Xiong_Yinjiang_HW5

February 27, 2020

1.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?

Answer: To grow a decision tree, Gini index and cross-entropy can control the variance to avoid overfitting so they are generally preferred over classification error. On the other hand, pruning is a process to prevent from fitting noise, so classification error can be used to maximize accuracy.

2.Estimate the following models, predicting colrac using the training set (the training .csv) with 10-fold CV:

Logistic regression
Naive Bayes
Elastic net regression
Decision tree (CART)
Bagging
Random forest
Boosting

Tune the relevant hyperparameters for each model as necessary. Only use the tuned model with the best performance for the remaining exercises. Be sure to leave sufficient time for hyperparameter tuning. Grid searches can be computationally taxing and take quite a while, especially for tree-aggregation methods.

[46]:
```
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
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import ElasticNetCV
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.inspection import plot_partial_dependence
```

[32]:
```
gss_train = pd.read_csv('gss_train.csv')
gss_test = pd.read_csv('gss_test.csv')
gss_train.head(5)
```

[32]:
```
   age  attend  authoritarianism  black  born  childs  colath  colrac  colcom  \
0   21       0                 4      0     0       0       1       1       0
1   42       0                 4      0     0       2       0       1       1
2   70       1                 1      1     0       3       0       1       1
3   35       3                 2      0     0       2       0       1       0
4   24       3                 6      0     1       3       1       1       0

   colmil  ...  partyid_3_Ind  partyid_3_Rep  relig_CATHOLIC  relig_NONE  \
0       1  ...              1              0               1           0
1       0  ...              1              0               0           0
2       0  ...              0              0               0           0
3       1  ...              1              0               0           0
4       0  ...              1              0               1           0

   relig_other  social_cons3_Mod  social_cons3_Conserv  spend3_Mod  \
0            0                 1                     0           0
1            0                 0                     0           1
2            0                 0                     0           0
3            1                 0                     0           0
4            0                 1                     0           0

   spend3_Liberal  zodiac_other
0               0             1
1               0             1
2               0             1
3               1             1
4               0             1

[5 rows x 56 columns]
```

[4]:
```
gss_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1476 entries, 0 to 1475
Data columns (total 56 columns):
age                     1476 non-null int64
attend                  1476 non-null int64
authoritarianism        1476 non-null int64
```

```
black                      1476 non-null int64
born                       1476 non-null int64
childs                     1476 non-null int64
colath                     1476 non-null int64
colrac                     1476 non-null int64
colcom                     1476 non-null int64
colmil                     1476 non-null int64
colhomo                    1476 non-null int64
colmslm                    1476 non-null int64
con_govt                   1476 non-null int64
egalit_scale               1476 non-null int64
evangelical                1476 non-null int64
grass                      1476 non-null int64
happy                      1476 non-null int64
hispanic_2                 1476 non-null int64
homosex                    1476 non-null int64
income06                   1476 non-null int64
mode                       1476 non-null int64
owngun                     1476 non-null int64
polviews                   1476 non-null int64
pornlaw2                   1476 non-null int64
pray                       1476 non-null int64
pres08                     1476 non-null int64
reborn_r                   1476 non-null int64
science_quiz               1476 non-null int64
sex                        1476 non-null int64
sibs                       1476 non-null int64
social_connect             1476 non-null int64
south                      1476 non-null int64
teensex                    1476 non-null int64
tolerance                  1476 non-null int64
tvhours                    1476 non-null int64
vetyears                   1476 non-null int64
wordsum                    1476 non-null int64
degree_Bachelor.deg        1476 non-null int64
degree_other               1476 non-null int64
marital_Divorced           1476 non-null int64
marital_Never.married      1476 non-null int64
marital_other              1476 non-null int64
news_FEW.TIMES.A.WEEK      1476 non-null int64
news_LESS.THAN.ONCE.WK     1476 non-null int64
news_NEVER                 1476 non-null int64
news_other                 1476 non-null int64
partyid_3_Ind              1476 non-null int64
partyid_3_Rep              1476 non-null int64
relig_CATHOLIC             1476 non-null int64
relig_NONE                 1476 non-null int64
relig_other                1476 non-null int64
```

```
social_cons3_Mod          1476 non-null int64
social_cons3_Conserv      1476 non-null int64
spend3_Mod                1476 non-null int64
spend3_Liberal            1476 non-null int64
zodiac_other              1476 non-null int64
dtypes: int64(56)
memory usage: 645.9 KB
```

