modelhw5

March 1, 2020

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


from time import time
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold, cross_val_score,  GridSearchCV
from sklearn.linear_model import LogisticRegression, Ridge, Lasso, ElasticNet,␣
 ↪SGDClassifier
from sklearn.naive_bayes import GaussianNB, BernoulliNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier,␣
 ↪GradientBoostingClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
from sklearn.metrics import mean_squared_error as mse
from pyearth import Earth
from skater.model import InMemoryModel
from sklearn.inspection import plot_partial_dependence

%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

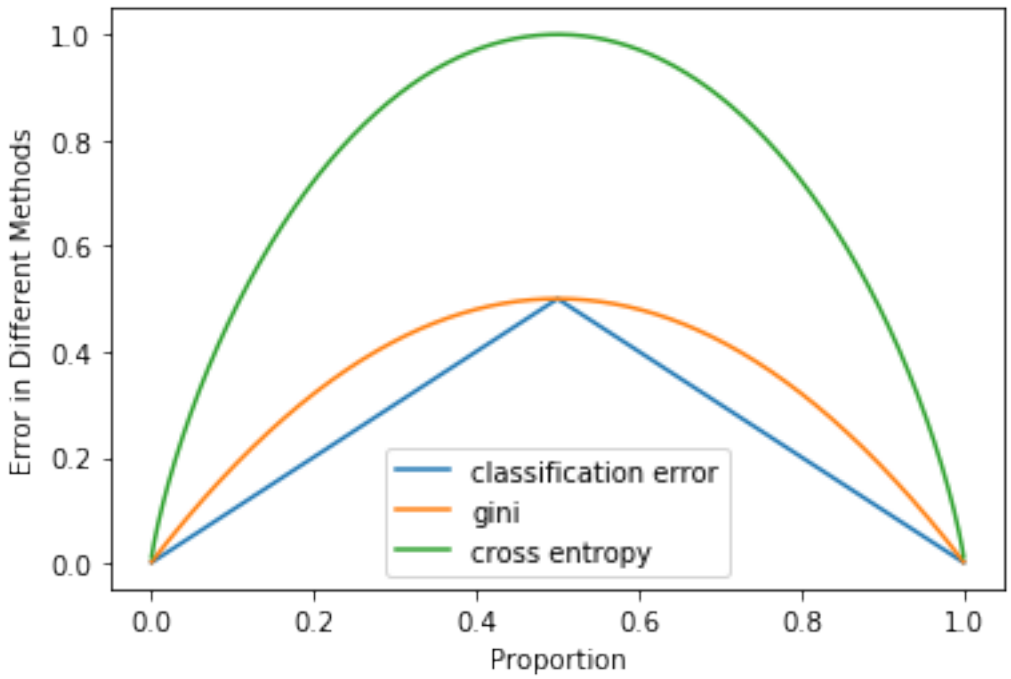
0.1 Conceptual: Cost functions for classification trees

0.1.1 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?

[2]:
```
p = np.linspace(0, 1, num=1000)[1:-1]
clf_err = [1 - max(i, 1-i) for i in p]
gini_err = [2*i*(1-i) for i in p]
```

```
ce_err = [-(i*np.log2(i)+(1-i)*np.log2(1-i)) for i in p]


plt.plot(p,clf_err,label='classification error')
plt.plot(p,gini_err,label='gini')
plt.plot(p,ce_err,label='cross entropy')
plt.ylabel('Error in Different Methods')
plt.xlabel('Proportion')
plt.legend()
plt.show()
```



When growing a decision tree, we would prefer Gini index or cross-entropy becuase they are more sensitive to node purity. When pruning a tree, we would use classification error rate because we have to make sure the trained decision tree is not overfitting the training set data, so as to have a more accurate prediction.

0.2 Application: Predicting attitudes towards racist college professors

[68]:
```
# Load the data
gss_train = pd.read_csv("gss_train.csv")
gss_test = pd.read_csv('gss_test.csv')

#get training/test dataset
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']

models = []
```

