Problem Set 5- Shengwenxin Ni

March 1, 2020

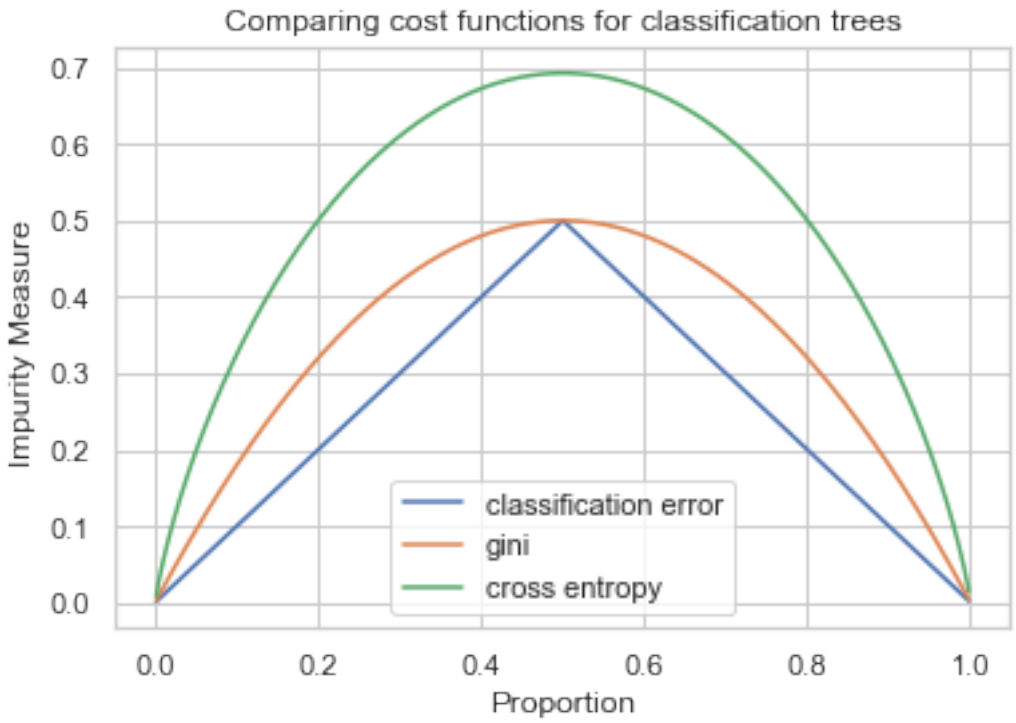
[9]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import preprocessing
plt.rc("font", size=14)
import seaborn as sns
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
```

1 Question 1

[2]:
```
p = [i/1000 for i in range(1000)][1:-1]
class_error = [1-max(i,1-i) for i in p ]
gini = [2*i*(1-i) for i in p ]
cross_entropy =  [-i*np.log(i)-(1-i)*np.log(1-i) for i in p ]

plt.plot(p,class_error,label='classification error')
plt.plot(p,gini,label='gini')
plt.plot(p,cross_entropy,label='cross entropy')
plt.ylabel('Impurity Measure ')
plt.xlabel('Proportion')
plt.title('Comparing cost functions for classification trees')
plt.legend()
```

[2]:
```
<matplotlib.legend.Legend at 0x1a24ca1ad0>
```



From the graph above, we can see that gini index and cross-entropy has a similar parabola shape, which is smooth, while the classification error has a triangular shape, which is sharp.

When BUILDing a classification tree, both GINI INDEX and CROSS ENTROPY are good choices. These two evaluations are numerically similar: they both take a small value near 0 if a node is pure and is very sensitive to the purity.Thus, they are effective in evaluating the quality of split.

In contrast, when PRUNE-ing a classification tree, the CLASSIFICATION ERROR is better. In this step, if we still use gini index or cross entropy to optimize classification accuracy, we would end up fitting on too much noise. To avoid the problem of overfitting, the insensitive classfication error becomes the better choice.

2 Question 2

[10]:
```
train = pd.read_csv('gss_train.csv')
test = pd.read_csv('gss_test.csv')

