Hu_Anqi_HW5

March 1, 2020

0.1 Problem 1

Classification error rate is the fraction of the training observations under an attribute that do not belong in the most common class. Gini index is a measure of node purity that evaluates the total variance across all classes. Cross entropy is a measure of the difference between two probability distributions for a given random variable. It calculates the number of bits required to transmit an average event between them. For growing a decision tree, Gini and cross entropy are both good measures as they are more sensitive to node purity, thus resulting in higher accuracies. Between these two, entropy is even more sensitive and thus more preferred. On the other hand, classification error is preferred when pruning the decision tree, as it is less prone to overfitting.

0.2 Problem 2

[1]:
```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import cross_validate, GridSearchCV,␣
↪cross_val_score
from sklearn.linear_model import LogisticRegression, ElasticNet
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier,␣
↪GradientBoostingClassifier
from sklearn.metrics import accuracy_score, mean_squared_error
import partial_dependence
```

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

[3]:
```
y = train['colrac']
x = train.loc[:, train.columns != 'colrac']
test_x = test.loc[:, train.columns != 'colrac']
test_y = test['colrac']

acc_scores = []
```

```
# logistic regression
logit = LogisticRegression(solver='liblinear')
param_grid = {}
grid = GridSearchCV(logit, param_grid=param_grid, scoring='accuracy', cv=10)
fit = grid.fit(x, y)
acc_scores.append(fit.best_score_)
logit_best = fit.best_estimator_
```

[4]:
```
# naive bayes
nb = GaussianNB().fit(x, y)
param_grid = {}
grid = GridSearchCV(logit, param_grid=param_grid, scoring='accuracy', cv=10)
fit = grid.fit(x, y)
acc_scores.append(fit.best_score_)
nb_best = fit.best_estimator_
```

[5]:
```
# elastic net
enet = ElasticNet()
parametersGrid = {"alpha": np.linspace(0.001, 0.01, 11),
                  "l1_ratio": np.arange(0.0, 1.0, 0.1)}
grid = GridSearchCV(enet, parametersGrid, cv=10, refit=True)
fit = grid.fit(x, y)
acc_scores.append(fit.best_score_)
enet_best = fit.best_estimator_
```

[6]:
```
# CART
cart = tree.DecisionTreeClassifier().fit(x, y)
param_grid = {'criterion': ['gini', 'entropy']}
grid = GridSearchCV(cart, param_grid, scoring = 'accuracy', cv=10)
fit = grid.fit(x, y)
acc_scores.append(fit.best_score_)
cart_best = fit.best_estimator_
```

[7]:
```
# bagging
bag = BaggingClassifier()
param_grid = {'n_estimators': np.arange(10, 50, 10)}
grid = GridSearchCV(bag, param_grid, scoring = 'accuracy', cv=10)
fit = grid.fit(x, y)
acc_scores.append(fit.best_score_)
bag_best = fit.best_estimator_
```

[8]:
```
# random forest
rf = RandomForestClassifier()
param_grid = {'n_estimators': np.arange(100, 200, 10),
              'criterion': ['gini', 'entropy']}
grid = GridSearchCV(rf, param_grid, scoring = 'accuracy', cv=10)
rf_fit = grid.fit(x, y)
```

```
acc_scores.append(rf_fit.best_score_)
rf_best = rf_fit.best_estimator_
```

[9]:
```
# boosting
boost = GradientBoostingClassifier()
param_grid = {'loss': ['deviance', 'exponential'],
              'learning_rate': np.arange(0.1, 1, 0.1)}
grid = GridSearchCV(boost, param_grid, scoring = 'accuracy', cv=10)
fit = grid.fit(x, y)
acc_scores.append(fit.best_score_)
boost_best = fit.best_estimator_
```

0.3 Problem 3

[11]:
```
error_rate = pd.DataFrame(columns=['model', 'error_rate'])
error_rate['model'] = ['logit', 'nb', 'enet', 'cart', 'bag', 'randomf', 'boost']
best_models = [logit_best, nb_best, enet_best, cart_best, bag_best, rf_best,␣
 ↪boost_best]

for i in range(len(best_models)):
    error_rate['error_rate'][i] = 1 - acc_scores[i]

error_rate['error_rate'][2] = -np.mean(cross_val_score(enet_best, x, y,␣
 ↪scoring='neg_mean_squared_error'))
error_rate.sort_values(by=['error_rate'], inplace=True)
error_rate.reset_index(drop=True)
```

