HW5

March 1, 2020

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[46]:
```
# ml learning package
from sklearn.linear_model import LogisticRegression, ElasticNet, ElasticNetCV
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
from sklearn.metrics import roc_auc_score, accuracy_score, roc_curve
from sklearn.inspection import plot_partial_dependence
```

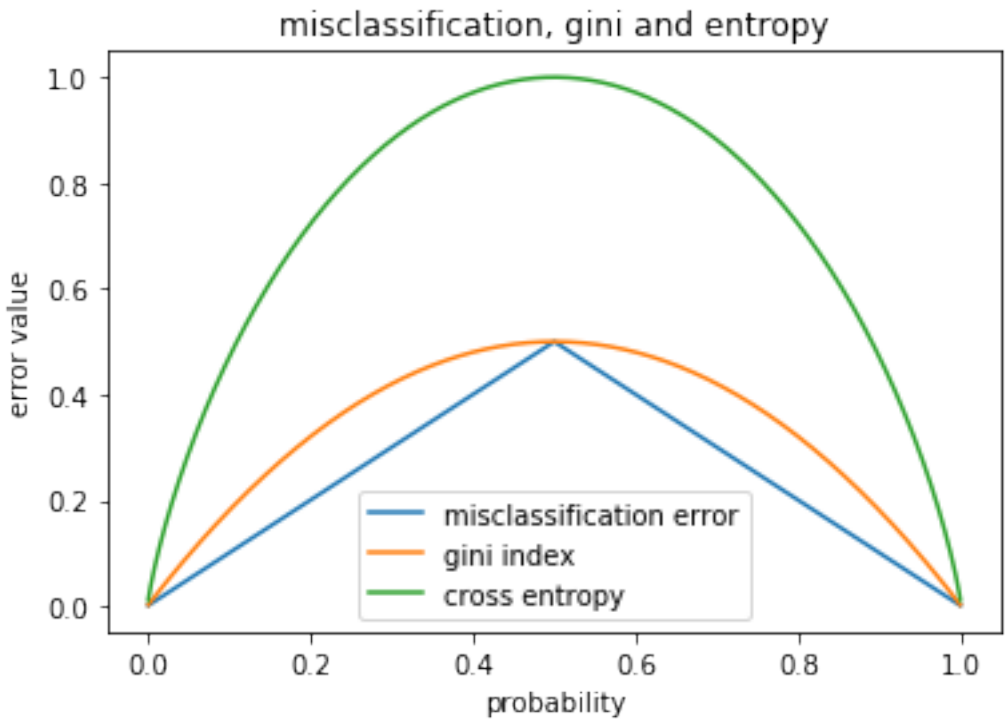
[47]:
```
import warnings
import tqdm
from pdpbox import info_plots,pdp
warnings.filterwarnings("ignore", category=FutureWarning)
```

1 Conceptual: Cost functions for classification trees.

[4]:
```
p=np.linspace(0,1,1000)[1:-1]
class_err=[1- max(i,1-i) for i in p]
gini_err=[2*i*(1-i) for i in p]
entropy_err=[-(i*np.log2(i)+(1-i)*np.log2(1-i)) for i in p]

#draw graph
plt.plot(p, class_err, label="misclassification error")
plt.plot(p, gini_err, label="gini index")
plt.plot(p, entropy_err, label="cross entropy")
plt.xlabel("probability")
plt.ylabel("error value")
plt.title("misclassification, gini and entropy")
plt.legend()
```

[4]:
```
<matplotlib.legend.Legend at 0x124887a50>
```



The above graph shows the performance of classficiation error, gini index and cross entropy as the splitting criteria for the two-class classfication problem.

Generally speaking, the decision tree algorithm aims to find the optimal splitting strategy that leads to a maximum decrease of the average child node impurities over the parent node. For growing a decision tree, cross entropy or Gini index would be better approachs than classification error. This is because, as the graph shows, the classficiation error is not sensitive enough for tree growing. Moreover, different from classification error, cross entropy and gini are differentiable, which make it easier to optimize.

As for pruning a decision tree, classification error is better to reduce the size of decision tree than cross entropy or gini index. For classification error is the most straightforward way to measure accuracy. Moreover, because of its insensitiveness, classification error is less likely to cause overfitting or disturbed by small node inpurity.

