Tree-based Inference

In [1]:

```
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 pdpbox import pdp, info_plots
```

Conceptual: Cost functions for classification trees

Q1 (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?

A1. Gini index and cross entropy are better when growing the tree as these two methods often lead to high labelling purity, in other words, they minimize misclassification. The gini index can have control on variance and avoid overfitting. By contrast, classification error is used to maximize accuracy and minimize test error rate. Therefore, it is often used when pruning the tree.

Application: Predicting attitudes towards racist college professors

Q2.Estimate the following models, predicting colrac using the training set (the training .csv) with 10-fold CV

In [5]:

```
train_df = pd.read_csv('data/gss_train.csv')
test_df = pd.read_csv('data/gss_test.csv')

X_train = train_df.drop(['colrac'], axis=1)
y_train = train_df['colrac']
X_test = test_df.drop(['colrac'], axis=1)
y_test = test_df['colrac']
```

In [64]:

```
def model_pred(model):
    model_acc = cross_val_score(model,
                                X_train, y_train,
                                cv=10, scoring='accuracy')
    model_rocauc = cross_val_score(model,
                                   X_train, y_train,
                                   cv=10, scoring='roc_auc')
    print('The error rate for the model is: ',1-np.mean(model_ac
c))
    print('The roc/auc for the model is: ',np.mean(model_rocauc)
)
```

In [65]:

```
#1.Logistic regression
lr = LogisticRegression(solver='liblinear', random_state=0)
model_pred(lr)
```

```
The error rate for the model is:  0.2073195576071894
5
The roc/auc for the model is:  0.8703107556427476
```

In [95]:

```
#2.Naive Bayes
nb = GaussianNB()
model_pred(nb)
```

```
The error rate for the model is:  0.2655525097759038
3
The roc/auc for the model is:  0.8080500250922787
```

In [40]:

```
#3.Elastic net regression
en = ElasticNetCV(cv=10).fit(X_train, y_train)
print('Alpha for ElasticNet model is:',en.alpha_)
print('Ratio for ElasticNet model is:',en.l1_ratio)
```

```
Alpha for ElasticNet model is: 0.0038452641680228584
Ratio for ElasticNet model is: 0.5
```

In [66]:

```
#Tuned Elastic net regression
en_tuned = ElasticNet(alpha=0.00385, l1_ratio=0.5)
print('The error rate 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_tuned, X_train, y_train,
                        scoring = 'roc_auc', cv=10)))
```

```
The error rate for elastic net is: 0.147145477829692
06
The roc/auc for elastic net is: 0.874041081159952
```

In [71]:

```
#4.Decision tree (CART)
param_grid = {
'max_features': ['auto', 'sqrt']
}
dt_grid = GridSearchCV(DecisionTreeClassifier(random_state=0),
                       param_grid = param_grid, cv = 10)
dt_grid.fit(X_train,y_train)
dt_grid.best_params_
```

Out[71]:

```
{'max_features': 'auto'}
```

In [73]:

```
dt = DecisionTreeClassifier(max_features = 'auto')
model_pred(dt)
```

```
The error rate for the model is:  0.31910567600884
The roc/auc for the model is:  0.6976305150248813
```

In [54]:

```
#5.Bagging: Tune hyperparameters 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)
bg_grid.best_params_
```

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

[Parallel(n_jobs=-1)]: Using backend LokyBackend wit
h 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed
:    4.4s
[Parallel(n_jobs=-1)]: Done  60 out of  60 | elapsed
:    8.1s finished
```

Out[54]:

```
{'n_estimators': 40}
```

In [55]:

```
bg_tuned = BaggingClassifier(
    n_estimators=bg_grid.best_params_['n_estimators'])
model_pred(bg_tuned)
```

```
The error rate for the model is:  0.2256685188132936
4
The roc/auc for the model is:  0.8704921687969979
```

In [58]:

```
#6.Random forest: Tune hyperparameters from random_grid
param_grid = {
'n_estimators': [5, 10, 20, 30, 40, 50],
'max_depth': [1, 5, 10, 30, 50, 80, 100],
'max_features': [5, 10, 20, 30, 40, 50]}
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)
rf_grid.best_params_
```