[11]:

	model	error_rate
0	enet	0.152008
1	randomf	0.193767
2	boost	0.194444
3	logit	0.207317
4	nb	0.207317
5	bag	0.214092
6	cart	0.267615

[12]:
```
roc_auc = pd.DataFrame(columns=['model', 'roc_auc'])
roc_auc['model'] = ['logit', 'nb', 'enet', 'cart', 'bag', 'randomf', 'boost']
for i in range(len(best_models)):
    roc_auc['roc_auc'][i] = np.mean(cross_val_score(best_models[i], x, y,␣
 ↪scoring='roc_auc'))
roc_auc.sort_values(by=['roc_auc'], ascending=False, inplace=True)
roc_auc.reset_index(drop=True)
```

[12]:

	model	roc_auc
0	randomf	0.877124
1	boost	0.868181
2	enet	0.866006
3	logit	0.864117
4	nb	0.864117
5	bag	0.853497
6	cart	0.72294

0.4 Problem 4

Judging by error rate, elastic net is the best performing model. By roc_auc score, random forest has the highest performance. Combining the two measures, the best performing model should be random forest, since it ranks second in error rate and first in roc_auc.

0.5 Problem 5

[13]:
```
# Cross-validated error rate
rf_pred_score = 1 - np.mean(cross_val_score(rf_best, test_x, test_y, cv=10,␣
 ↪scoring='accuracy'))
rf_pred_score
```

[13]: 0.20709863945578233

[14]:
```
# roc_auc
rf_roc_auc = np.mean(cross_val_score(rf_best, x, y, scoring='roc_auc'))
rf_roc_auc
```

[14]: 0.8818656163159938

The classification error rate of the randomforest model is 0.207, and the roc_auc score is 0.882. In the test performance, the error rate for the same model was 0.194; the roc_auc score was 0.877. Roughly speaking, these scores are not significantly different in the testing and training performances, meaning that this model generalizes pretty well.

0.6 Problem 6

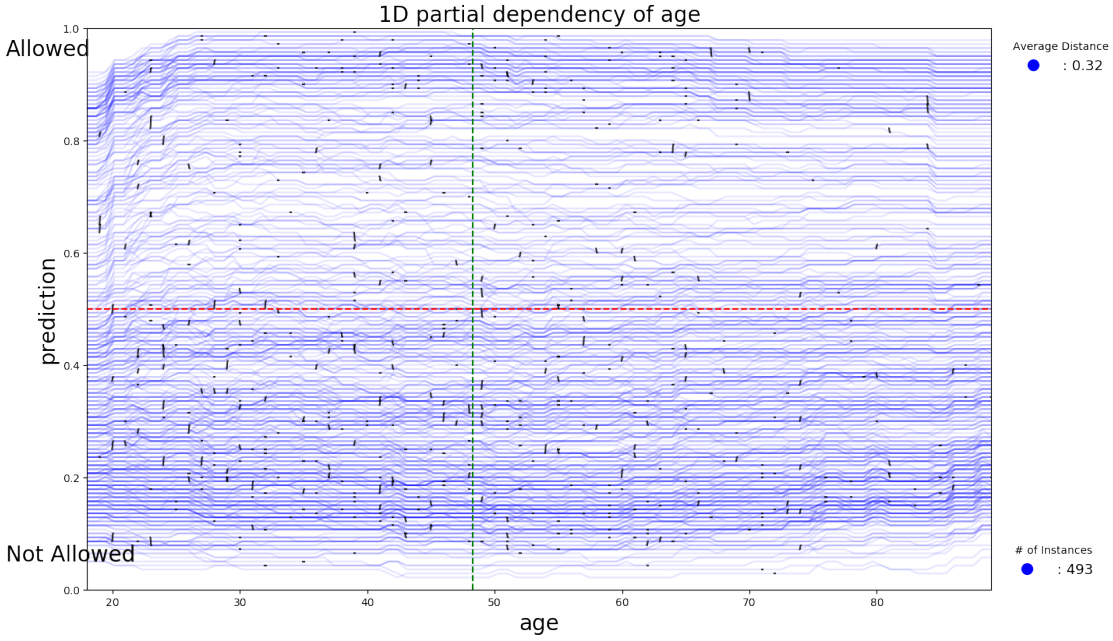
[15]:
```
# PDPs/ICE
a = x.columns.get_loc('age')
t = x.columns.get_loc('tolerance')

# for age
prediction = rf_best.predict_proba(test_x)
pred = pd.DataFrame(columns=['Allowed', 'Not Allowed'])
```

```
pred['Allowed'] = prediction[:, 0]
pred['Not Allowed'] = prediction[:, 1]

age = partial_dependence.PartialDependence(test_x,
                                           rf_best,
                                           pred.columns.tolist(),
                                           pred.columns.tolist()[0])

age.plot(age.pdp(x.columns[a]), local_curves=True, plot_full_curves=True)
```