X_train = train.loc[:, train.columns != 'colrac']
y_train = train['colrac']

X_test = test.loc[:, test.columns != 'colrac']
y_test = test['colrac']
```

[11]:
```python
from sklearn.model_selection import KFold
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn import metrics
```

2.1 Logistic Regression

[6]:
```python
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
lr_pred = cross_val_predict(LogisticRegression(solver= "lbfgs"), X_train,␣
 ↪y_train, cv = 10)
lr_pred
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:947:
ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:947:
ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:947:
ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:947:
ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:947:
ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:947:
ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:947:
ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:947:
ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:947:
ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:947:
ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
```

[6]:
```
array([1, 0, 0, …, 1, 0, 1])
```

2.2 Naive Bayes

[149]:
```
from sklearn.naive_bayes import GaussianNB
```

[198]:
```
nb_pred = cross_val_predict(GaussianNB(), X_train, y_train, cv=10)
nb_pred
```

[198]:
```
array([1, 0, 1, …, 1, 0, 1])
```

2.3 Elastic net regression

[5]:
```
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV
```

[202]:
```
param = {'alpha':[0.1,0.5,1,10,100],
         'l1_ratio':np.linspace(0.1,1,11),
         'penalty':['elasticnet']}
eNet_cv = GridSearchCV(SGDClassifier(), param, cv = 10)
eNet_fit = eNet_cv.fit(X_train,y_train)
eNet_pred = eNet_fit.predict(X_train)
eNet_pred
```

```
/opt/anaconda3/lib/python3.7/site-
packages/sklearn/model_selection/_search.py:814: DeprecationWarning: The default
of the `iid` parameter will change from True to False in version 0.22 and will
be removed in 0.24. This will change numeric results when test-set sizes are
unequal.
  DeprecationWarning)
```

[202]:
```
array([1, 0, 1, …, 1, 1, 1])
```

2.4 Decision tree (CART)

[9]:
```
from scipy.stats import randint
from sklearn.tree import DecisionTreeClassifier
```

[203]:
```
param = {"max_depth": [3, None],
         "max_features": randint(1, 9),
         "min_samples_leaf": randint(1, 9),
         "criterion": ["gini", "entropy"]}

cart_cv = RandomizedSearchCV(DecisionTreeClassifier(), param, cv = 10)

#print("Tuned Decision Tree Parameters: {}".format(tree_cv.best_params_))
```

```
#print("Best score is {}".format(tree_cv.best_score_))

cart_fit = cart_cv.fit(X_train, y_train)
cart_pred = cart_fit.predict(X_train)
cart_pred
```

```
/opt/anaconda3/lib/python3.7/site-
packages/sklearn/model_selection/_search.py:814: DeprecationWarning: The default
of the `iid` parameter will change from True to False in version 0.22 and will
be removed in 0.24. This will change numeric results when test-set sizes are
unequal.
  DeprecationWarning)
```

[203]:
```
array([1, 0, 1, …, 1, 1, 1])
```

2.5 Bagging

[205]:
```
from sklearn.ensemble import BaggingClassifier
```

[207]:
```
param = {'n_estimators': [int(x) for x in np.linspace(start = 200, stop = 2000,␣
 ↪num = 10)],
         'base_estimator__max_depth' : [1, 2, 3, 4, 5],
         'max_samples' : [0.05, 0.1, 0.2, 0.5]}

bag_cv = GridSearchCV(BaggingClassifier(DecisionTreeClassifier()), param, cv =␣
 ↪10)

bag_fit = bag_cv.fit(X_train, y_train)
bag_pred = bag_fit.predict(X_train)
bag_pred
```

[207]:
```
array([1, 0, 1, …, 1, 0, 1])
```

2.6 Random Forest

[4]:
```
from sklearn.ensemble import RandomForestClassifier
```

[12]:
```
param = {'n_estimators':[1, 4, 16, 64, 100, 200],
         'max_features': ['auto','log2'],
         'min_samples_leaf': [1,5,10],
         'max_depth': [10, 30, 50,70, 90, None]}