```
Fitting 10 folds for each of 252 candidates, totalli
ng 2520 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend wit
h 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done 208 tasks      | elapsed
:    5.0s
[Parallel(n_jobs=-1)]: Done 730 tasks      | elapsed
:   24.2s
[Parallel(n_jobs=-1)]: Done 1136 tasks      | elapse
d:   45.2s
[Parallel(n_jobs=-1)]: Done 1702 tasks      | elapse
d:  1.2min
[Parallel(n_jobs=-1)]: Done 2432 tasks      | elapse
d:  1.9min
[Parallel(n_jobs=-1)]: Done 2520 out of 2520 | elaps
ed:  2.0min finished
```

Out[58]:

```
{'max_depth': 10, 'max_features': 5, 'n_estimators':
40}
```

In [59]:

```
rf_tuned = RandomForestClassifier(n_estimators=40,
                                  max_depth=10,
                                  max_features=5)
model_pred(rf_tuned)
```

```
The error rate for the model is:  0.21276173447793778
The roc/auc for the model is:  0.8768558001354781
```

In [60]:

```
#7.Boosting
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)
gb_grid.best_params_
```

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

[Parallel(n_jobs=-1)]: Using backend LokyBackend wit
h 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed
:    5.6s
[Parallel(n_jobs=-1)]: Done 480 out of 480 | elapsed
:   14.4s finished
```

Out[60]:

```
{'learning_rate': 0.3, 'max_features': 'sqrt', 'n_es
timators': 40}
```

In [61]:

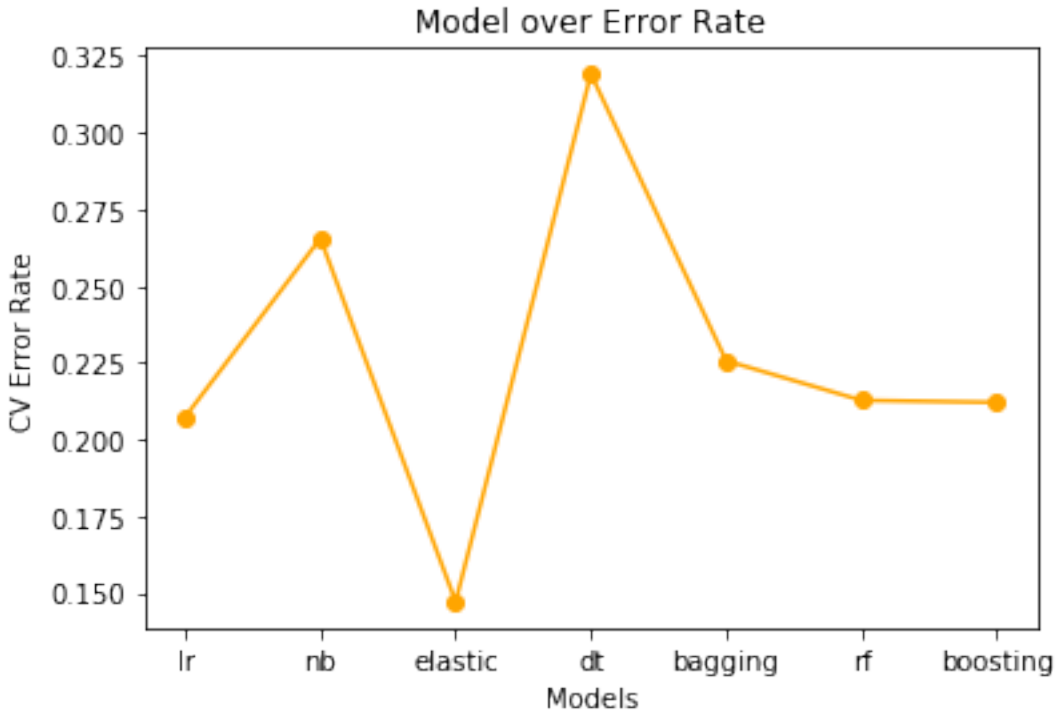
```
gb_tuned = GradientBoostingClassifier(learning_rate=0.3,
                                      max_features='sqrt',
                                      n_estimators=40)
model_pred(gb_tuned)
```

```
The error rate for the model is:  0.2120951899715082
2
The roc/auc for the model is:  0.8761125460723045
```