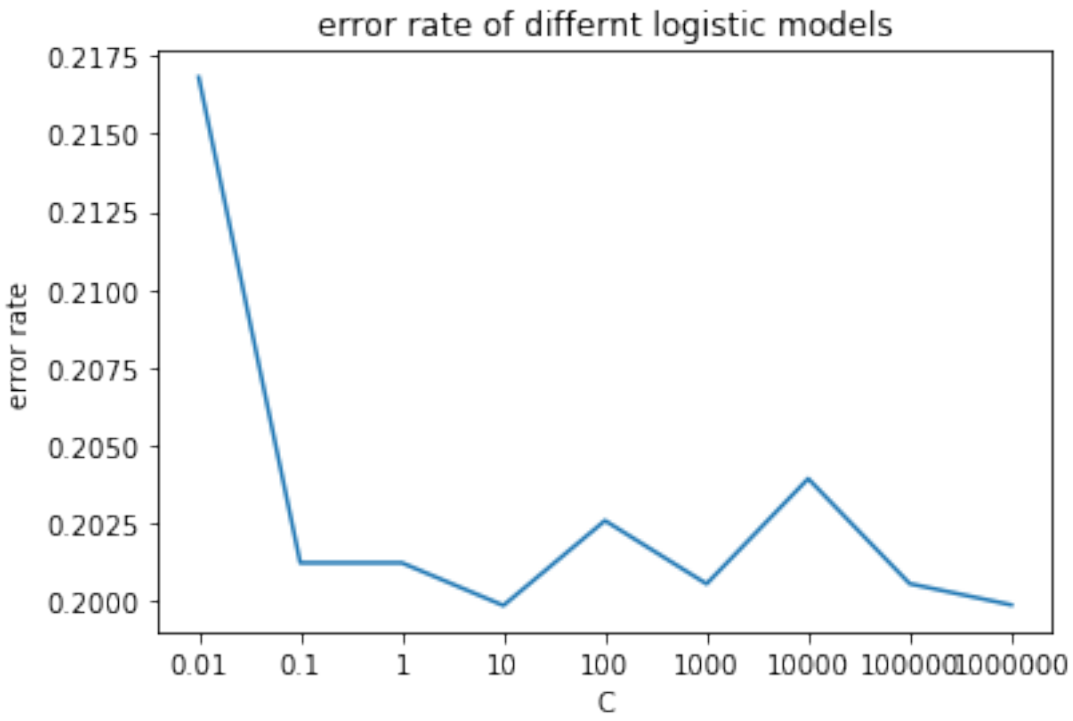
0.2.1 Estimate the models

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

0.2.2 Logistic Regression

[69]:
```
# tuning the logistic regression with different C parameters
errors_logit=[]
c_lst = [0.01, 0.1, 1, 10, 100, 1000,10000,100000,1000000]
for c in c_lst:
    model=LogisticRegression(C=c)
    error = 1-np.mean(cross_val_score(model, x_train, y_train, cv=KFold(10),␣
 ↪scoring='accuracy'))
    errors_logit.append(error)

# plot the error rate of logistic models
xrange= np.arange(1, len(c_lst)+ 1)
plt.plot(np.arange(1, len(c_lst)+ 1), errors_logit)
plt.title('error rate of differnt logistic models')
plt.xlabel('C')
plt.ylabel('error rate')
plt.xticks(xrange, c_lst)
plt.show()
print("The error rate is the smallest at C={}".format(c_lst[np.
 ↪argmin(errors_logit)]))
```



The error rate is the smallest at C=10

[70]:
```
models.append(LogisticRegression(C=c_lst[np.argmin(errors_logit)]))
```

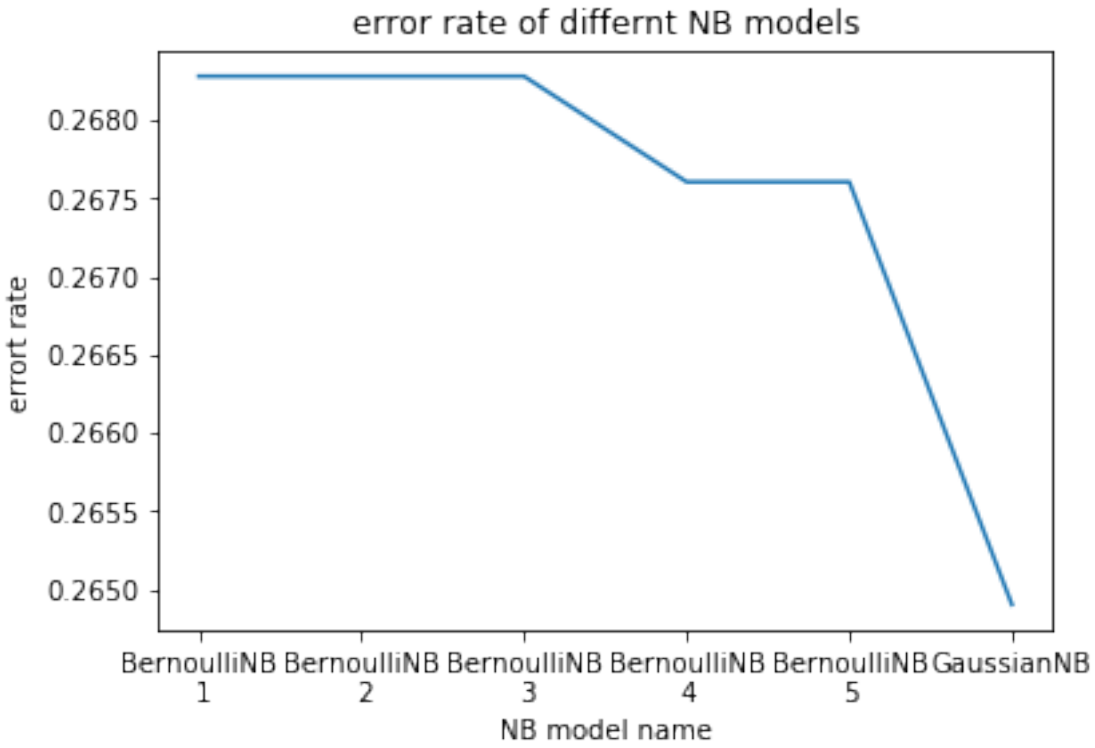
0.2.3 Naive Bayes

[71]:
```
# tuning NB models with different alpha
errors_nb = []
names_nb=[]
for i in range(1, 6):
    errors_nb.append(BernoulliNB(alpha=i))
    names_nb.append('BernoulliNB\n {}'.format(i))
errors_nb.append(GaussianNB())
names_nb.append('GaussianNB')
errors_nb= [(1-np.mean(cross_val_score(model, x_train, y_train, cv=KFold(10),␣
 ↪scoring='accuracy')))for model in errors_nb]


xrange = np.arange(1, len(errors_nb) + 1)
plt.plot(xrange, errors_nb)
plt.title('error rate of differnt NB models')
plt.xticks(xrange, names_nb)
plt.xlabel('NB model name')
```

```
plt.ylabel('errort rate')
plt.show()

print("The error rate is the smallest when the model is {}".format(names_nb[np.
 ↪argmin(errors_nb)]))
```




```
The error rate is the smallest when the model is GaussianNB
```