rf_cv = GridSearchCV(RandomForestClassifier(),param, cv = 10)
```

```
rf_fit = rf_cv.fit(X_train, y_train)
rf_pred = rf_fit.predict(X_train)
rf_pred
```

[12]:
```
array([1, 1, 1, …, 1, 1, 1])
```

2.7 Boosting

[182]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

[222]:
```
param = {'n_estimators':[1, 2, 4, 8, 16, 32, 64, 100, 200],
         'learning_rate': [0.0001, 0.001, 0.01, 0.1, 0.2]}
bst_cv = GridSearchCV(GradientBoostingClassifier(),param, cv = 10)
bst_fit = bst_cv.fit(X_train, y_train)
bst_pred = bst_fit.predict(X_train)
bst_pred
```

[222]:
```
array([1, 0, 1, …, 1, 0, 1])
```

3 Question 3

[253]:
```
import sklearn.metrics as metrics

def roc_auc(model):
    probs = model.predict_proba(X_train)
    preds = probs[:,1]
    fpr, tpr, threshold = metrics.roc_curve(y_train, preds)
    roc_auc = metrics.auc(fpr, tpr)
    return roc_auc
```

3.1 Logistic Regression

[342]:
```
lr_err = 1 - cross_val_score(LogisticRegression(), X_train, y_train, cv = 10).
 ↪mean()
print('The cv error rate for logistic regression is:', lr_err)
lr_fit = LogisticRegression().fit(X_train,y_train)
lr_ra = roc_auc(lr_fit)
print('The roc-auc for logistic regression is:', lr_ra)
```

```
The cv error rate for logistic regression is: 0.20731955760718945
The roc-auc for logistic regression is: 0.8958925037964198
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

3.2 Naive Bayes

[341]:
```
nb_err = 1 - cross_val_score(GaussianNB(), X_train, y_train, cv = 10).mean()
print('The cv error rate for naive bayes is:', nb_err)
nb_fit = GaussianNB().fit(X_train,y_train)
nb_ra =  roc_auc(nb_fit)
print('The roc-auc for naive bayes is:', nb_ra)
```

```
The cv error rate for naive bayes is: 0.26555250977590383
The roc-auc for naive bayes is: 0.816411577930146
```

3.3 Elastic Net

[339]:
```
eNet_err = 1- eNet_fit.best_score_
print('The cv error rate for elastic net is:',eNet_err)

from sklearn.calibration import CalibratedClassifierCV
eNet_fit.best_params_
eNet_best = SGDClassifier(loss = 'hinge',alpha = 0.1, l1_ratio= 0.1, penalty =␣
 ↪"elasticnet")
eNet_best_fit = eNet_best.fit(X_train,y_train)
calibrator = CalibratedClassifierCV(eNet_best_fit, cv='prefit')
model = calibrator.fit(X_train, y_train)
eNet_prob = model.predict_proba(X_train)
eNet_ra = roc_auc(model)
print('The roc-auc for elastic net is:', eNet_ra)
```

```
The cv error rate for elastic net is: 0.22967479674796742
The roc-auc for elastic net is: 0.8716708849109567
```

3.4 Cart

[340]:
```
cart_err = 1- cart_fit.best_score_
print('The cv error rate for CART is:',cart_err)
cart_ra = roc_auc(cart_fit)
print('The roc-auc for CART is:', cart_ra)
```

```
The cv error rate for CART is: 0.27168021680216803
The roc-auc for CART is: 0.835002530946574
```

3.5 Random Forest

[336]:
```
rf_err = 1- rf_fit.best_score_
print('The cv error rate for random forest is:',rf_err)
rf_ra = roc_auc(rf_fit)
print('The roc-auc for random forest is:', rf_ra)
```

```
The cv error rate for random forest is: 0.19241192411924124
The roc-auc for random forest is: 1.0
```

