HW05_Aabir_AK

March 1, 2020

0.1 Homework 5 - Aabir Abubaker Kar

[35]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression, ElasticNet, ElasticNetCV
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_auc_score, accuracy_score, roc_curve
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, GradientBoostingClassifier,␣
 ↪RandomForestClassifier, GradientBoostingClassifier
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.exceptions import ConvergenceWarning
from sklearn.inspection import plot_partial_dependence

import seaborn as sns

import tabulate as tb

import warnings
warnings.filterwarnings("ignore", category=ConvergenceWarning)

sns.set_style('whitegrid')

np.random.seed(42)
```

0.1.1 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?

The three loss functions are conceptually different in the outcomes achieved.

Classification error examines the net accuracy over the entire prediction. Here, each observation in the training or test set is treated as an independent important contributor to the estimation of error. This is the naive error that we have been using commonly all this time - the RMSE for regression, and the accuracy for classification, are commonly used forms.

The Gini index optimizes for the variance across all classes. This means that it tries to reduce the overall 'inequality' in the variances, making sure to even out the disparities between predictions of one class over another. This is a way of ensuring that say, we do not produce a decision tree that predicts success with 90% accuracy, but failure with only 50% accuracy.

The cross entropy examines the difference between the distribution of outcomes in the data (usually the test set) and that generated by the classification tree. This is a mathematical optimization on the 'distance' between probability distributions. Rather than considering 'inequality' like the Gini index, this is a raw metric of how two probability distributions differ.

By this intuition, it would appear that we should make the following choices: 1. For *growing* a decision tree:

Use the classification error. The baseline for making new decisions should be correctness, rather than a macroscopic view of the 'big picture'. This is in line with the greedy search that decision trees are fundamentally based on - we are not trying to find a global optimum, rather, the most optimal next step. 2. For *pruning* a decision tree:

At this point, it makes sense to use either the Gini index or the cross entropy to ensure that our incremental progress is matching up to ground truths. The Gini index would be useful for classification where there are different numbers of samples for each category. Cases using stratification would naturally have to take care of this case. However, if there are approximately equal numbers of observations for each category, the cross entropy is the most robust way of making sure that the distribution generated by the decision tree matches that of the data.

[18]:
```python
# load data
train = pd.read_csv('./data/gss_train.csv')
test = pd.read_csv('./data/gss_test.csv')
x_train, y_train = train.drop('colrac', axis=1), train.colrac
x_test, y_test = test.drop('colrac', axis=1), test.colrac

x_train, x_test = [StandardScaler().fit_transform(i) for i in (x_train, x_test)]
```

[19]:
```python
modelnames = ["Logistic regression", "Naive Bayes", "Elastic net regression",
"Decision tree (CART)", "Bagging", "Random forest", "Boosting"]

all_models = [(LogisticRegression(), {'C': np.logspace(-6, 1.5, 10)}),
                (GaussianNB(), {}),
                (ElasticNet(), {'alpha': np.logspace(-6, 1.5, 10),
                                'l1_ratio': [.1, .5, .7, .9, .95, .99, 1]}),
                (DecisionTreeClassifier(), {'criterion': ['gini', 'entropy'],
                                            'max_depth':range(2, 21, 2)}),
                (BaggingClassifier(), {'n_estimators': range(10, 21, 2)}),
                (RandomForestClassifier(), {'n_estimators': range(50, 250, 25),
                                            'criterion': ['gini', 'entropy']}),
                (GradientBoostingClassifier(), {'learning_rate': np.
↪logspace(-6, 0.5, 10),
                                                  'loss': ['deviance',␣
↪'exponential'],
                                                  'n_estimators': range(50, 250,␣
↪25)})
            ]

best_model = {}
score = {}

for i, (model, parameters) in enumerate(all_models):
    gscv = GridSearchCV(model, parameters, cv=10)
    gscv.fit(x_train, y_train)
    score[model.__class__.__name__] = gscv.best_score_
    best_model[model.__class__.__name__] = gscv.best_estimator_
```