[49]:
```
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']
```

[14]:
```
NB = GaussianNB()
print('The accuracy for naive bayes is',
      np.mean(cross_val_score(NB, X_train, y_train, scoring = 'accuracy',␣
 ↪cv=10)))
print('The roc/auc for naive bayes is',
      np.mean(cross_val_score(NB, X_train, y_train, scoring = 'roc_auc', cv=10)))
```

```
The accuracy for naive bayes is 0.7344474902240962
The roc/auc for naive bayes is 0.8080500250922787
```

[8]:
```
EN = ElasticNetCV(cv=10).fit(X_train, y_train)
EN.alpha_, EN.l1_ratio
```

[8]:
```
(0.0038452641680228584, 0.5)
```

[15]:
```
# since elastic net is a regression instead of classifier, we use mse
EN_tuned = ElasticNet(alpha=0.00385, l1_ratio=0.5)
print('The mse for elastic net is',
      -np.mean(cross_val_score(EN_tuned, X_train, y_train,␣
 ↪scoring='neg_mean_squared_error', cv=10)))
print('The roc/auc for elastic net is',
      np.mean(cross_val_score(EN, X_train, y_train, scoring = 'roc_auc', cv=10)))
```

```
The mse for elastic net is 0.14714547782969206
The roc/auc for elastic net is 0.8741367971166685
```

[10]:
```
DT = DecisionTreeClassifier()
```

[16]:
```
print('The accuracy for decision tree is',
      np.mean(cross_val_score(DT, X_train, y_train, scoring = 'accuracy',␣
 ↪cv=10)))
print('The roc/auc for decision tree is',
      np.mean(cross_val_score(DT, X_train, y_train, scoring = 'roc_auc', cv=10)))
```

```
The accuracy for decision tree is 0.7256358810529473
The roc/auc for decision tree is 0.7260006894513937
```

[83]:
```
# tune hyper parameters for bagging from random_grid
param_grid = {
    'n_estimators': [5, 10, 20, 30, 40, 50]
              }
BG = BaggingClassifier()
BG_grid = GridSearchCV(estimator = BG, param_grid = param_grid,
                          cv = 10, n_jobs = -1, verbose = 2)
BG_grid.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 6 candidates, totalling 60 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  60 out of  60 | elapsed:    7.2s finished
```

[83]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=BaggingClassifier(base_estimator=None, bootstrap=True,
                                         bootstrap_features=False,
                                         max_features=1.0, max_samples=1.0,
                                         n_estimators=10, n_jobs=None,
                                         oob_score=False, random_state=None,
                                         verbose=0, warm_start=False),
             iid='warn', n_jobs=-1,
             param_grid={'n_estimators': [5, 10, 20, 30, 40, 50]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=2)
```

[85]:
```
BG_grid.best_params_
```

[85]:
```
{'n_estimators': 50}
```

[17]:
```
BG_final = BaggingClassifier(n_estimators=50)
print('The accuracy for bagging is',
      np.mean(cross_val_score(BG_final, X_train, y_train, scoring = 'accuracy',␣
 ↪cv=10)))
print('The roc/auc for bagging is',
      np.mean(cross_val_score(BG_final, X_train, y_train, scoring = 'roc_auc',␣
 ↪cv=10)))
```

```
The accuracy for bagging is 0.7845584215910199
The roc/auc for bagging is 0.8660663230130032
```

[82]:
```
# tune hyper parameters for random forest from random_grid
param_grid = {
    'n_estimators': [5, 10, 20, 30, 40, 50],
    'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
```

```
    'max_features': [5, 10, 20, 30, 50, 55]
              }
RF = RandomForestClassifier()
RF_grid = GridSearchCV(estimator = RF, param_grid = param_grid,
                          cv = 10, n_jobs = -1, verbose = 2)
RF_grid.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 360 candidates, totalling 3600 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  36 tasks      | elapsed:    3.4s
[Parallel(n_jobs=-1)]: Done 278 tasks      | elapsed:   15.5s
[Parallel(n_jobs=-1)]: Done 533 tasks      | elapsed:   35.0s
[Parallel(n_jobs=-1)]: Done 879 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 1244 tasks      | elapsed:  1.7min
[Parallel(n_jobs=-1)]: Done 1802 tasks      | elapsed:  2.5min
[Parallel(n_jobs=-1)]: Done 2410 tasks      | elapsed:  3.4min
[Parallel(n_jobs=-1)]: Done 3017 tasks      | elapsed:  4.4min
[Parallel(n_jobs=-1)]: Done 3600 out of 3600 | elapsed:  5.5min finished
```

[82]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
                                              criterion='gini', 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='warn', n_jobs=None,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
                         'max_features': [5, 10, 20, 30, 50, 55],
                         'n_estimators': [5, 10, 20, 30, 40, 50]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=2)
```

[77]:
```
RF_grid.best_params_
```

[77]:
```
{'max_depth': 20, 'max_features': 5, 'n_estimators': 50}
```

[18]:
```
RF_final = RandomForestClassifier(n_estimators=50, max_depth=20, max_features=5)
print('The accuracy for random forest is',
```

```
    np.mean(cross_val_score(RF_final, X_train, y_train, scoring = 'accuracy',␣
↪cv=10)))
print('The roc/auc for random forest is',
    np.mean(cross_val_score(RF_final, X_train, y_train, scoring = 'roc_auc',␣
↪cv=10)))
```

```
The accuracy for random forest is 0.7885347681608468
The roc/auc for random forest is 0.872091897036565
```

[95]:
```
# tune hyper parameters for gradiant boosting from random_grid
param_grid = {
    'n_estimators': [5, 10, 20, 30, 40, 50],
     'learning_rate': [0.05, 0.1, 0.2, 0.3],
    'max_features': [None, 'sqrt']
            }
GB = GradientBoostingClassifier()
GB_grid = GridSearchCV(estimator = GB, param_grid = param_grid,
                          cv = 10, n_jobs = -1, verbose = 2)
GB_grid.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 48 candidates, totalling 480 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:    4.8s
[Parallel(n_jobs=-1)]: Done 480 out of 480 | elapsed:   16.7s finished
```

[95]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=GradientBoostingClassifier(criterion='friedman_mse',
                                                  init=None, learning_rate=0.1,
                                                  loss='deviance', 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),
             iid='warn', n_jobs=-1,
             param_grid={'learning_rate': [0.05, 0.1, 0.2, 0.3],
                         'max_features': [None, 'sqrt'],
                         'n_estimators': [5, 10, 20, 30, 40, 50]},
```