3.6 Bagging

[335]:
```
bag_err = 1- bag_fit.best_score_
print('The cv error rate for bagging is:',bag_err)
bag_ra = roc_auc(bag_fit)
print('The roc-auc for bagging is:', bag_ra)
```

```
The cv error rate for bagging is: 0.20189701897018975
The roc-auc for bagging is: 0.9266780175785744
```

3.7 Boosting

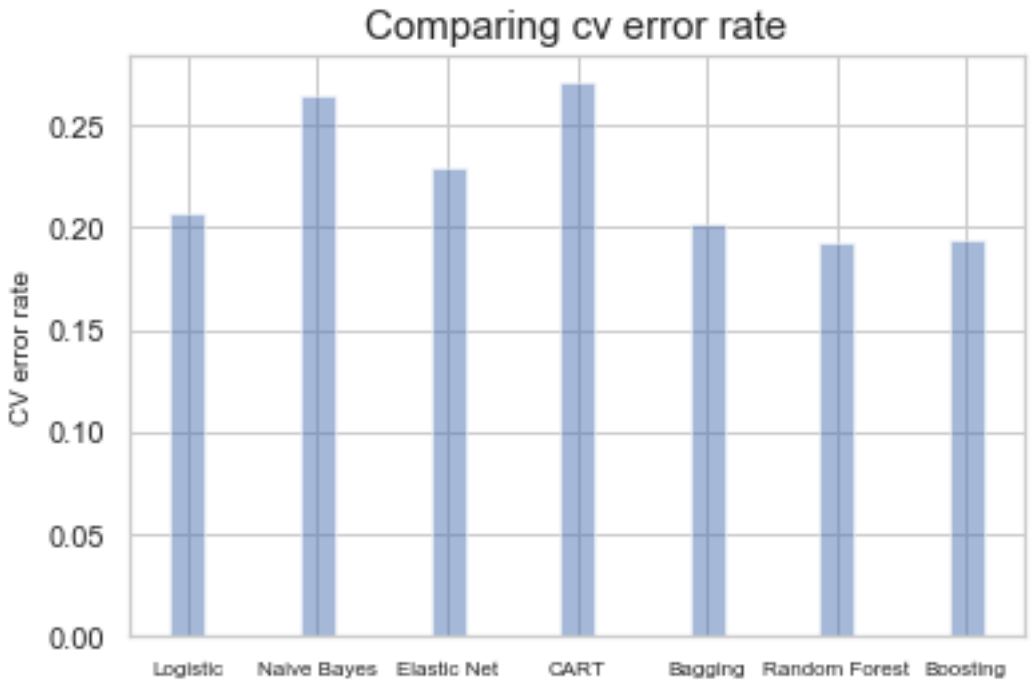
[334]:
```
bst_err = 1- bst_fit.best_score_
print('The cv error rate for boosting is:',bst_err)
bst_ra = roc_auc(bst_fit)
print('The roc-auc for boosting is:', bst_ra)
```

```
The cv error rate for boosting is: 0.19444444444444442
The roc-auc for boosting is: 0.9474547880907459
```