[72]:
```
models.append(GaussianNB())
```

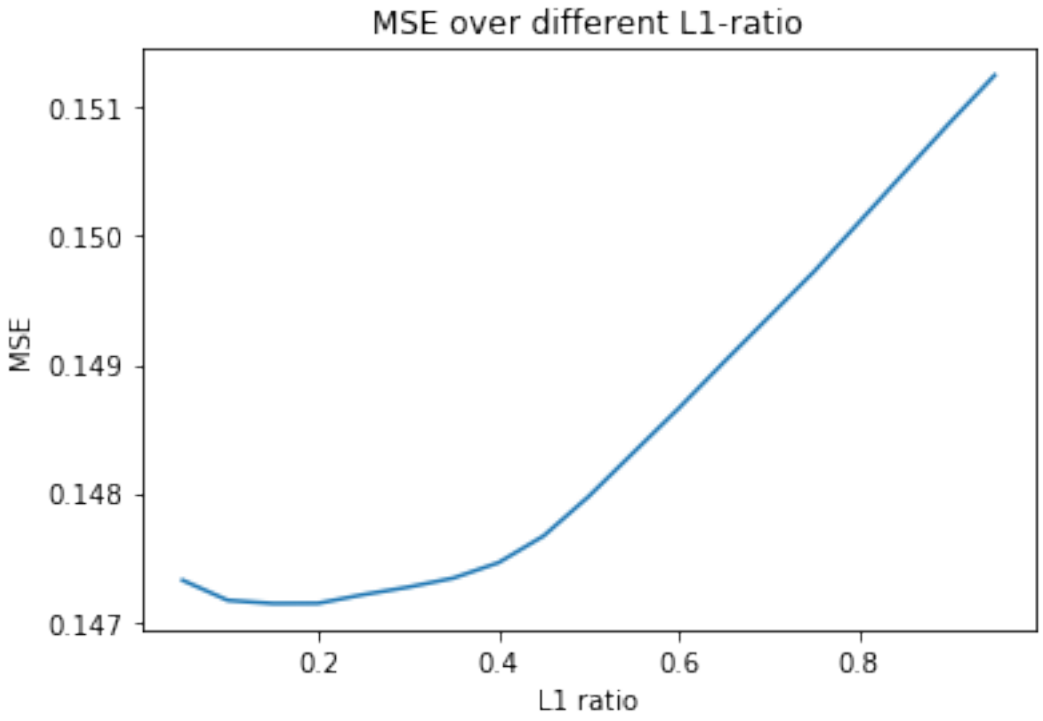
0.2.4 Elastic net

[73]:
```
mses_en = []
opt_alpha = []
alpha_lst = np.logspace(-2, 1, 30);
ratio_lst = np.arange(0.05, 1, 0.05)
for r in ratio_lst:
    mses = []
    for a in alpha_lst:
        model = ElasticNet(alpha=a, l1_ratio=r)
        mse = -np.mean(cross_val_score(model, x_train, y_train, cv=KFold(10),␣
 ↪scoring='neg_mean_squared_error'))
```

```
        mses.append(mse)
    idx = np.argmin(mses)
    mses_en.append(mses[idx])
    opt_alpha.append(alpha_lst[idx])


plt.plot(ratio_lst, mses_en)
plt.title("MSE over different L1-ratio")
plt.xlabel("L1 ratio")
plt.ylabel("MSE")
plt.show()

idx = np.argmin(mses_en)
print("The MSE is the smallest when L1-ratio={:.2f} and alpha={}".
 ↪format(ratio_lst[idx], opt_alpha[idx]))
```




```
The MSE is the smallest when L1-ratio=0.15 and alpha=0.01
```

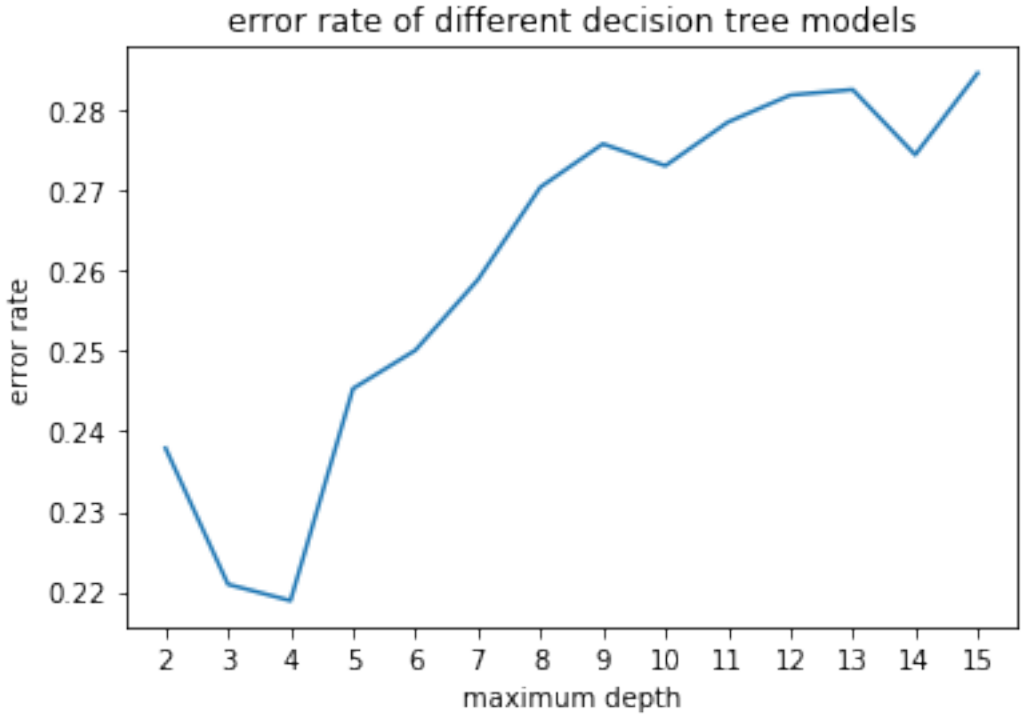
[74]:
```
models.append(ElasticNet(alpha=opt_alpha[idx], l1_ratio=ratio_lst[idx]))
```