2 Application: Predicting attitudes towards racist college professors

2.1 Estimate the models

[8]:
```
# load the data
gss_train=pd.read_csv("./data/gss_train.csv")
gss_test=pd.read_csv("./data/gss_test.csv")
x_train = gss_train.drop('colrac', axis=1)
```

```
y_train = gss_train.colrac
x_test = gss_test.drop('colrac', axis=1)
y_test = gss_test.colrac
```

[44]:
```
model_dict={
    "Logistic regression":(LogisticRegression(), {'C': np.logspace(-5,5,10)}),
    "Naive Bayes":(GaussianNB(),{}),
    #"Elastic net":(ElasticNet(),{'l1_ratio:':[.1, .7, .9, .95, .99, 1]}),
    "Decision tree":(DecisionTreeClassifier(),{'criterion':['gini','entropy'],
                                               'max_depth':range(1,20)}),
    "Bagging":(BaggingClassifier(),{'n_estimators':range(10,50,5)}),
    "Random forest":(RandomForestClassifier(),{'n_estimators':range(10,50,5),
                                               'max_depth':range(1,20),
                                               "criterion":['gini','entropy']}),
    "Boosting":(GradientBoostingClassifier(),{'learning_rate':np.
 ↪logspace(-5,0,10),
                                              'loss':['deviance','exponential']})
}
```

[49]:
```
best_model={}
score={}

for model in tqdm.tqdm(model_dict.keys()):
    gscv=GridSearchCV(model_dict[model][0],model_dict[model][1], cv=10,␣
 ↪refit=True)
    gscv.fit(x_train, y_train)
    best_model[model]=gscv.best_estimator_
    score[model]=gscv.best_score_
```

```
100%|      | 6/6 [03:51<00:00, 38.61s/it]
```

[50]:
```
# For elastic net
enet_cv= ElasticNetCV(cv=10, random_state=0).fit(x_train, y_train)
print('best alpha', enet_cv.alpha_)
print('best l1_ratio', enet_cv.alpha_)
```

```
best alpha 0.0038452641680228584
best l1_ratio 0.0038452641680228584
```

2.2 Evaluate the models

2.2.1 Cross-validated error rate

[57]:
```
# elastic net score
score['Elastic net']=accuracy_score(y_train, np.greater_equal(enet_cv.
 ↪predict(x_train),0.5))
```

[62]:
```
# The Cross-validated error rate
for model in score.keys():
    print(f'{model} error rate:', 1-score[model])
```

```
Logistic regression error rate: 0.20189701897018975
Naive Bayes error rate: 0.26558265582655827
Decision tree error rate: 0.21612466124661245
Bagging error rate: 0.20867208672086723
Random forest error rate: 0.1917344173441734
Boosting error rate: 0.1964769647696477
Elastic net error rate: 0.1842818428184282
```

Hence Elastic Net has the lowest error rate, the second best performing model is Random Forest.

2.2.2 ROC/AUC

[63]:
```
roc_auc={}
for model in best_model.keys():
    y_pred=best_model[model].predict_proba(x_train)[:,1]
    roc_auc[model]=roc_auc_score(y_train, y_pred)
```

[64]:
```
# for elastic net
roc_auc['Elastic net']=roc_auc_score(y_train, enet_cv.predict(x_train))
```

[65]:
```
# The ROC/AUC score
for model in roc_auc.keys():
    print(f'{model} ROC/AUC:', roc_auc[model])
```

```
Logistic regression ROC/AUC: 0.8945837743316
Naive Bayes ROC/AUC: 0.816411577930146
Decision tree ROC/AUC: 0.8799825134600341
Bagging ROC/AUC: 0.9999990796557914
Random forest ROC/AUC: 1.0
Boosting ROC/AUC: 0.9548083383185312
Elastic net ROC/AUC: 0.8914233123188071
```

Based on the above evaluation metric, I would initially choose random forest as the best performance model. This is because random forest get the highest roc/auc score and is also one of the models that get the lowest cross-validation error. However, since the auc score for random forest is too high (i.e., 1), there could be some potential overfitting issue. Similar problem could also exist with bagging, therefore, boosting could be the actual best model in terms of generalization power.

2.3 Evaluate the best model (boosting)

[83]:
```
# error rate
y_test_pred=best_model['Boosting'].predict(x_test)
err_rate=accuracy_score(y_test, y_test_pred )
print("Error rate is:", err_rate)
```

```
Error rate is: 0.795131845841785
```

[78]:
```
# roc/auc
y_test_pred=best_model['Boosting'].predict_proba(x_test)[:,1]
roc_auc=roc_auc_score(y_test, y_test_pred)
print("ROC/AUC score is:", roc_auc)
```

```
ROC/AUC score is: 0.8720125786163522
```

[84]:
```
# validation: compared to random forest
# error rate
y_test_pred=best_model['Random forest'].predict(x_test)
err_rate=accuracy_score(y_test, y_test_pred )
print("Error rate is:", err_rate)
# roc/auc
y_test_pred=best_model['Random forest'].predict_proba(x_test)[:,1]
roc_auc=roc_auc_score(y_test, y_test_pred)
print("ROC/AUC score is:", roc_auc)
```