Q3.(20 points) Compare and present each model's (training) performance based on Cross-validated error rate and ROC/AUC

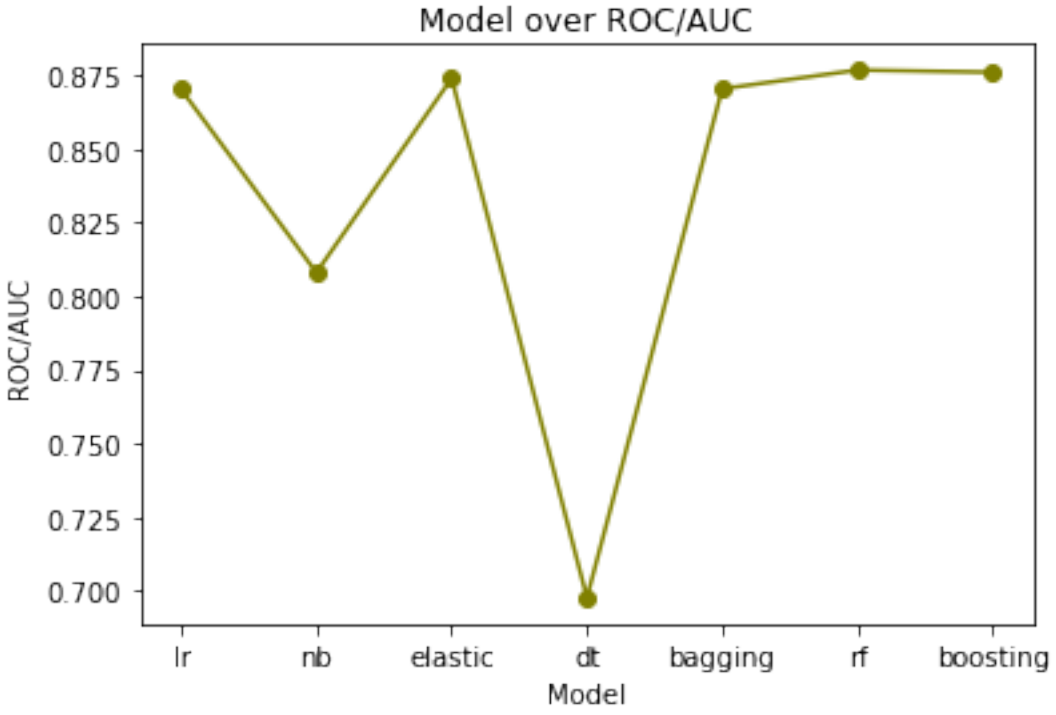
In [77]:

```
#Error rate performance
error_rate = [0.20731955760718945,0.26555250977590383, 0.1471454
7782969206, 0.31910567600884,0.2256685188132936 4, 0.212761734477
93778,0.212095189971508 22]
model = ['lr', 'nb', 'elastic', 'dt', 'bagging', 'rf', 'boosting
']
plt.plot(model, error_rate, marker='o', color='orange')
plt.xlabel('Models')
plt.ylabel('CV Error Rate')
plt.title('Model over Error Rate');
```



In [80]:

```
#ROC/AUC performance
roc_auc = [0.8703107556427476, 0.8080500250922787,
           0.874041081159952, 0.6976305150248813,
           0.8704921687969979, 0.8768558001354781,
           0.8761125460723045]
model = ['lr', 'nb', 'elastic', 'dt', 'bagging',
         'rf', 'boosting']
plt.plot(model, roc_auc, marker='o', color='Olive')
plt.xlabel('Model')
plt.ylabel('ROC/AUC')
plt.title('Model over ROC/AUC');
```



Q4 .(15 points) Which is the best model? Defend your choice.

A4. Considering both error rate and ROC/AUC, Random forest is the best model as it has fairly high ROC/AUC and low error rate. Boosting and Elastic Net are also good models.