0.2.5 Decision Tree

[75]:
```
errors_dt = []
for d in range(2, 16):
    model = DecisionTreeClassifier(max_depth=d)
    error = 1-np.mean(cross_val_score(model, x_train, y_train, cv=KFold(10),␣
 ↪scoring='accuracy'))
    errors_dt.append(error)

plt.plot(range(2, 16), errors_dt)
plt.title('error rate of different decision tree models')
plt.xlabel("maximum depth")
plt.ylabel("error rate")
plt.xticks(np.arange(2, 16))
plt.show()

print("The error rate is the smallest when d={}".format(np.argmin(errors_dt)+2))
```




```
The error rate is the smallest when d=4
```

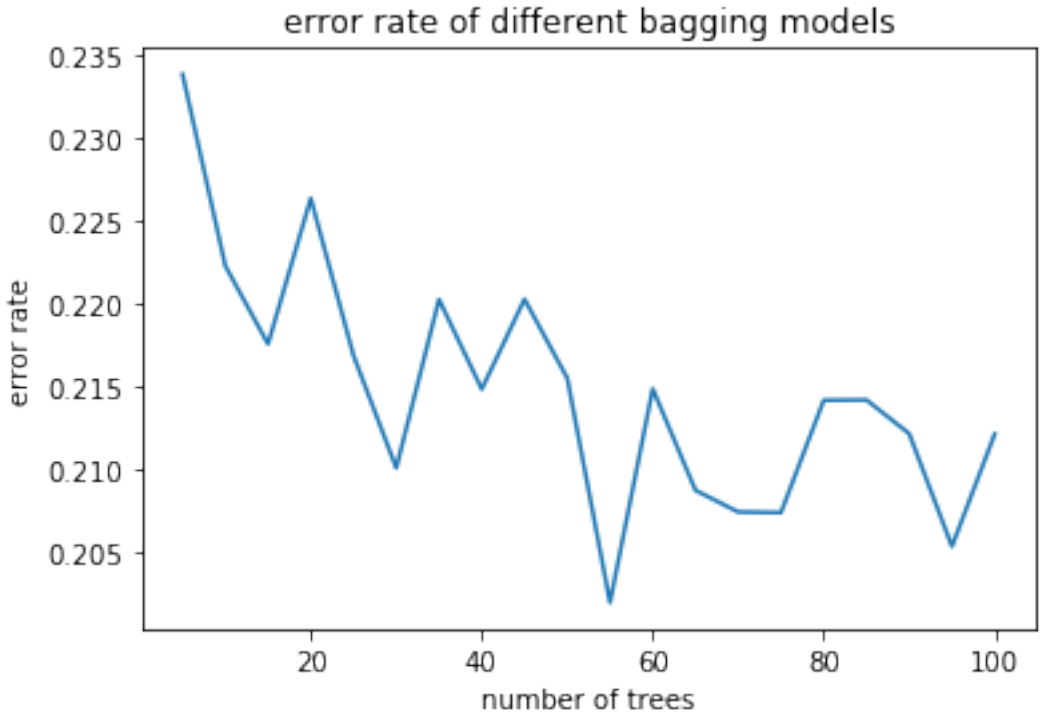
[76]:
```
models.append(DecisionTreeClassifier(max_depth=np.argmin(errors_dt)+2))
```

0.2.6 Bagging

[77]:
```
errors_bagging = []
n_tree = np.arange(5, 101, 5)
for n in n_tree:
    model = BaggingClassifier(n_estimators=n)
    error = 1-np.mean(cross_val_score(model, x_train, y_train, cv=KFold(10),␣
 ↪scoring='accuracy'))
    errors_bagging.append(error)

plt.plot(n_tree, errors_bagging)
plt.title('error rate of different bagging models')
plt.xlabel("number of trees")
plt.ylabel("error rate")
plt.show()

print("The error rate is the smallest when n={}".format(n_tree[np.
 ↪argmin(errors_bagging)]))
```




```
The error rate is the smallest when n=55
```

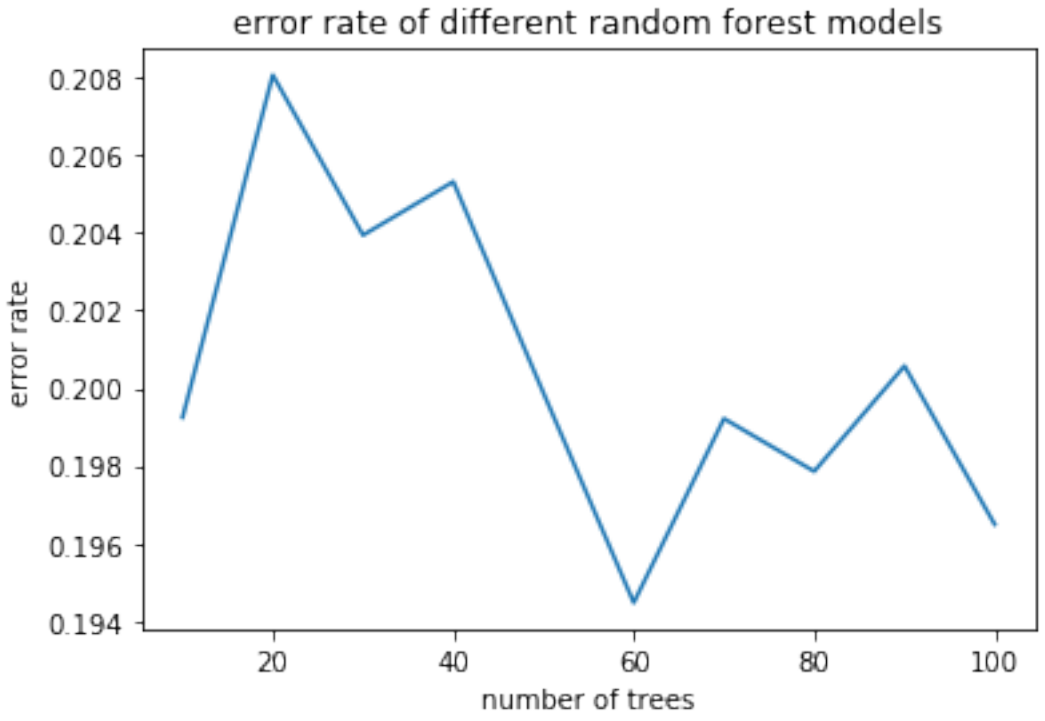
[78]:
```
models.append(BaggingClassifier(n_estimators=n_tree[np.argmin(errors_bagging)]))
```