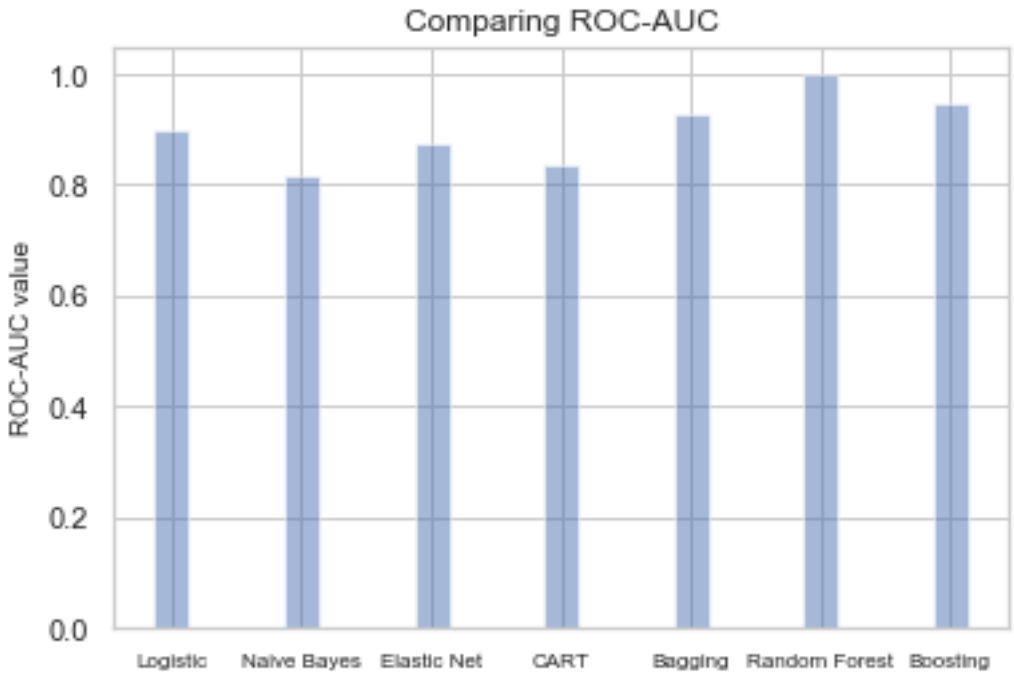
3.8 Comparing Models

[347]:
```
objects = ('Logistic', 'Naive Bayes', 'Elastic Net', 'CART', 'Bagging', 'Random␣
 ↪Forest','Boosting')
performance = [lr_err,nb_err,eNet_err,cart_err,bag_err,rf_err,bst_err]
pos = [1,4,7,10,13,16,19]
plt.bar(pos,performance, align='center', alpha=0.5)
plt.xticks(pos,objects,fontsize=8)
plt.ylabel('CV error rate', fontsize= 10 )
plt.title('Comparing cv error rate',fontsize=15)

plt.show()
```



[348]:
```
objects = ('Logistic', 'Naive Bayes', 'Elastic Net', 'CART', 'Bagging', 'Random␣
↪Forest','Boosting')
performance = [lr_ra,nb_ra,eNet_ra,cart_ra,bag_ra,rf_ra,bst_ra]
pos = [1,4,7,10,13,16,19]
plt.bar(pos,performance, align='center', alpha=0.5)
plt.xticks(pos,objects,fontsize=8)
plt.ylabel('ROC-AUC value', fontsize= 10 )
plt.title('Comparing ROC-AUC',fontsize=12)
plt.show()
```



From above two diagrams, it's salient that RANDOM FOREST has is the best because, for the best model after hyperparameter tuning:

1. It has the lowest CV error rate : CV error rate is generated by calculating the mean of the error rate from each cross-validation. Since lower the error rate, better the accuracy of prediction. Thus, based on the performance on the training set, random forest wins.
2. It has the highest ROC-AUC: Suprisingly, this model gain a perfect ROC-AUC score, which denotes the overall performance of the classifier across all potential threshold. Equivalently, high the ROC-AUC score, better the accuracy of prediction. Thus, based on the performance on the training set, random forest wins again.

