimators': range(100, 500,50),␣
 ↪'criterion': ['gini', 'entropy']}),
          (GradientBoostingClassifier(), {'loss': ['deviance','exponential'],␣
 ↪'learning_rate': np.linspace(0.1,1,10)})]

best_models = []
for i in models:
    gridsearch = GridSearchCV(i[0], i[1], cv = 10, refit = True)
    gridsearch.fit(x_train, y_train)
    best_models.append(gridsearch.best_estimator_)

best_models
```

[18]:
```
[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),
 GaussianNB(priors=None, var_smoothing=1e-09),
 ElasticNet(alpha=0.009000000000000001, copy_X=True, fit_intercept=True,
            l1_ratio=0.2, max_iter=1000, normalize=False, positive=False,
```

```
            precompute=False, random_state=None, selection='cyclic', tol=0.0001,
            warm_start=False),
 DecisionTreeClassifier(class_weight=None, criterion='entropy', 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=False,
                        random_state=None, splitter='best'),
 BaggingClassifier(base_estimator=None, bootstrap=True,
bootstrap_features=False,
                   max_features=1.0, max_samples=1.0, n_estimators=25,
                   n_jobs=None, oob_score=False, random_state=None, verbose=0,
                   warm_start=False),
 RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                        max_depth=None, max_features='auto',
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=200,
                        n_jobs=None, oob_score=False, random_state=None,
                        verbose=0, warm_start=False),
 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)]
```

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

Cross-validated error rate

ROC/AUC

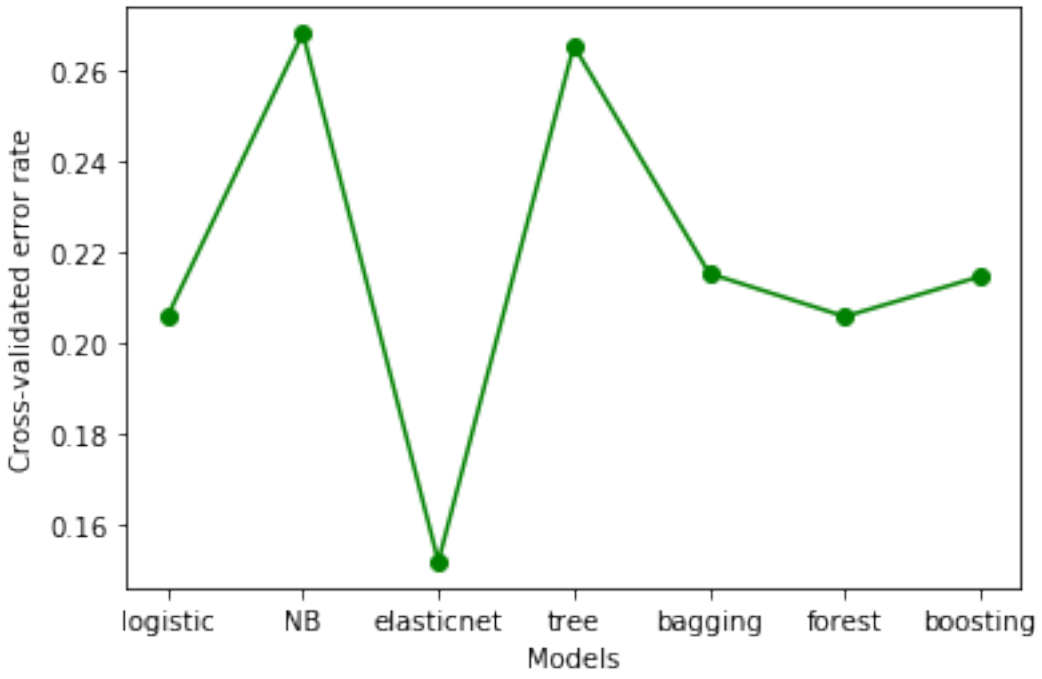
[31]:
```
md_list = ['logistic', 'NB', 'elasticnet','tree', 'bagging', 'forest',␣
 ↪'boosting']
cv_err_rate = {}
roc_auc_dic = {}
for i in range(len(best_models)):
    if i != 2:
        acc_score = np.mean(cross_val_score(best_models[i], x_train, y_train,␣
 ↪scoring = 'accuracy'))
        cv_err_rate[md_list[i]] = 1 - acc_score
    else:
```