0.2.7 Random forest

[79]:
```
n_tree = np.arange(10, 101, 10)
max_depth = np.arange(2, 11)
errors_rf = []
opt_depth = []
for n in n_tree:
    errors = []
    for d in max_depth:
        model = RandomForestClassifier(n_estimators=n, max_depth=d)
        error = 1-np.mean(cross_val_score(model, x_train, y_train,␣
 ↪cv=KFold(10), scoring='accuracy'))
        errors.append(error)
    idx = np.argmin(errors)
    errors_rf.append(errors[idx])
    opt_depth.append(max_depth[idx])

plt.plot(n_tree, errors_rf)
plt.title('error rate of different random forest models')
plt.xlabel("number of trees")
plt.ylabel("error rate")
plt.show()

opt_idx = np.argmin(errors_rf)
print("The error rate is the smallest when n={} and max-depth={}".
 ↪format(n_tree[opt_idx], opt_depth[opt_idx]))
```



```
The error rate is the smallest when n=60 and max-depth=9
```

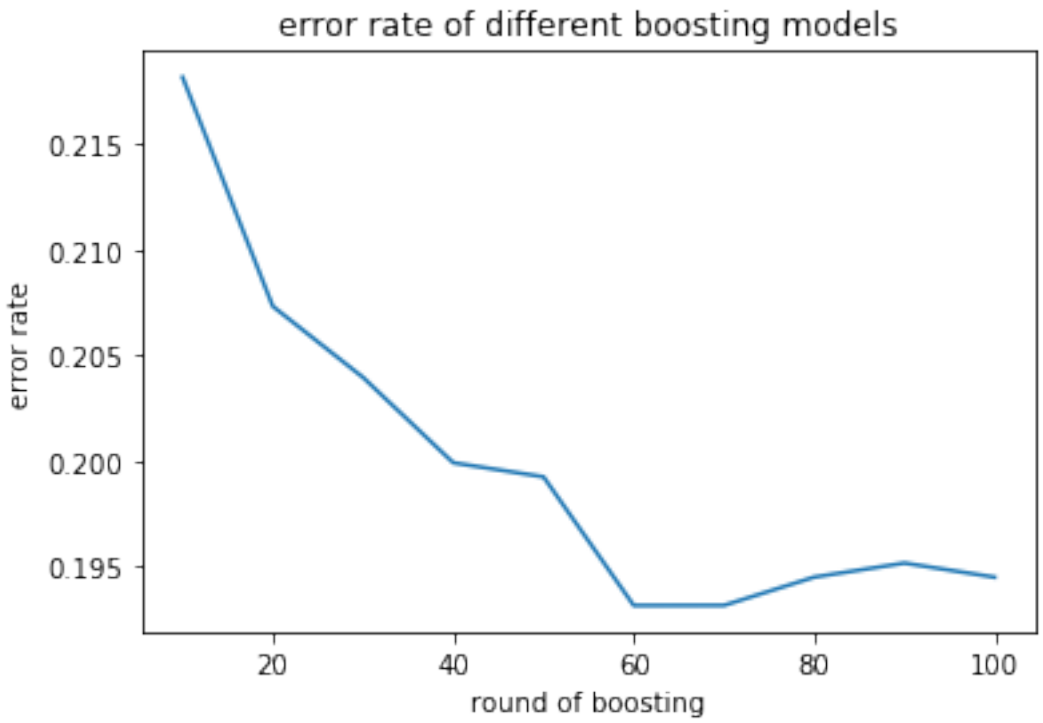
[80]:
```
models.append(RandomForestClassifier(n_estimators=n_tree[opt_idx],␣
↪max_depth=opt_depth[opt_idx]))
```

0.2.8 Boosting

[81]:
```
n_tree = np.arange(10, 101, 10)
max_depth = np.arange(2, 11)
errors_xgboost = []
opt_depth = []
for n in n_tree:
    errors = []
    for d in max_depth:
        model = xgb.XGBClassifier(objective="binary:logistic", n_estimators=n,␣
↪max_depth=d)
        error = 1-np.mean(cross_val_score(model, x_train, y_train,␣
↪cv=KFold(10), scoring='accuracy'))
        errors.append(error)
    idx = np.argmin(errors)
    errors_xgboost.append(errors[idx])
    opt_depth.append(max_depth[idx])

plt.plot(n_tree, errors_xgboost)
plt.title('error rate of different boosting models')
plt.xlabel("round of boosting")
plt.ylabel("error rate")
plt.show()

opt_idx = np.argmin(errors_xgboost)
print("The error rate is the smallest when n={} and max-depth={}".
↪format(n_tree[opt_idx], opt_depth[opt_idx]))
```



The error rate is the smallest when n=60 and max-depth=3

[82]:
```
models.append(xgb.XGBClassifier(objective="binary:logistic",␣
↪n_estimators=n_tree[opt_idx], max_depth=opt_depth[opt_idx]))
```

0.3 Evaluate the models

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

a. Cross-validated error rate

b. ROC/AUC

[83]:
```
models  #have a look at the models  #elasticnet is model[2]
```

[83]:
```
[LogisticRegression(C=10, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                    warm_start=False),
 GaussianNB(priors=None, var_smoothing=1e-09),
 ElasticNet(alpha=0.01, copy_X=True, fit_intercept=True,
            l1_ratio=0.15000000000000002, max_iter=1000, normalize=False,
            positive=False, precompute=False, random_state=None,
            selection='cyclic', tol=0.0001, warm_start=False),
```