```
For model LogisticRegression: best score = 0.7974443831586688
For model GaussianNB: best score = 0.7344226879941166
For model ElasticNet: best score = 0.40466869797838684
For model DecisionTreeClassifier: best score = 0.7804467733039162
For model BaggingClassifier: best score = 0.7811224489795918
For model RandomForestClassifier: best score = 0.8055570876999448
For model GradientBoostingClassifier: best score = 0.804853833425262
```

[46]:
```
# elasticnet is a regression and its default error is MSE - we therefore␣
↪recompute the error-rate using
score['ElasticNet'] = accuracy_score(y_train, np.
↪greater_equal(best_models['ElasticNet'].predict(x_train), 0.5))
```

[27]:
```
for k, v in best_model.items():
    try:
        print(f"\n-----------------------------------\n\nFor model: {k}, best␣
↪parameters are:\n\n{v.get_params()}")
    except AttributeError:
        pass
```

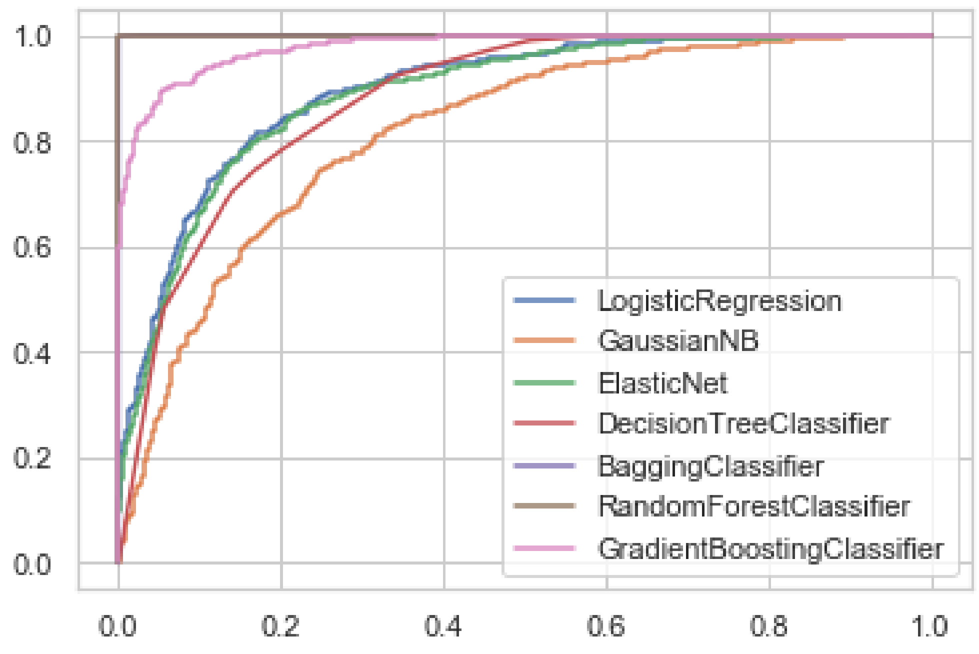
```
-----------------------------------

For model: LogisticRegression, best parameters are:

{'C': 0.1, '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}
```

```
-----------------------------------

For model: GaussianNB, best parameters are:

{'priors': None, 'var_smoothing': 1e-09}

-----------------------------------

For model: ElasticNet, best parameters are:

{'alpha': 0.014677992676220705, '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}

-----------------------------------

For model: DecisionTreeClassifier, best parameters are:

{'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'}

-----------------------------------

For model: BaggingClassifier, best parameters are:

{'base_estimator': None, 'bootstrap': True, 'bootstrap_features': False,
'max_features': 1.0, 'max_samples': 1.0, 'n_estimators': 18, 'n_jobs': None,
'oob_score': False, 'random_state': None, 'verbose': 0, 'warm_start': False}

-----------------------------------

For model: RandomForestClassifier, best parameters are:

{'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': 50, 'n_jobs': None, 'oob_score': False, 'random_state': None,
'verbose': 0, 'warm_start': False}

-----------------------------------

For model: GradientBoostingClassifier, best parameters are:
```

```
{'ccp_alpha': 0.0, 'criterion': 'friedman_mse', 'init': None, 'learning_rate':
0.11364636663857243, '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': 125, '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}
```

[51]:
```
roc_scores = []

for name, model in best_model.items():
    if name == 'ElasticNet':
        y_pred = model.predict(x_train)
    else:
        y_pred = model.predict_proba(x_train)[:, 1]
    roc_scores.append(roc_auc_score(y_train, y_pred))
    fpr, tpr, thresholds = roc_curve(y_train, y_pred)
    plt.plot(fpr, tpr, label=name)

plt.legend()

a, b, c = list(score.keys()), list(score.values()), roc_scores

print(tb.tabulate([[a[i], b[i], c[i]] for i in range(len(a))],␣
 ↪headers=['Model', 'Train Accuracy', 'Train ROC/AUC']))
```

```
Model                           Train Accuracy    Train ROC/AUC
--------------------------  ----------------  ---------------
LogisticRegression                  0.797444         0.896191
GaussianNB                          0.734423         0.816412
ElasticNet                          0.751355         0.888808
DecisionTreeClassifier              0.780447         0.879983
BaggingClassifier                   0.781122         0.999978
RandomForestClassifier              0.805557         1
GradientBoostingClassifier          0.804854         0.978212
```

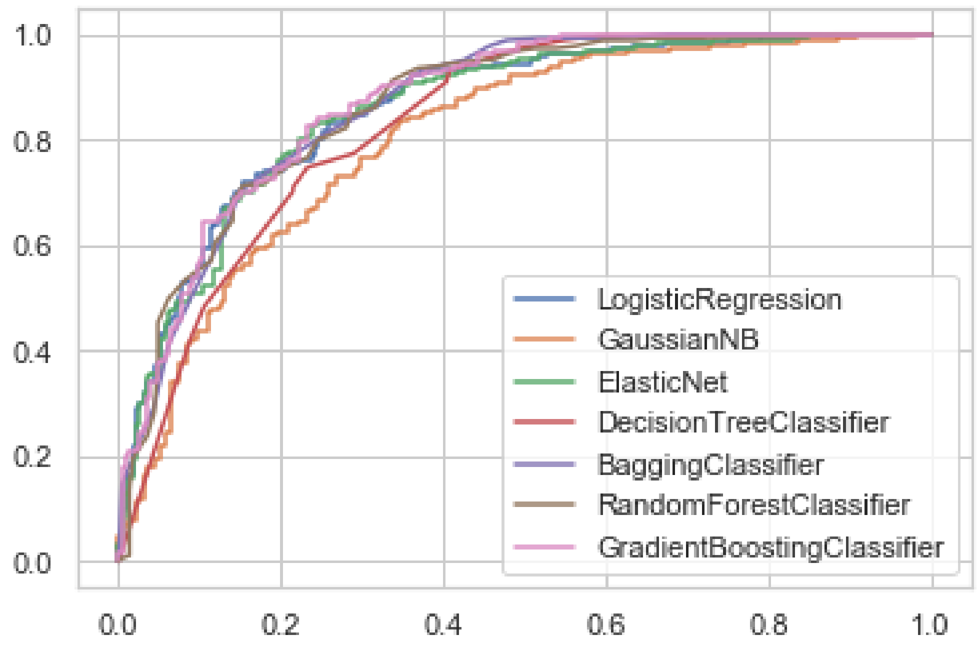


The performance metrics seem to indicate that RandomForest is the best model for the job. However, the ROC curves tell us that the RandomForest model could be overfitting. It looks like the RandomForest has an unrealistically 'perfect' prediction for each sample in the training set. Validating this on the test set below shows that this is true - on the test data, the GradientBoostingClassifier does better than any other model, with BaggingClassifier also exceeding RandomForest in terms of accuracy.

GradientBoostingClassifier is therefore the hands-down best model for this task. As intuitively expected, complicated decision boundary with non-linearities is probably better modeled by an ensemble with Gradient Boosting than by assuming a linear relationship. This is also probably because of the finite number of iterations to train the model. Training until convergence may have different results.

[52]:
```
roc_scores = []
test_acc = []

for name, model in best_model.items():
    if name == 'ElasticNet':
        y_pred = model.predict(x_test)
    else:
        y_pred = model.predict_proba(x_test)[:, 1]
    test_acc.append(accuracy_score(y_test, y_pred.round(), normalize=False)/
 ↪y_train.shape[0]*2)
    roc_scores.append(roc_auc_score(y_test, y_pred))
    fpr, tpr, thresholds = roc_curve(y_test, y_pred)
    plt.plot(fpr, tpr, label=name)
plt.legend()

a2, c2 = list(score.keys()), roc_scores

print(tb.tabulate([[a2[i], b[i], test_acc[i], c[i], c2[i]] for i in␣
 ↪range(len(a))],
                  headers=['Model', 'Train Accuracy', 'Test Accuracy', 'Train␣
 ↪ROC/AUC', ' Test ROC/AUC']))
```