```
        mse = -np.mean(cross_val_score(best_models[i], x_train, y_train,␣
↪scoring = 'neg_mean_squared_error'))
        cv_err_rate[md_list[i]] = mse
    roc_auc = np.mean(cross_val_score(best_models[i], x_train, y_train, scoring␣
↪= 'roc_auc'))
    roc_auc_dic[md_list[i]] = roc_auc
print(cv_err_rate)
print(roc_auc_dic)
```

```
{'logistic': 0.20594473171651984, 'NB': 0.2682672942306773, 'elasticnet':
0.152023247246076, 'tree': 0.2655434854150659, 'bagging': 0.21543535156870341,
'forest': 0.2059405921639227, 'boosting': 0.21476471605910297}
{'logistic': 0.8641335715478178, 'NB': 0.8097828942390941, 'elasticnet':
0.8659894764269868, 'tree': 0.7271669632184654, 'bagging': 0.8577849378619583,
'forest': 0.8788928851185563, 'boosting': 0.8686294725154227}
```

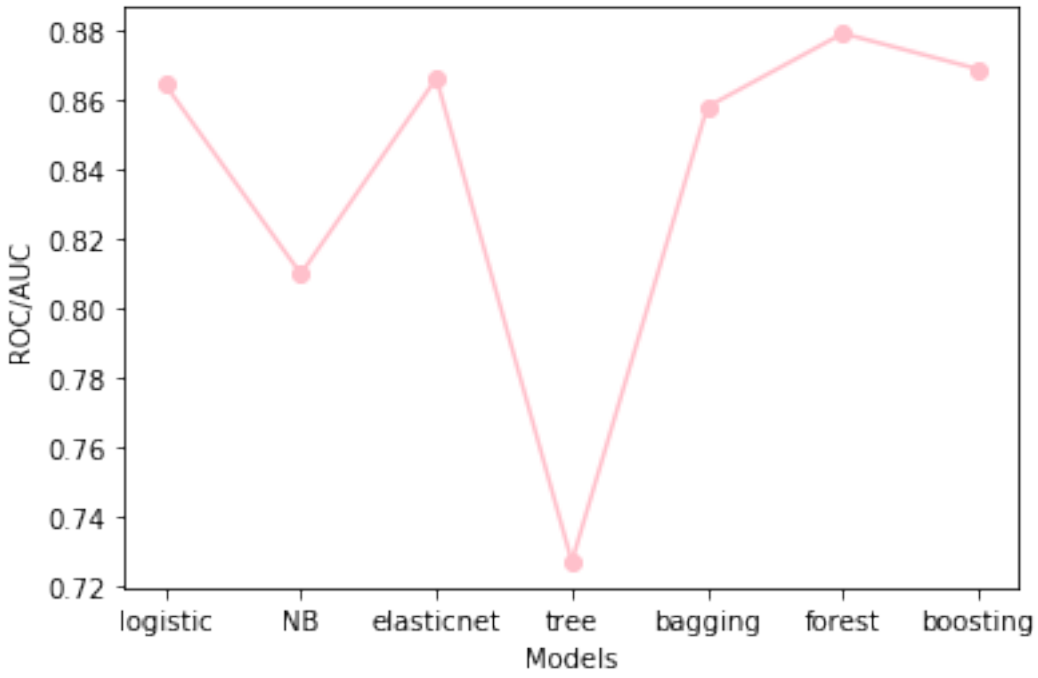
[32]:
```
plt.plot(md_list, list(cv_err_rate.values()), marker = 'o', color = 'green')
plt.xlabel('Models')
plt.ylabel('Cross-validated error rate')
```

[32]:
```
Text(0, 0.5, 'Cross-validated error rate')
```



[33]:
```
plt.plot(md_list, list(roc_auc_dic.values()), marker = 'o', color = 'pink')
plt.xlabel('Models')
plt.ylabel('ROC/AUC')
```

[33]: Text(0, 0.5, 'ROC/AUC')



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

Taking both error rate and ROC/AUC into account, the best model is the random forest, since it has the highest ROC/AUC as well as the second lowest error rate. Besides random forest, elastic net and boosting are also good models becasue they both have high ROC/AUC and low error rates.

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?

[38]:
```
forest_pred = best_models[5].predict(x_test)
error_rate = 1- accuracy_score(y_test, forest_pred)
roc_auc = roc_auc_score(y_test, forest_pred)
print("error_rate:", error_rate)
print('ROC/AUC:', roc_auc)
```

```
error_rate: 0.2068965517241379
ROC/AUC: 0.785501489572989
```

Compared to the fit evaluated on the training set in question 3-4, the random forest model did not generalize well. Although the test error rate only slightly increases for the test set, the AUC score 0.7855 for the test set is much smaller than the train AUC of 0.87889. The implication from the result could be that the model has overfitting.

6. (Up to 5 extra points) Present and substantively interpret the "best" model (selected in question 4) using PDPs/ICE curves over the range of: tolerance and age. Note, interpretation must be more than simple presentation of plots/curves. You must sufficiently describe the changes in probability estimates over the range of these two features. You may earn up to 5 extra points, where partial credit is possible if the solution is insufficient along some dimension (e.g., technically/code, interpretation, visual presentation, etc.).

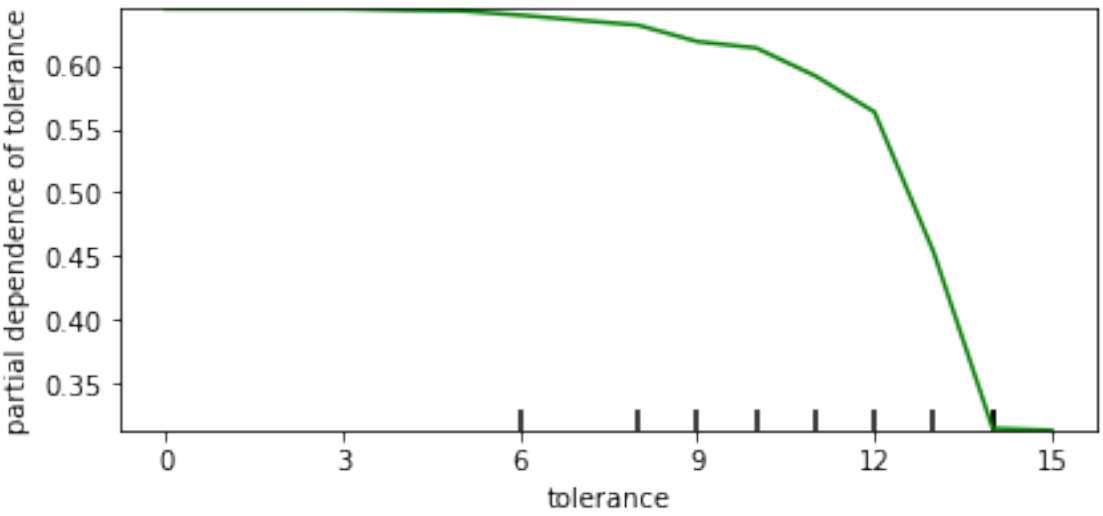
[48]:
```
#reference from https://scikit-learn.org/stable/modules/partial_dependence.html
from sklearn.inspection import plot_partial_dependence
age = x_train.columns.get_loc('age')
features = [age]
plot_partial_dependence(best_models[5], x_train, features)
plt.xlabel('age')
plt.ylabel('partial dependence of age')
```

[48]:
```
Text(0, 0.5, 'partial dependence of age')
```

The partial dependence graph for age is generated by the random forest model. It has shown that younger people have positive influence on allowing racist teacher to teach in general. More specific, people under 18 have a very high positive accociation with allowing racist teachers to teach. Such positive influence decrease drasticly for people older than 18 and, in general, the positive influence decrease for older age group, though there is some fluctuation from 30 to 60.

[47]:
```
tolerance = x_train.columns.get_loc('tolerance')
features = [tolerance]
plot_partial_dependence(best_models[5], x_train, features)
plt.xlabel('tolerance')
plt.ylabel('partial dependence of tolerance')
```

[47]:
```
Text(0, 0.5, 'partial dependence of tolerance')
```



The partial depedence graph for tolerance has shown that for people with higher tolerance score, they tend to have lower positive marginal effect for allowing racist teacher to teach.