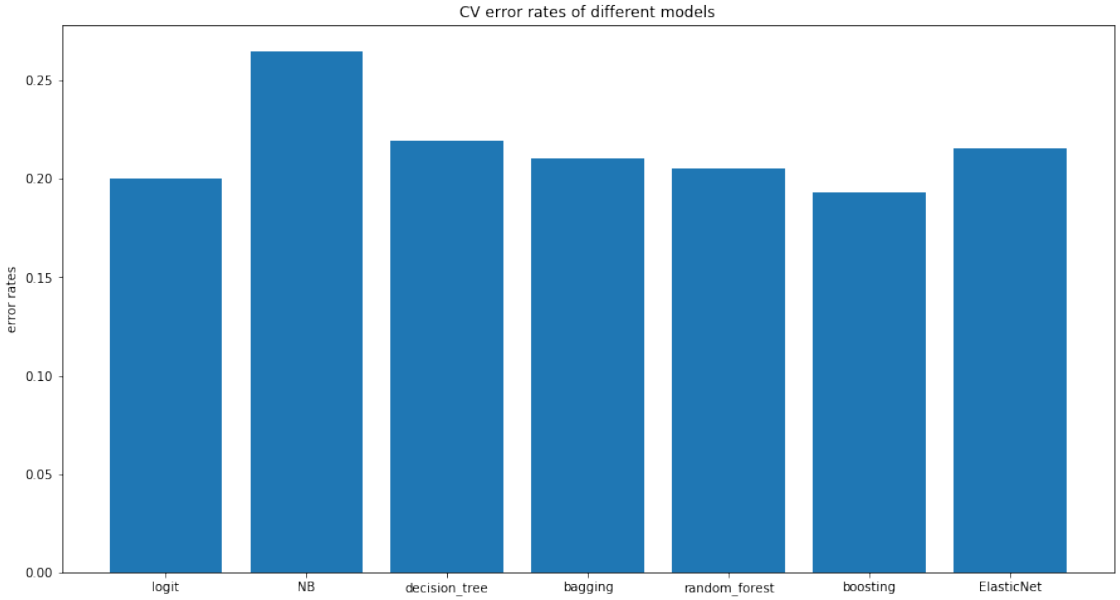
```
 DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                        max_depth=4, max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, presort='deprecated',
                        random_state=None, splitter='best'),
 BaggingClassifier(base_estimator=None, bootstrap=True,
bootstrap_features=False,
                   max_features=1.0, max_samples=1.0, n_estimators=55,
                   n_jobs=None, oob_score=False, random_state=None, verbose=0,
                   warm_start=False),
 RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                        criterion='gini', max_depth=9, max_features='auto',
                        max_leaf_nodes=None, max_samples=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=60,
                        n_jobs=None, oob_score=False, random_state=None,
                        verbose=0, warm_start=False),
 XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
               colsample_bynode=1, colsample_bytree=1, gamma=0,
               learning_rate=0.1, max_delta_step=0, max_depth=3,
               min_child_weight=1, missing=None, n_estimators=60, n_jobs=1,
               nthread=None, objective='binary:logistic', random_state=0,
               reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
               silent=None, subsample=1, verbosity=1)]
```

[84]:
```
#cv error rate and auc calculation for models except ElasticNet
errors = [1-np.mean(cross_val_score(model, x_train, y_train, cv=KFold(10),␣
 ↪scoring='accuracy'))
             for model in models[:2]+models[3:]]
aucs = [np.mean(cross_val_score(model, x_train, y_train, cv=KFold(10),␣
 ↪scoring='roc_auc'))
          for model in models[:2]+models[3:]]
```

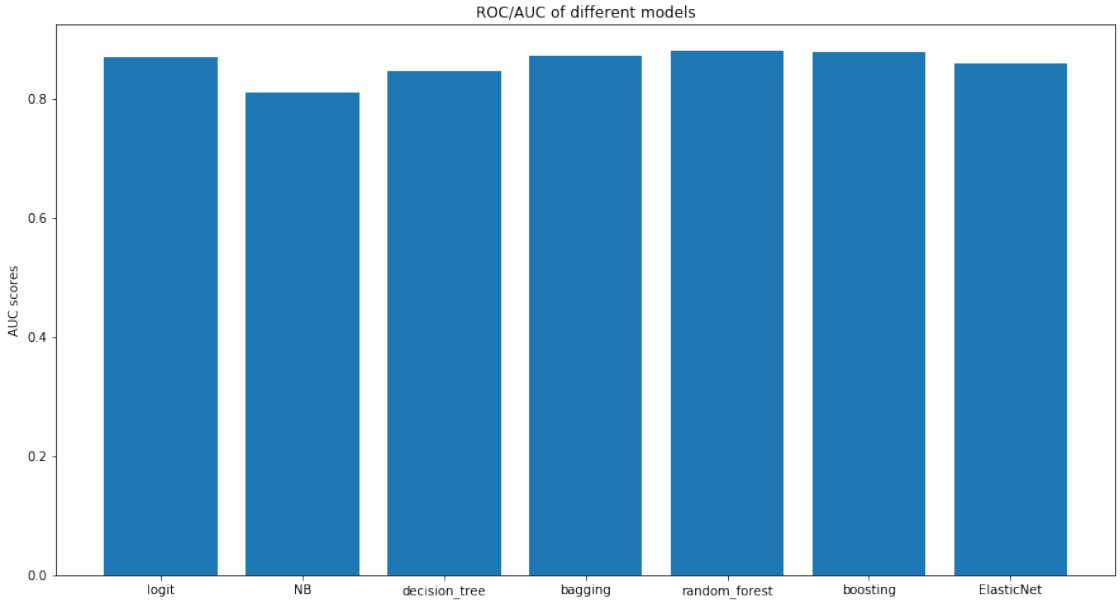
[85]:
```
# cv error rate and auc calculation for ElasticNet model
en=models[2]
np.random.seed(124)
idx = np.random.choice(5, x_train.shape[0])
en_errors = []
en_aucs = []
for i in range(5):
    x = x_train[idx!=i]
    y = y_train[idx!=i]
    pred = 1*(en.fit(X=x, y=y).predict(x_train[idx==i])>0.5)
    y_true = y_train[idx==i]
    en_errors.append(1 - accuracy_score(y_true, pred))
```

```
    en_aucs.append(roc_auc_score(y_true, pred))
errors.append(np.mean(errors))
aucs.append(np.mean(aucs))
```