Q5.(15 points) 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?

In [93]:

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
print('rf error rate', 1 - accuracy_score(
    y_test,rf_tuned.fit(X_train,
                        y_train).predict(X_test)))
print('rf roc/auc', roc_auc_score(
    y_test,rf_tuned.fit(X_train,
                        y_train).predict(X_test)))

print('dt error rate', 1 - accuracy_score(
    y_test,dt.fit(X_train,
                  y_train).predict(X_test)))
print('dt roc/auc', roc_auc_score(
    y_test,dt.fit(X_train,
                  y_train).predict(X_test)))

print('gb error rate', 1 - accuracy_score(
    y_test,gb_tuned.fit(X_train,
                        y_train).predict(X_test)))
print('gb roc/auc', roc_auc_score(
    y_test,gb_tuned.fit(X_train,
                        y_train).predict(X_test)))
```

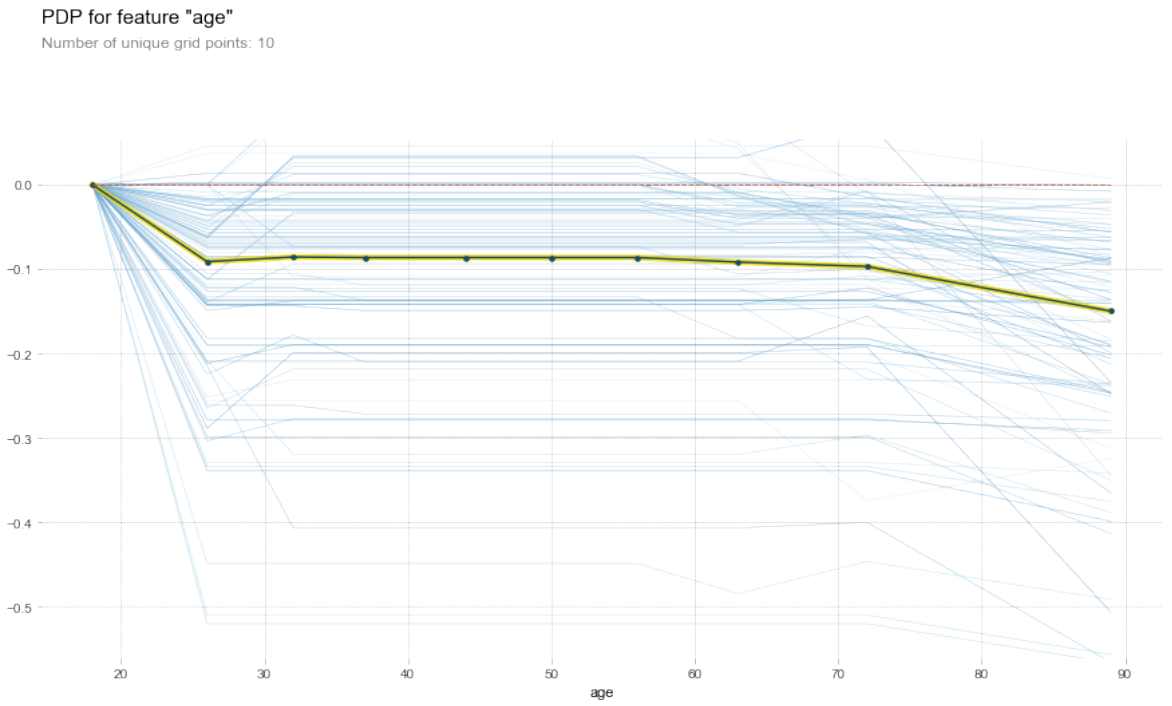
```
rf error rate 0.215010141987829 64
rf roc/auc 0.7550562727573651
dt error rate 0.3164300202839757
dt roc/auc 0.6570423700761338
gb error rate 0.2089249492900609
gb roc/auc 0.7861138695796093
```

Random forest still performs good with the test set, but the roc/auc decreases a bit and the model does not remain to be the 'best'. By contrast, the boosting model performs better in the test set. Therefore, random forest cannot be generalized to be the 'best' but remains to be a good choice.

Q6.Bonus: PDPs/ICE