```
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=2)
```

[96]:
```
GB_grid.best_params_
```

[96]:
```
{'learning_rate': 0.2, 'max_features': None, 'n_estimators': 40}
```

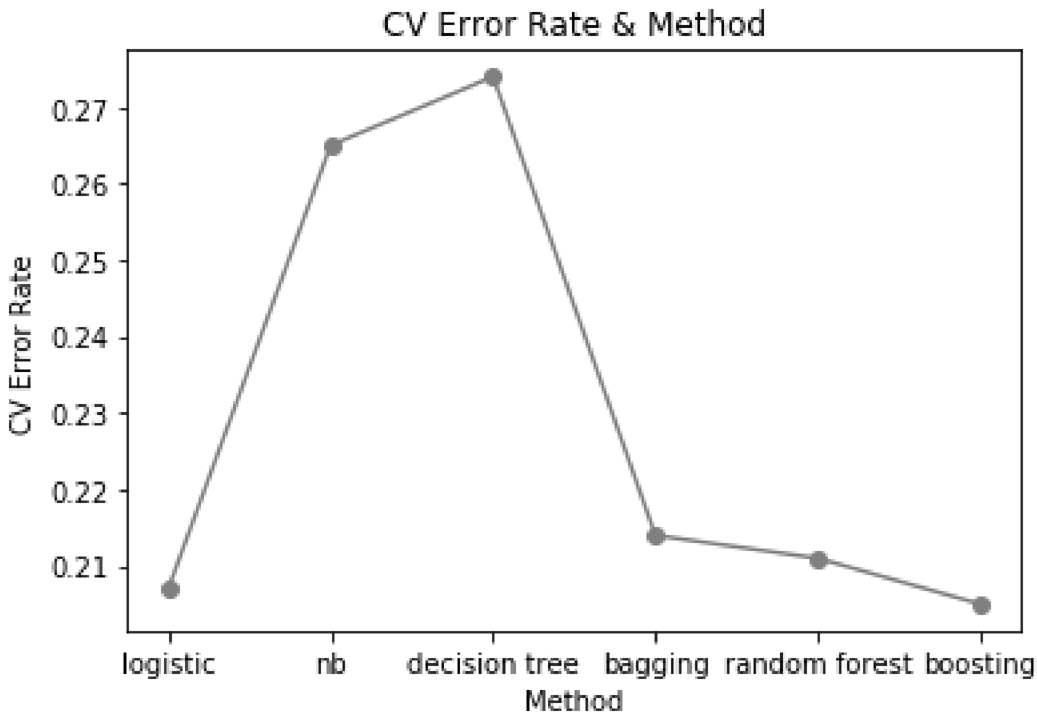
[19]:
```
GB_final = GradientBoostingClassifier(learning_rate=0.2, max_features='sqrt',␣
 ↪n_estimators=40)
print('The accuracy for gradiant boosting is',
      np.mean(cross_val_score(GB_final, X_train, y_train, scoring = 'accuracy',␣
 ↪cv=10)))
print('The roc/auc for gradiant boosting is',
      np.mean(cross_val_score(GB_final, X_train, y_train, scoring = 'roc_auc',␣
 ↪cv=10)))
```

```
The accuracy for gradiant boosting is 0.7946938652116026
The roc/auc for gradiant boosting is 0.8730648727530015
```

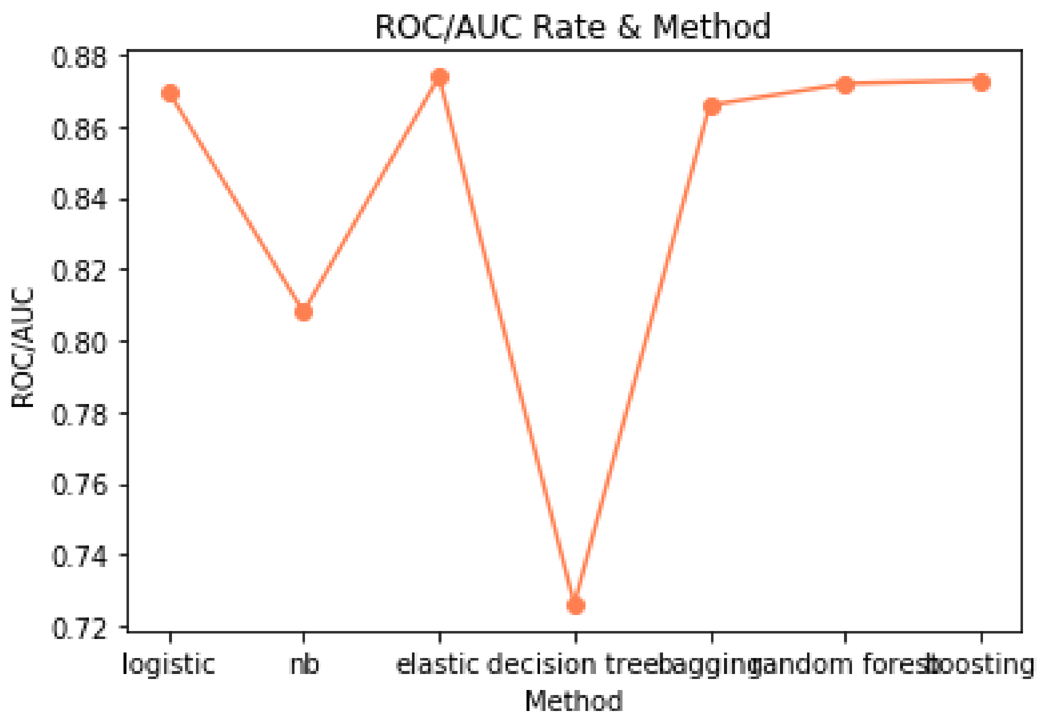
3.Compare and present each model's (training) performance based on

- Cross-validated error rate
- ROC/AUC

[30]:
```
# we can see that boosting has the lowest cv error rate and logistic regression␣
 ↪has the second lowest
# graph the error rate
CV_error_rate = list(1 - np.array([0.793, 0.735, 0.726, 0.786, 0.789, 0.795]))
method = ['logistic', 'nb', 'decision tree', 'bagging', 'random forest',␣
 ↪'boosting']
plt.plot(method, CV_error_rate, marker='o', color='grey')
plt.xlabel('Method')
plt.ylabel('CV Error Rate')
plt.title('CV Error Rate & Method');
```



[31]:
```
# graph roc/auc
# elastic net has the best ROC/AUC
roc_auc = [0.870, 0.808, 0.874, 0.726, 0.866, 0.872, 0.873]
method = ['logistic', 'nb', 'elastic', 'decision tree', 'bagging', 'random␣
↪forest', 'boosting']
plt.plot(method, roc_auc, marker='o', color='coral')
plt.xlabel('Method')
plt.ylabel('ROC/AUC')
plt.title('ROC/AUC Rate & Method');
```



4.Which is the best model? Defend your choice.

Answer: Overall, boosting is the best model. It has the lowest error rate and second highest ROC/AUC.

5.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?

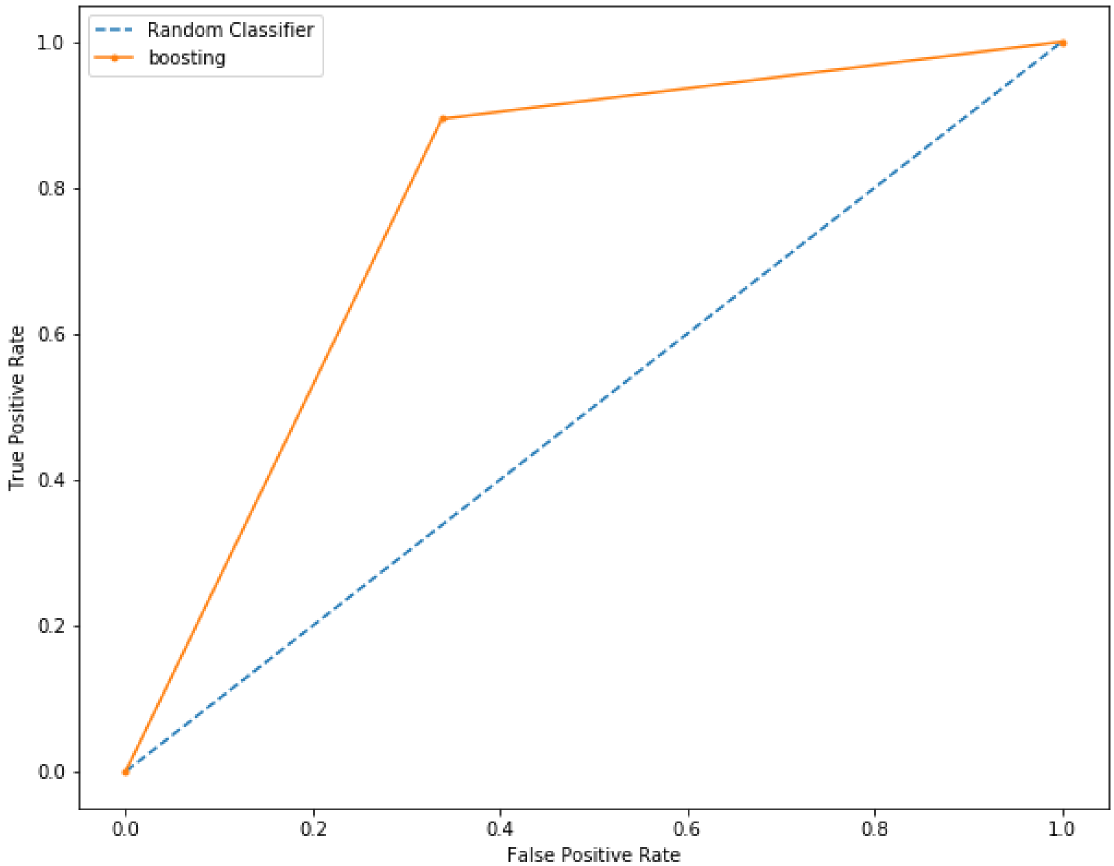
[52]:
```
GB_final.fit(X_train, y_train)
print('The test error rate is', 1 - accuracy_score(y_test, GB_final.
 ↪predict(X_test)))
print('The roc/auc rate is', roc_auc_score(y_test, GB_final.predict(X_test)))
```