```
Model                          Train Accuracy    Test Accuracy    Train ROC/AUC
Test ROC/AUC
--------------------------  ----------------  ---------------  ---------------
---------------
LogisticRegression                  0.797444         0.51897          0.896191
0.860824
GaussianNB                          0.734423         0.487805         0.816412
0.805263
ElasticNet                          0.751355         0.53252          0.888808
0.858077
DecisionTreeClassifier              0.780447         0.51084          0.879983
0.832522
BaggingClassifier                   0.781122         0.520325         0.999978
0.866907
RandomForestClassifier              0.805557         0.52439          1
0.863936
GradientBoostingClassifier          0.804854         0.5271           0.978212
0.871168
```

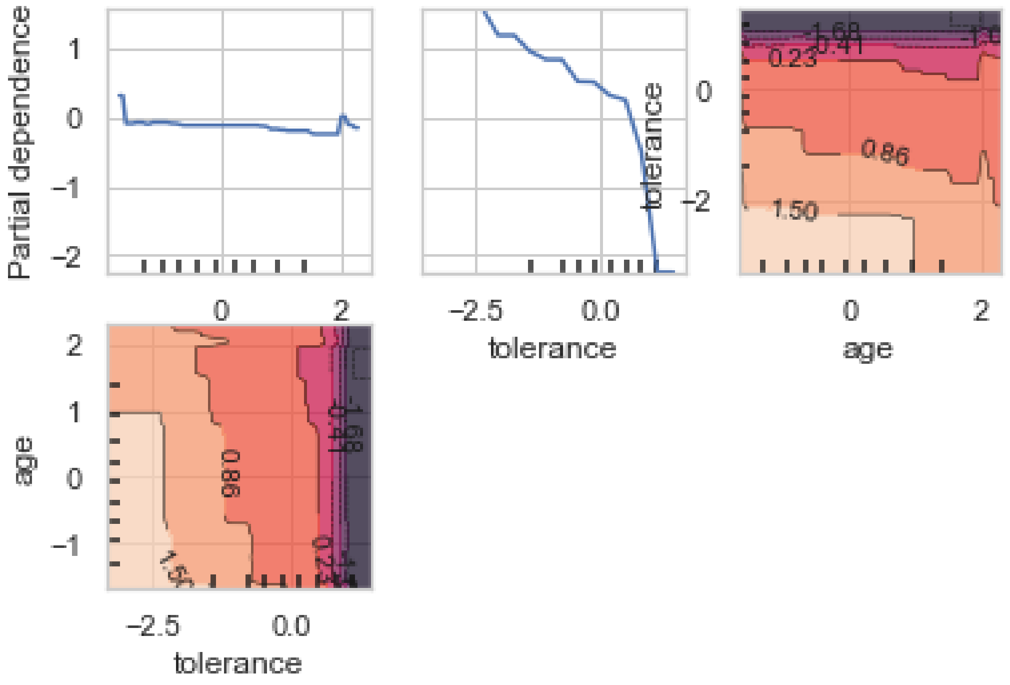


As seen above, the test set also shows better performance by the GradientBoostingClassifier. Note that suspiciously low test accuracy for each of these models. It is at least heartening that the GradientBooster's ROC/AUC is a high of 0.87. This indicates that it was not overfitting as much as RandomForest probably was.

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.).

[80]:
```
gbmodel = best_model['GradientBoostingClassifier']

feature_names = [i for i in train.columns if i!='colrac']

plot_partial_dependence(gbmodel, x_train, ['age', 'tolerance', ('age',␣
↪'tolerance'), ('tolerance','age')], feature_names=feature_names)

plt.show()
```



From the PDPs, we see that there is not a strong relationship between the model marginal and the `age` variable. This may be surprising, as it indicates that for each age group, if we marginalize the distribution, we don't see much of a variation in the `colrac` variable. Another interpretation of this is that the probability distribution does not shift much as we take the marginal distribution for different values of `age`.

On the other hand, the `tolerance` variable seems to be strongly predictive of the `colrac` variable with a strong negative correlation. As we compute the marginal distribution over `tolerance`, we see negative values of `colrac` as the tolerance increases. This implies that more tolerant people gave lower answers to the `colrac` question - which is tantamount to them generally answering that racist college professors should *not* be allowed to teach.

This can be interpreted as the probability of higher values of `colrac` decreasing significantly (a negative dependence) as we encounter higher values of `tolerance`.