[86]:
```
plt.figure(figsize=(15,8))
plt.bar(range(len(models)), errors)
plt.ylabel('error rates')
plt.title('CV error rates of different models')
plt.xticks(range(len(models)), ('logit', 'NB', 'decision_tree'
                                , 'bagging', 'random_forest', 'boosting',␣
 ↪'ElasticNet'))
plt.show()
```



[87]:
```
plt.figure(figsize=(15,8))
plt.bar(range(len(models)), aucs)
plt.ylabel('AUC scores')
plt.title('ROC/AUC of different models')
plt.xticks(range(len(models)), ('logit', 'NB', 'decision_tree'
                                , 'bagging', 'random_forest', 'boosting',␣
 ↪'ElasticNet'))
plt.show()
```



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

The Boosting model has the lowest error rates of all, while the Random Forest Model hits the top ROC/AUC score. For me, A better should be featured with higher ROC/AUC because it can better deal with skewed sample distribution and would not overfit to a single class. Therefore, the best model is the Random Forest Model, with highest ROC/AUC(And it has the second lowest error rate as well).

0.4 Evaluate the best model

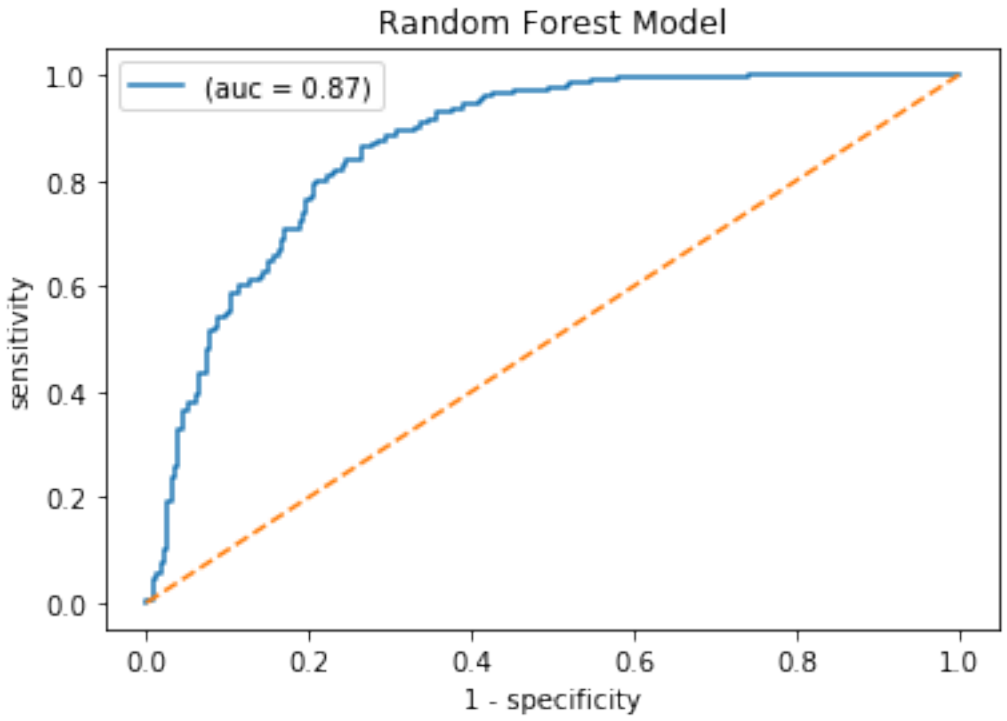
0.4.1 5. Evaluate the final, best model's (selected in 4) performance on the test set (the test .csv) by calculating and presenting the classification error rate and AUC. Compared to the fit evaluated on the training set in questions 3-4, does the "best" model generalize well? Why or why not? How do you know?

[96]:
```
best_model = models[5].fit(x_train, y_train) # we know models[5] is the random␣
↪forest model
```

[97]:
```
pred_prob =best_model.predict_proba(x_test)[:,1]
pred = best_model.predict(x_test)
error = 1 - accuracy_score(y_test, pred)  #error rate
roc_auc = roc_auc_score(y_test, pred_prob)  #roc/auc score


fpr, tpr, thresholds = roc_curve(y_test, pred_prob)
plt.plot(fpr, tpr, label='(auc = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], '--')
plt.xlabel('1 - specificity')
```

```
plt.ylabel('sensitivity')
plt.title('Random Forest Model')
plt.legend()
plt.show()
```