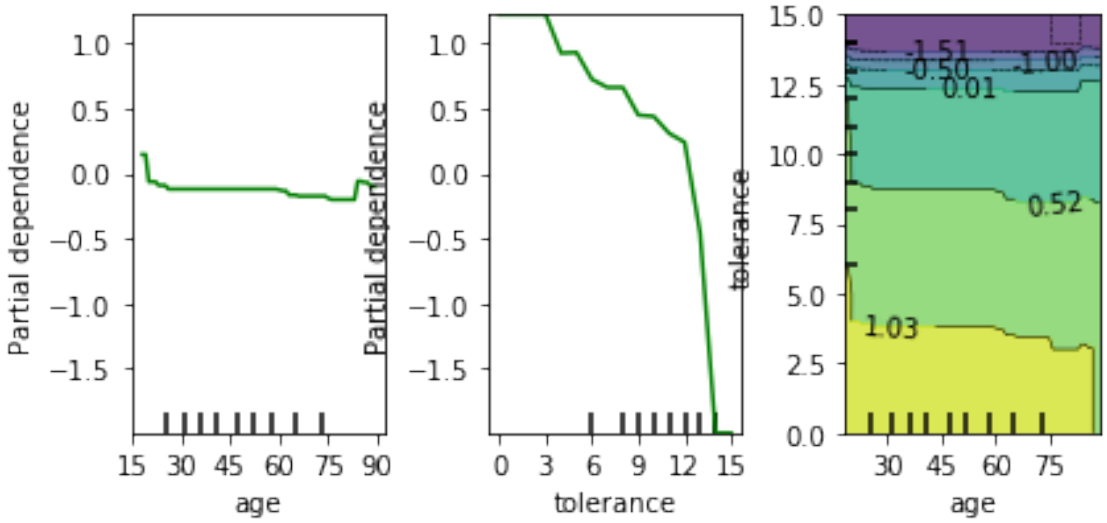
```
Error rate is: 0.7849898580121704
ROC/AUC score is: 0.8594670638861304
```

Compared to the performance of random forest and its own performance on training set, we could find that random forest has better performance: it generalize better than random forest on trainng set, its traing-test set performance is similar. That said, our best model gradient boosting has satifiying generalization power.

This result is intuitive, as we already got evidence from the model evaluations. Random forest has a high probability in overfitting the training data. Moreover, since this is a non-linear classification task, boosting is theoritcally more flexible and hence has better performance.