4 Question 4

[355]:
```
print('The best model of the random forest has the parameters as:')
rf_fit.best_params_
```

```
The best model of the random forest has the parameters as:
```

[355]:
```
{'max_depth': 50,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'n_estimators': 200}
```

[12]:
```
best_model = RandomForestClassifier(max_depth = 50, max_features= 'auto',␣
↪min_samples_leaf = 1,n_estimators = 200)
```

[13]:
```
test_pred = best_model.fit(X_train,y_train).predict(X_test)

err_ct = 0
for i in range(len(y_test)):
    if y_test[i] != test_pred[i]:
        err_ct += 1

print('Error rate for the best model is:',err_ct/len(y_test))

probs = best_model.fit(X_train,y_train).predict_proba(X_test)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

print('ROC-AUC value for the best model is:',roc_auc)
```

```
Error rate for the best model is: 0.2028397565922921
ROC-AUC value for the best model is: 0.8704071499503476
```

4.1 Comments

The error rate of this best model increases slightly from 0.1924 to 0.1967 and the roc-auc value decreases from 1.0 to 0.8710. It can be expected that the performance of predictions drop (evaluated by both methods) compare to the predictions on the training dataset. I still think this model works well because:

1. The increases in error rate is very insiginificant. And the error rate as 0.1967 is lower than majority of models on the training set.
2. ROC-AUC value as 0.8710 is higher than naive bayes (0.8164) and cart (0.8360) and almost equivalent to logistic regression (0.8958) and elastic net (0.87167) on their training set. We can expect that ROC-AUC would drop for the fitting on the training set as well for all other alternative models.

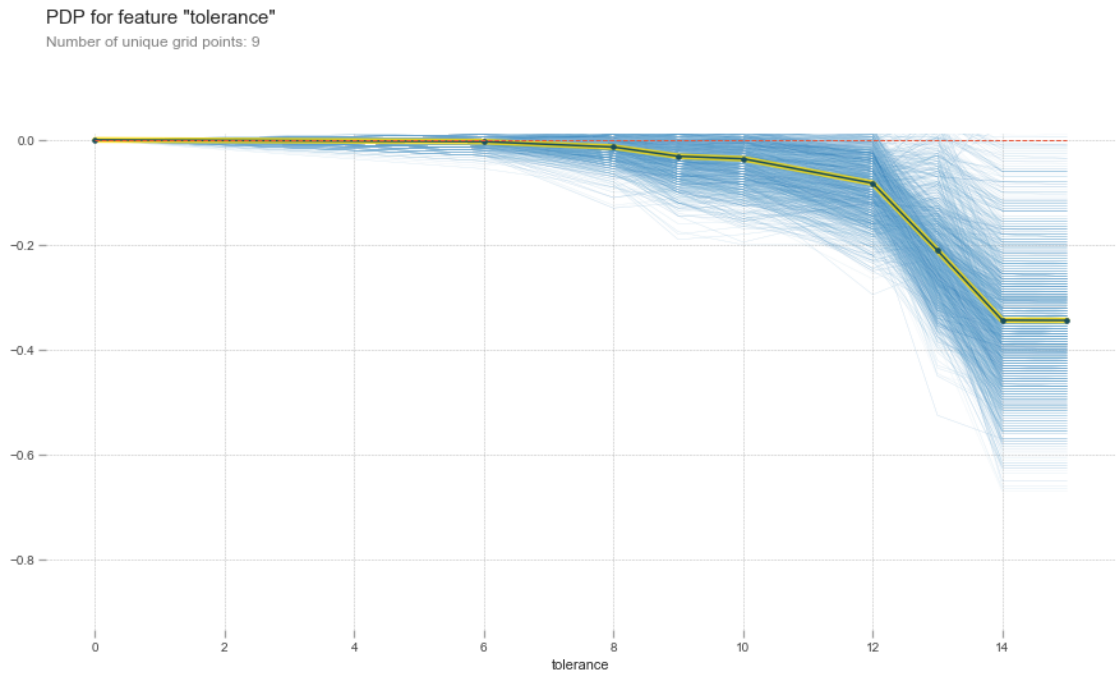
Therefore, I believe this selected best model is a good one.

However, we still need to be aware the possibility of overfitting, since ROC-AUC 1.0 sounds quite 'dangerous'. In addition, we can see that this best model takes extremly large value on number of trees (200) and max-depth (50) and a small value for minimal observations for each leaf(1). Therefore, it's highly likely that this model has the problem of over-fitting.

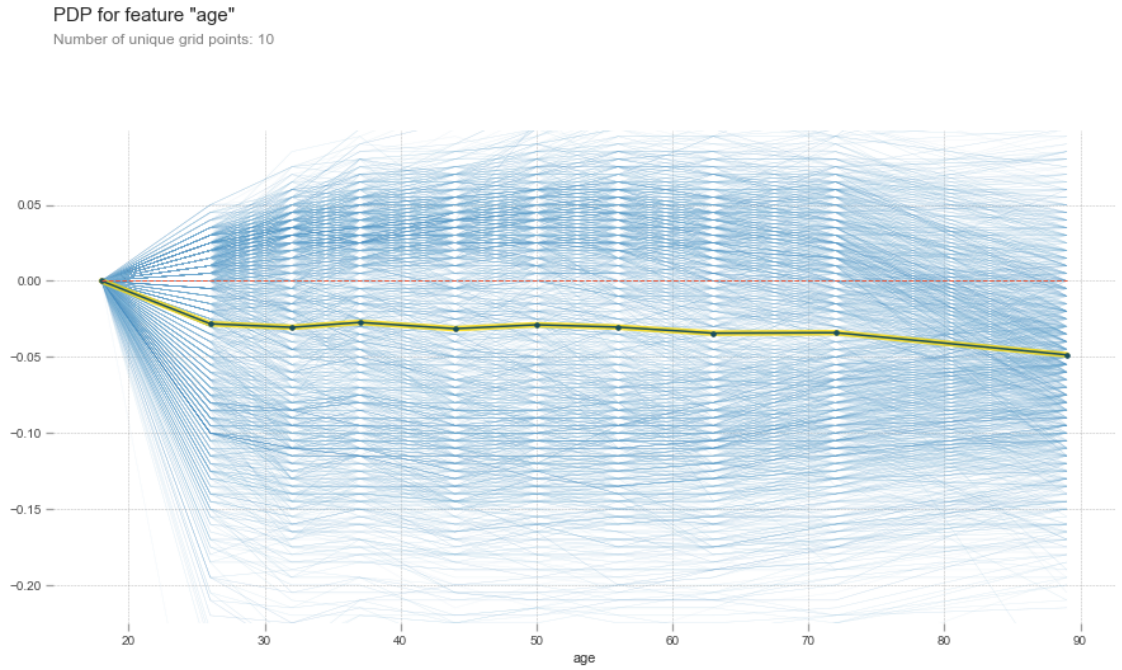
5 Question 5

[5]:
```
from pdpbox import pdp, get_dataset,info_plots
```

[21]:
```
pdp_tol = pdp.pdp_isolate(model = best_model, dataset = X_train, model_features␣
↪= X_train.columns, feature = 'tolerance')
fig,axes = pdp.pdp_plot(pdp_tol,'tolerance',plot_lines = True)
```



[20]:
```
pdp_age = pdp.pdp_isolate(model = best_model, dataset = X_train, model_features␣
↪= X_train.columns, feature = 'age')
fig,axes = pdp.pdp_plot(pdp_age,'age',plot_lines = True)
```



ICE plots display one line per instance that shows how its prediction according to the variation of a particular feature. In the above two diagrams,PDP as the average of the lines of ICE plot, is displayed in greenish-yellow color.

For the feature 'tolerance', we can see that the value of predictions has a salient decreasing trend as tolerance value become greater. Such decrease is especially significant as the tolerance value drops from 10-14. Since each individual instance generally follows the decreasing major pattern, we can say that the PDP is already a good summary of the relationships between the 'tolerance' and the predicted 'colrac'.

In contrast, for the feature 'age', instances vary significantly and the PDP line remains flat around 0 (around -0.03). This pattern indicates that the variation of the feature 'age' won't change much to predicted 'colrac'.