```
The test error rate is 0.20486815415821502
The roc/auc rate is 0.7876944720291295
```

[43]:
```
# plot the roc curve
def get_roc_auc(model, label):
    #calcualte auc score
    model_auc = roc_auc_score(y_test, model.predict(X_test))
    # summarize scores
    print(label+ ': AUC = %.3f' % (model_auc))
    # calculate roc curves
```

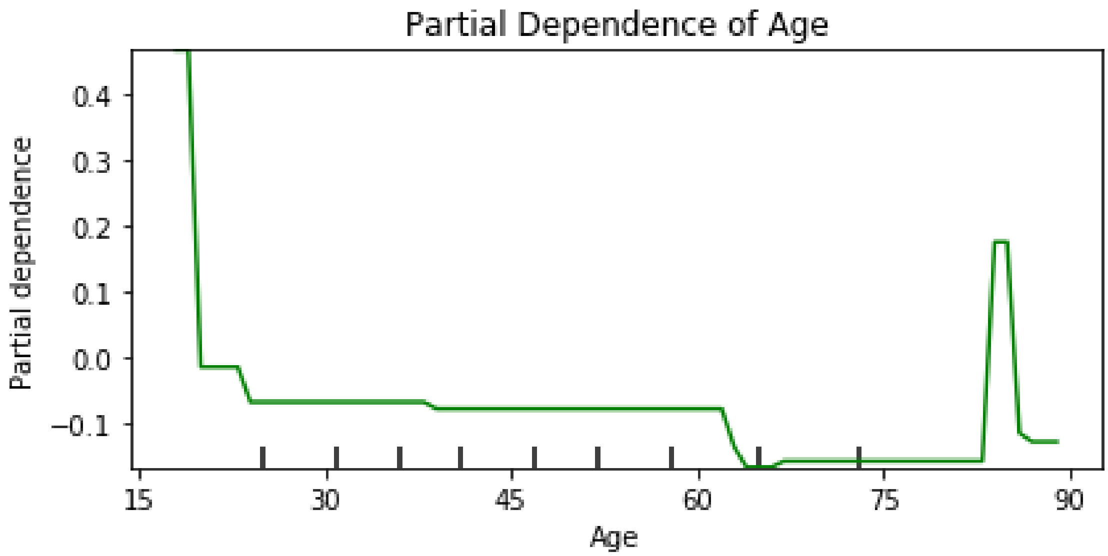
```
    model_fpr, model_tpr, _ = roc_curve(y_test, model.predict(X_test))
    # plot the roc curve for the model
    plt.plot(model_fpr, model_tpr, marker='.', label=label)
    # axis labels
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    # show the legend
    plt.legend()

def plot_random_classifier():
    plt.figure(figsize=(10,8))
    # generate a random prediction line
    random_probs = [0 for _ in range(len(y_test))]
    # calculate scores
    random_auc = roc_auc_score(y_test, random_probs)
    random_fpr, random_tpr, _ = roc_curve(y_test, random_probs)
    plt.plot(random_fpr, random_tpr, linestyle='--', label='Random Classifier')

plot_random_classifier()

get_roc_auc(GB_final, 'boosting')
```

```
boosting: AUC = 0.778
```



6.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.).

[79]:
```
# ppd of age
features = [0]
plot_partial_dependence(GB_final, X_train, features)
plt.xlabel('Age')
plt.title('Partial Dependence of Age');
```

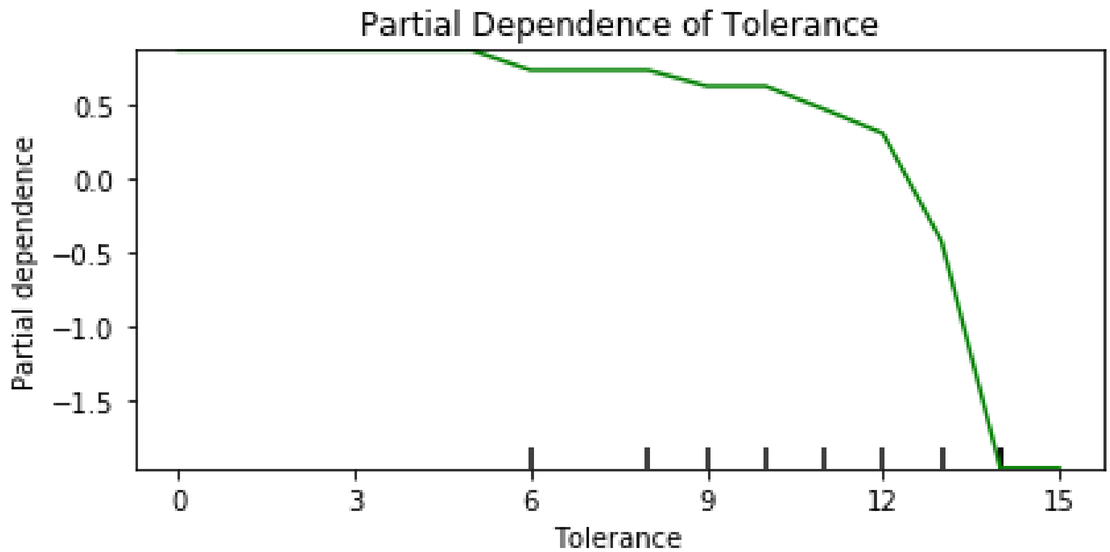


This graph illustrates the partial marginal influence of age on the prediction. As the graph suggests, when age is lower than 18, age seems to have positive marginal effect on the prediction, meaning that as people get older before 18, they think racist teachers should be able to teach. The partial dependence stays below 0 except spiking up at about 80 years old, showing that as people get older from 18 to 80, people increasingly believe that racist teachers should not teach. At around 80 years old, this pattern changes for some reason.

[82]:
```
X_train.columns.get_loc('tolerance')
```

[82]: 32

[83]:
```
# ppd of tolerance
features = [32]
plot_partial_dependence(GB_final, X_train, features)
plt.xlabel('Tolerance')
plt.title('Partial Dependence of Tolerance');
```



This graph illustrates the partial marginal influence of tolerance on the prediction. As tolerance increases, it tends to lead to higher probability of thinking racist teachers shouldn't be teaching.