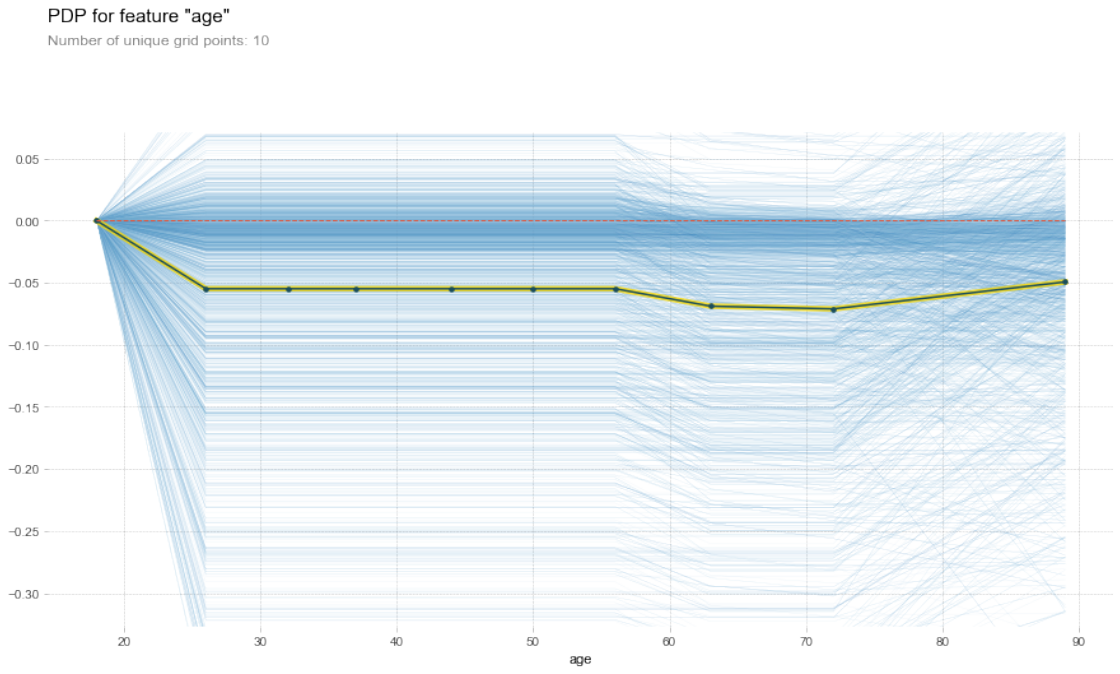
3 Bonus: PDPs/ICE

[89]:
```
# partial dependence plot
boost_model=best_model['Boosting']
feature_names=[n for n in x_train.columns if n!='colrac']
plot_partial_dependence(boost_model, x_train,␣
 ↪['age','tolerance',('age','tolerance')],feature_names)
```

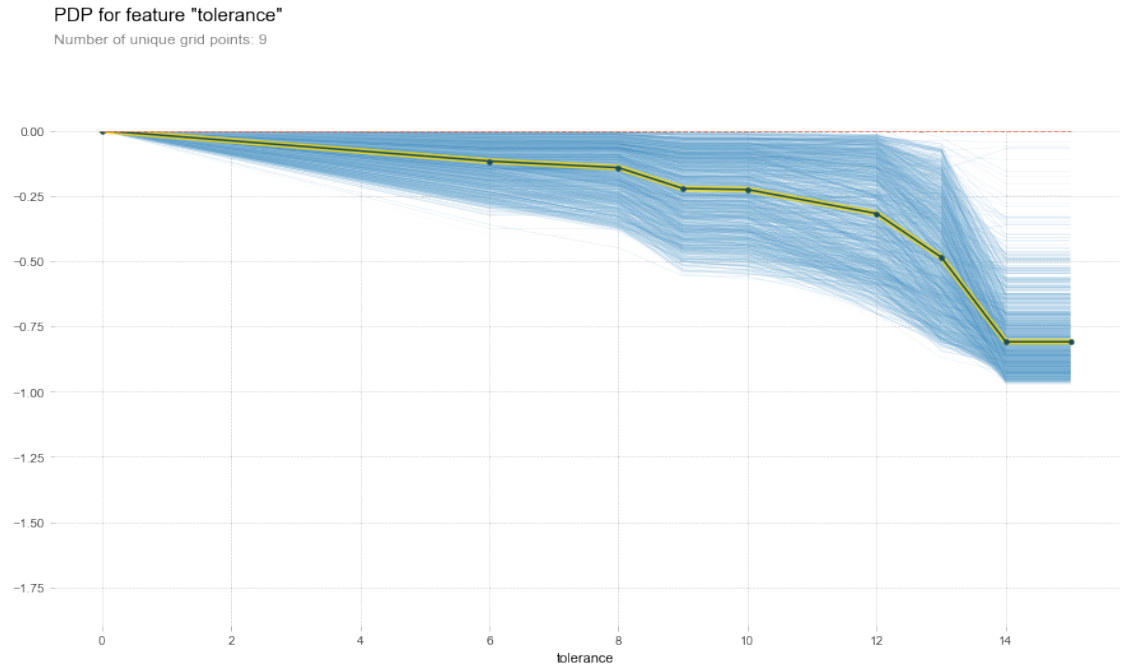


[99]:
```
# Individual Conditional Expectation (ICE)

pdp_age = pdp.pdp_isolate(model=boost_model,
                          dataset=x_train,
                          model_features=x_train.columns,
                          feature='age')
fig, axes = pdp.pdp_plot(pdp_isolate_out=pdp_age, feature_name='age',␣
 ↪plot_lines=True)
```



[100]:
```
pdp_tol = pdp.pdp_isolate(model=boost_model,
                          dataset=x_train,
                          model_features=x_train.columns,
                          feature='tolerance')
fig, axes = pdp.pdp_plot(pdp_isolate_out=pdp_tol, feature_name='tolerance',␣
↪plot_lines=True)
```



From the PDP plot, we could find that marginal effect 'age' features have on the predicted outcome is fairly limited. As the value of age change, the prediction outcome will basically hold the same. However, in terms of 'tolerance', the partial dependence plot shows that tolerance is a meaningful feature in this prediction model. As tolerance increase, the predicted 'colrac' drops. That said, as tolerance goes up, the willingness to allow a racist professor to teach goes down.

From the ICE plot we could find more details in terms of the marginal effect of age and tolerance. The general trend of these two features on the prediction results hold the same for most people/line in the gss dataset. Generally, all data lines have no specific interactions, that means that the PDP is already a good summary of the relationships between age/tolerance features and the predicted opinion on racist professor in college.