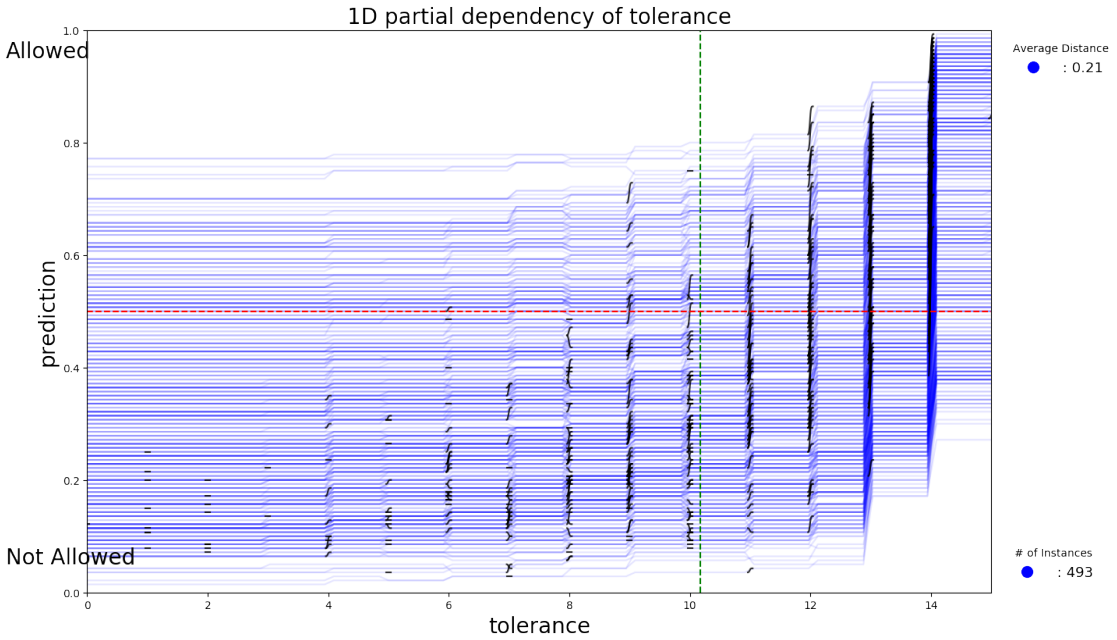


[16]:
```
# for tolerance
prediction = rf_best.predict_proba(test_x)
pred = pd.DataFrame(columns=['Allowed', 'Not Allowed'])

pred['Allowed'] = prediction[:, 0]
pred['Not Allowed'] = prediction[:, 1]

tolerance = partial_dependence.PartialDependence(test_x,
                                                 rf_best,
                                                 pred.columns.tolist(),
                                                 pred.columns.tolist()[0])

tolerance.plot(tolerance.pdp(x.columns[t]), local_curves=True,␣
↪plot_full_curves=True)
```



With the partial dependence plots using random forest, we can see that the prediction probabilities do not vary much as age increases. When age is closer to the minimum, the predicted probabilities aggregate within a narrower interval of values, where as with an increase in age, the variance increases slightly. With tolerance, we see that the probability of being predicted as allowed or not allowed increases with one's tolerance. For the tolerance levels 0 through 7, the predicted probabilities are very similar. However, starting at 7 and onward, the increase in predicted probability becomes steadier and more consistent. Depending on what this `tolerence` represents, it might be a lot more tightly related to a tolerance towards racism, as opposed to `age`.