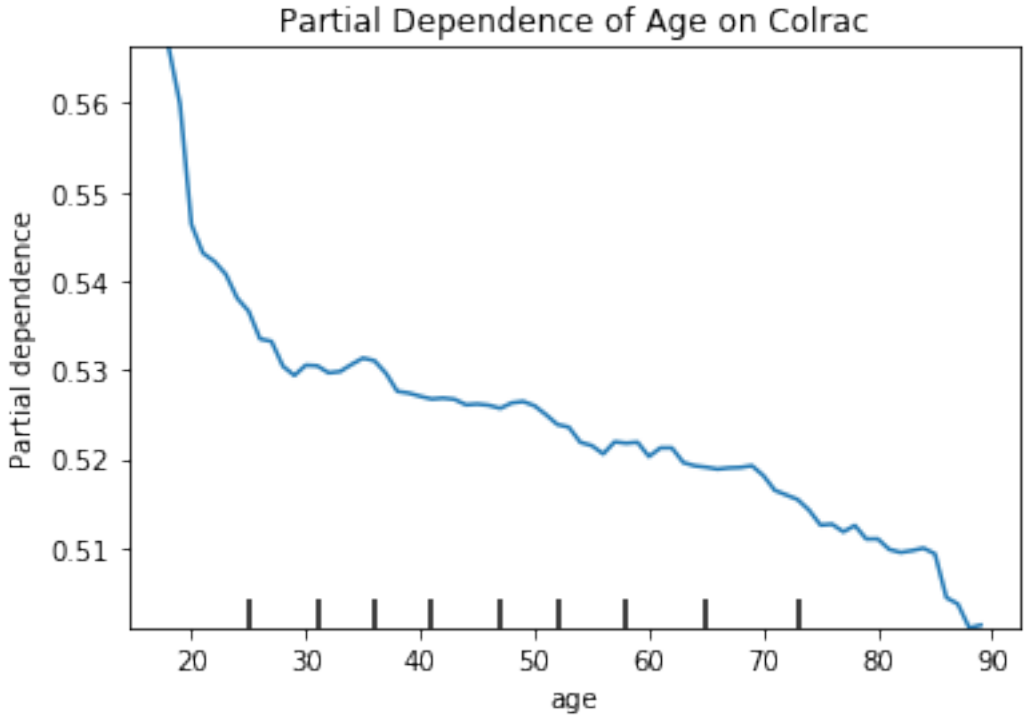
[98]:
```
print('''
The train error rate of Random Forest: {}
The train ROC/AUC of Random Forest: {}
The test error rate of Random Forest: {}
The test ROC/AUC of Random Forest: {}
'''.format(errors[4],aucs[4], error, roc_auc))
```

```
The train error rate of Random Forest: 0.20532726604155171
The train ROC/AUC of Random Forest: 0.8819256695316036
The test error rate of Random Forest: 0.2068965517241379
The test ROC/AUC of Random Forest: 0.8653757034094671
```

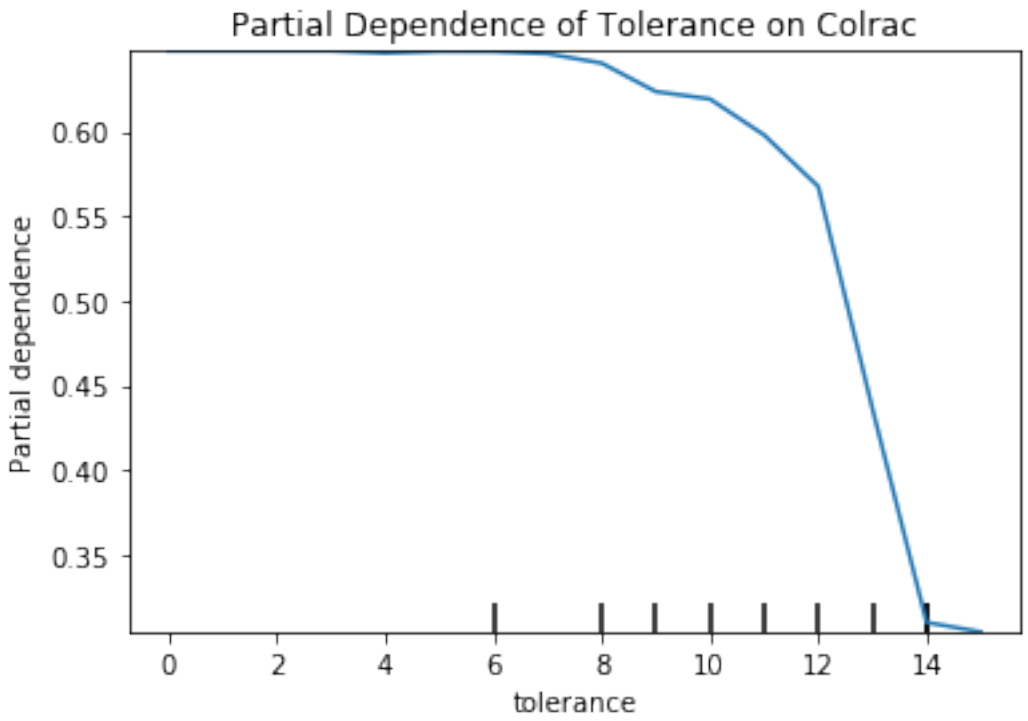
The errot rate and ROC/AUC of testing set are similar to training dataset. Therefore, the best model-Random Forest Model generalizes well to the test set.

0.4.2 6. (Up to 5 extra points) 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.).

[101]:
```
#partial dependence of age
age=[x_train.columns.get_loc('age')]
plot_partial_dependence(best_model, x_train, age)
plt.title('Partial Dependence of Age on Colrac');
```



[102]:
```
#partial dependece of tolerance
tol = [x_train.columns.get_loc('tolerance')]
plot_partial_dependence(best_model, x_train, tol)
plt.title('Partial Dependence of Tolerance on Colrac');
```



The graphs above indicate the effect of the **age**/**tolerance** on **colrac**. As **age** goes up, its partical dependence effect goes down sightly from 0.57 to 0.50, a small range, which means **age** might not be a strong predictor for **colrac**, but it still shows that when **age** increases, people increasingly believe racist teachers should not teach. As **tolerance** goes up, its partical dependence effect goes down more sharply than that of **age**, which means **tolerance** is a relatively stronger predictor for **colrac**, showing when **tolerance** gets larger, people are more likely to belive racist teachers should not teach.

[ ]: