Hu_Chun_HW5

February 29, 2020

[54]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression, ElasticNet
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier,␣
↪GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV, cross_val_score

import warnings
warnings.filterwarnings('ignore')
```

0.1 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?

Gini index and cross-entropy are frequently used impurity measures and are preferred when growing a decision tree. Between these two methods, cross-entropy is more sensitive to node impurity as it selects the split that minimizes uncertainty about the classification. Classification error is not sensitive enough for tree-growing. The tree learning algorithm is greedy, so trying to maximize classification accuracy at each step may not end up selecting the accuracy-maximizing classifier overall, and will end up fitting on noises. By contrast, doing accuracy-based pruning is less prone to overfitting because the consideration of maximizing the loss function directly is more important. In summary, `cross-entropy` is preferred when growing a tree and `classification error` is preferred when pruning a tree.

0.2 Application: Predicting attitudes towards racist college professors

0.2.1 Estimate the models

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

[2]:
```
train = pd.read_csv("gss_train.csv")
test = pd.read_csv("gss_test.csv")
```

[3]:
```
X_train = train.drop('colrac',axis=1)
y_train = train['colrac']
X_test = test.drop('colrac',axis=1)
y_test = test['colrac']
```

[20]:
```
models = {
    LogisticRegression(): {},
    GaussianNB(): {},
    ElasticNet(): {'l1_ratio': np.linspace(.1,1,11), 'alpha': np.linspace(.001,.
 ↪01,11)},
    DecisionTreeClassifier(): {'criterion': ['gini', 'entropy']},
    BaggingClassifier(): {'n_estimators': range(10,50,5)},
    RandomForestClassifier(): {'n_estimators': range(100,300,25), 'criterion':␣
 ↪['gini', 'entropy']},
    GradientBoostingClassifier(): {'loss': ['deviance', 'exponential'],␣
 ↪'learning_rate': np.linspace(.1,1,11)}
}

best_estimator = []
best_score = []
for model, param in models.items():
    gscv = GridSearchCV(model, param, cv=10, refit=True)
    gscv.fit(X_train, y_train)
    print(gscv.best_estimator_)
    best_estimator.append(gscv.best_estimator_)
    best_score.append(gscv.best_score_)
```

```
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=0.0091, copy_X=True, fit_intercept=True, l1_ratio=0.19,
           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='gini', 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='entropy',
                       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)
```

0.2.2 Evaluate the models

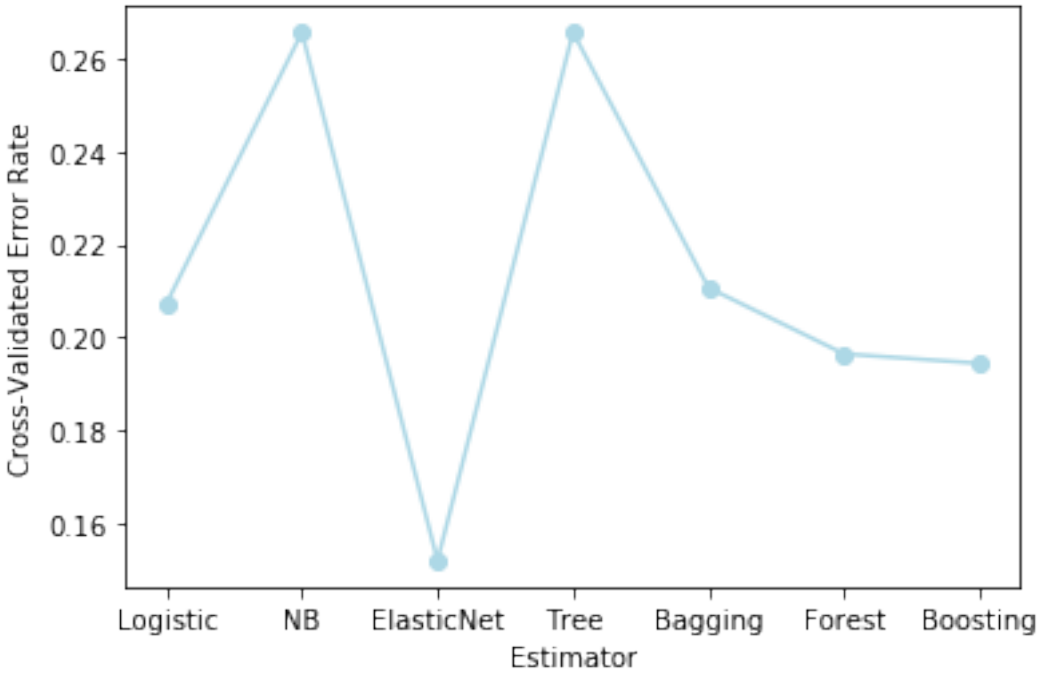
3.(20 points) Compare and present each model's (training) performance based on `Cross-validated error rate` and `ROC/AUC`.

[47]:
```
model = ['Logistic', 'NB', 'ElasticNet', 'Tree', 'Bagging', 'Forest',␣
 ↪'Boosting']
error_rate = {}
for i in range(len(best_score)):
    error_rate[model[i]] = 1- best_score[i]
# use mse for elastic net regression
error_rate['ElasticNet'] = -np.mean(cross_val_score(best_estimator[2], X_train,␣
 ↪y_train, scoring='neg_mean_squared_error'))
error_rate
```

[47]:
```
{'Logistic': 0.20731707317073167,
 'NB': 0.26558265582655827,
 'ElasticNet': 0.15206008123118034,
 'Tree': 0.26558265582655827,
 'Bagging': 0.2107046070460704,
 'Forest': 0.1964769647696477,
 'Boosting': 0.19444444444444442}
```

[48]:
```
plt.plot(model, list(error_rate.values()), marker='o', color='lightblue')
plt.xlabel('Estimator')
plt.ylabel('Cross-Validated Error Rate')
```

[48]:
```
Text(0, 0.5, 'Cross-Validated Error Rate')
```



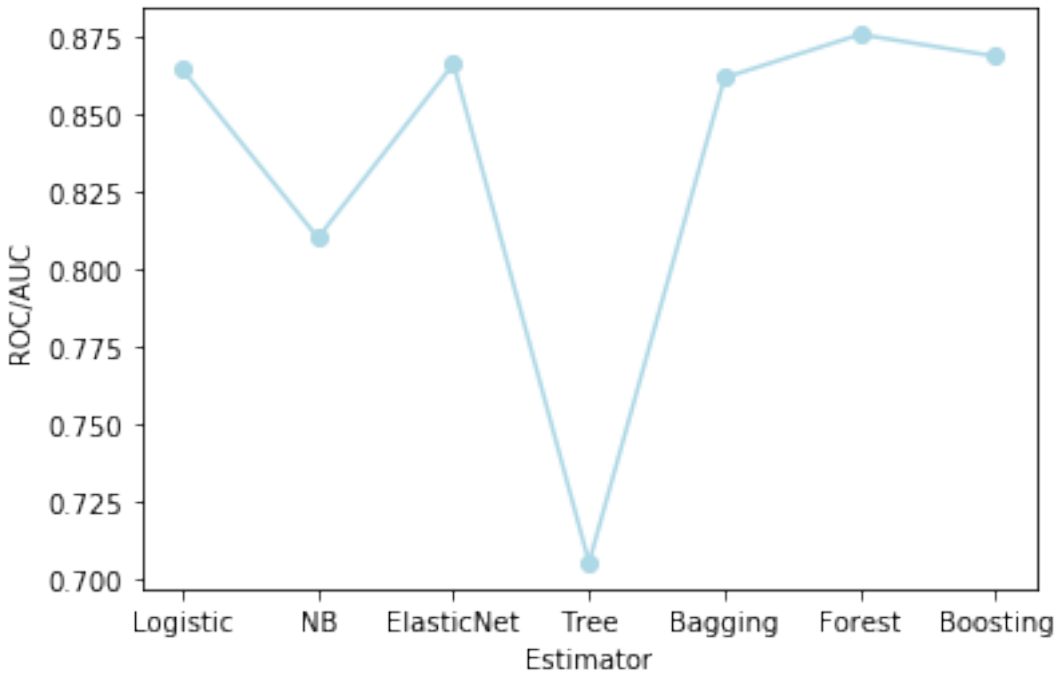
Elastic Net has the lowest CV error rate at 0.152.

[50]:
```
roc_auc = {}
for i in range(len(best_estimator)):
    roc_auc[model[i]] = np.mean(cross_val_score(best_estimator[i], X_train,␣
 ↪y_train, scoring='roc_auc'))
roc_auc
```

[50]:
```
{'Logistic': 0.8641169601844321,
 'NB': 0.8097828942390941,
 'ElasticNet': 0.8659619990531406,
 'Tree': 0.7049850213589108,
 'Bagging': 0.8614632734513293,
 'Forest': 0.8754504785296282,
 'Boosting': 0.8683684206108268}
```

[51]:
```
plt.plot(model, list(roc_auc.values()), marker='o', color='lightblue')
plt.xlabel('Estimator')
plt.ylabel('ROC/AUC')
```

[51]:
```
Text(0, 0.5, 'ROC/AUC')
```



Random Forest has the highest ROC/AUC score at 0.875.

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

Taking into account both error rate and ROC/AUC, `Gradient Boosting` is the best model. It has the second lowest error rate at 0.194 and the second highest AUC score at 0.868. Elastic Net and Random Forest are good models as well. They both have low scores in error rate and high scores in AUC.

0.2.3 Evaluate the best model

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?

[58]:
```
boosting_pred = best_estimator[-1].predict(X_test)
boosting_error_rate = 1 - np.mean(cross_val_score(best_estimator[-1], X_train,␣
 ↪y_train, scoring='accuracy'))
boosting_auc = np.mean(cross_val_score(best_estimator[-1], X_train, y_train,␣
 ↪scoring='roc_auc'))
print('Error Rate', boosting_error_rate, '\nAUC', boosting_auc)
```

```
Error Rate 0.21476471605910297
AUC 0.8683079276867746
```

Compared to the fit evaluated on the training set, `Gradient Boosting` generalizes pretty well. The error rate is a little bit higher than the training error, but since the training set experienced 10-

fold cv, the difference in error rate between training and testing performance is small. The AUC score is similar as well.

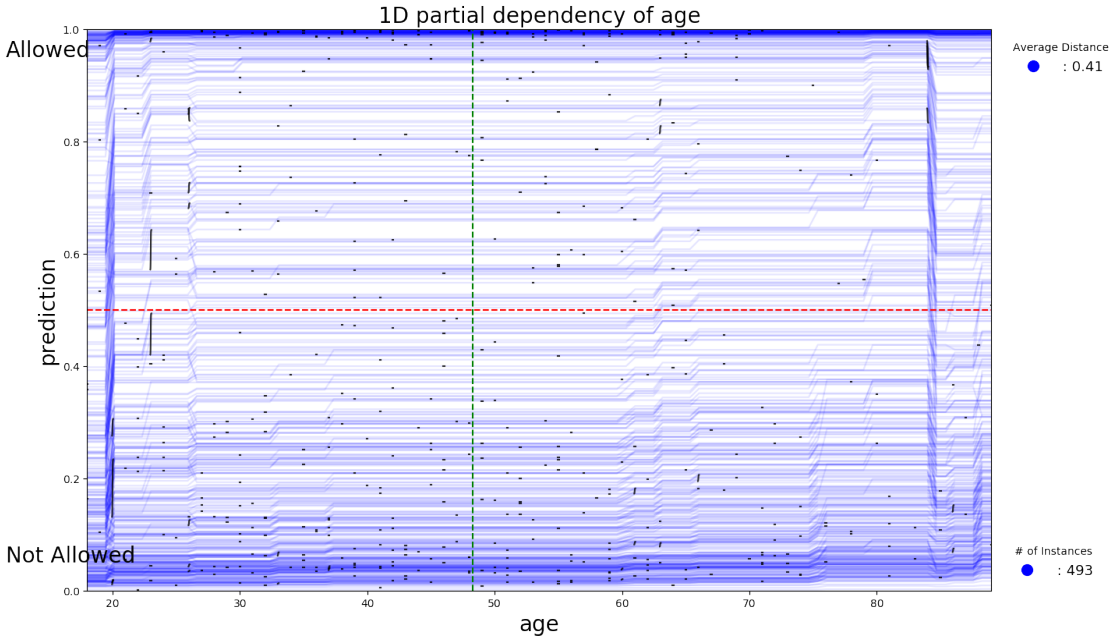
0.2.4 Bonus: PDPs/ICE

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.).

[65]:
```
import partial_dependence as pdp_plot
age = X_train.columns.get_loc('age')
tol = X_train.columns.get_loc('tolerance')
```

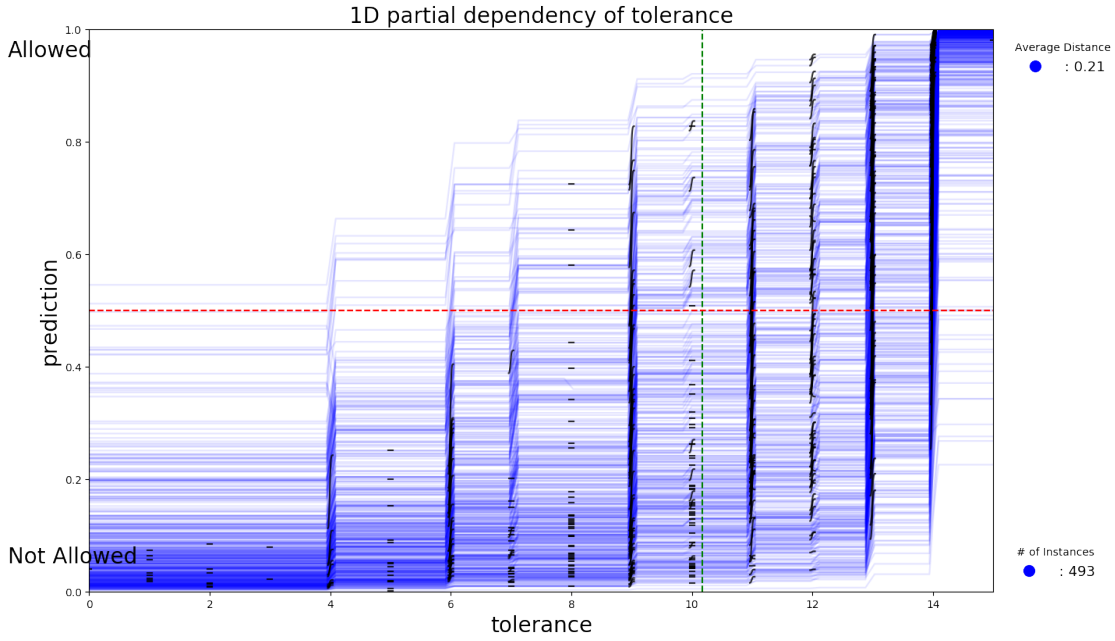
[100]:
```
pred_data = best_estimator[-1].predict_proba(X_test)
pred_df = pd.DataFrame({'Allowed': pred_data[:, 0], 'Not Allowed': pred_data[:,␣
↪1]})
```

[103]:
```
# partial dependency of age
age_pdp_plot = pdp_plot.PartialDependence(X_test, best_estimator[-1], pred_df.
↪columns.tolist(), pred_df.columns.tolist()[0])
curves = age_pdp_plot.pdp(X_test.columns[age])
age_pdp_plot.plot(curves, local_curves=True, plot_full_curves=True)
```



[104]:
```
# partial dependency of tolerance
tol_pdp_plot = pdp_plot.PartialDependence(X_test, best_estimator[-1], pred_df.
↪columns.tolist(), pred_df.columns.tolist()[0])
```

```
curves = tol_pdp_plot.pdp(X_test.columns[tol])
tol_pdp_plot.plot(curves, local_curves=True, plot_full_curves=True)
```



I used `Gradient Boosting` to predict the outcomes and visualized the marginal effect of the two features in simple PDP plots. From age's PDP plot, we can see that `age` does not have a strong influence on `colrac`. At age intervals around 20-25 and 60-65, there are slight fluctuations in the probability estimates but in general the shape of PDP plot is relatively smooth. On the other hand, as `tolerance` level increases, the probability of the racist professor allowed to teach significantly increases. The increase is consistent - no outlier detected. The sub-region devisions are clear as well as a result of the nature of decision tree, and so we can how the algorithm decides the splits.