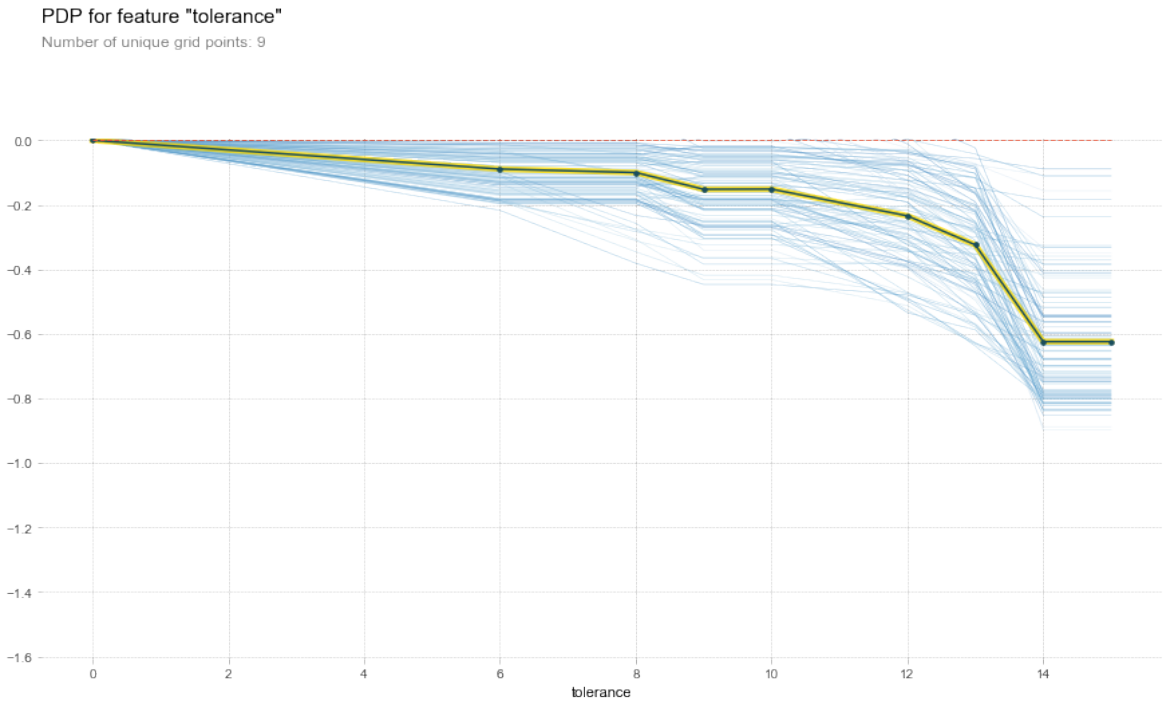
In [99]:

```
age = pdp.pdp_isolate(model=gb_tuned, dataset=X_train,
                      model_features=X_train.columns,feature='age
')
fig, axes = pdp.pdp_plot(age,'age',plot_lines=True,frac_to_plot=
100)
```



In [100]:

```
tolerance = pdp.pdp_isolate(model=gb_tuned, dataset=X_train,
                    model_features=X_train.columns,feature='tol
erance')
fig, axes = pdp.pdp_plot(tolerance,'tolerance',plot_lines=True,f
rac_to_plot=100)
```



I used Gradient Boosting to predict the results and used PDP to show the marginal effect of feature. When age is between 0 to 18, it decreases from positive to negative correlation, when it is between 18 to 26, it decreases sharply. But when the age continues to increase, the effect becomes stable. Then when the age is larger than 72, the effect goes down again. This means that people who aged over 18 does not support racists to be teachers. But people who aged under 18 can accept racist teachers in general. I interpret this as people becomes more aware of the possible consequence of having a racist professor.

The tolerance graph tells us that tolerance has a positive effect on colrac. When tolerance level is between 0 to 12, it decreases gradually, but after tolerance level = 12, it decreases rapidly. This means that when tolerance goes high, the racist teachers are widely accepted.

In [ ]: