MACS30100 Homework 5

Takuyo Ozaki

2/28/2020

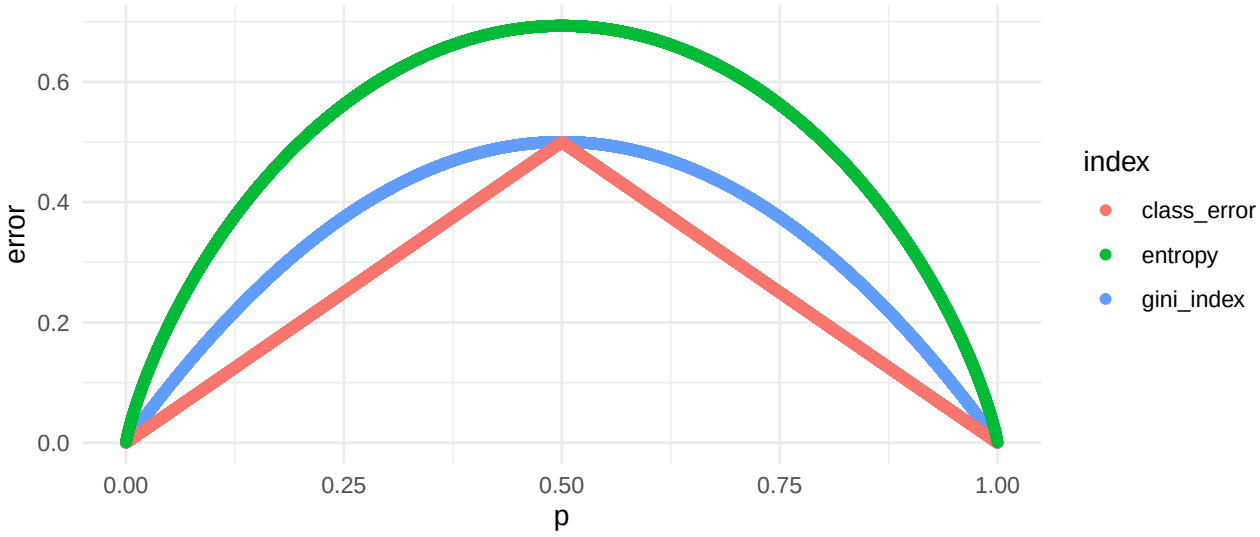
```
knitr::opts_chunk$set(message = FALSE, warning = FALSE)
knitr::opts_chunk$set(fig.width = 7, fig.height = 3, fig.align = 'center')

# load necessary package
library(tidyverse)
library(broom)
library(rsample)
library(tibble)
library(rcfss)
library(knitr)
library(caret)
library(elasticnet)
library(gridExtra)
library(e1071)
library(glmnet)
library(ROCR)
library(ICEbox)
library(klaR)

# Set the theme as minimal
theme_set(theme_minimal())

# Import the data
gss_test <- read_csv("data/gss_test.csv")
gss_train <- read_csv("data/gss_train.csv")
```

Conceptual: Cost functions for classification trees

Question 1

- In simple classification settings with two classes, I have plotted a graph below about the function of the Gini index, classification error, and cross-entropy. The x-axis is p (the proportion of training observations in the region that are from the first class), and the y-axis is the error rate.
- Of these three possible cost functions, cross-entropy would be the best to use when growing a decision tree because the it is the most sensitive to the node purity since it will take a value near zero only when p is all near zero or near one as the graph shows. On the other hand, classification error would be best to use when pruning a decision tree because in most cases, the prediction accuracy of the final pruned tree is the goal.

```r
p <- seq(0, 1, 0.0001)
gini_index <- p * (1 - p) * 2
class_error <- 1 - pmax(p, 1 - p)
entropy <- -(p * log(p) + (1 - p) * log(1 - p))
df_q1 <- data.frame(p, gini_index, class_error, entropy)
df_q1 %>%
  gather(key = "index", value = "error", -p) %>%
  ggplot(aes(x = p, y = error, color = index)) +
  geom_point()
```



Application: Predicting attitudes towards racist college professors

Estimate the models

Question 2

Logitstic regression

```r
set.seed(0) # Ensure the reproducibility

features <- setdiff(names(gss_train), "colrac")
train_x <- gss_train[, features]
train_y <- as.factor(gss_train$colrac)
test_x <- gss_test[, features]
test_y <- as.factor(gss_test$colrac)

# Logistic regression
(logit <- train(x = train_x, y = train_y, method = 'glm',
                trControl = trainControl(method = 'cv', number = 10)))
```

```
## Generalized Linear Model
```

```
##
## 1476 samples
##   55 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1329, 1329, 1329, 1328, 1328, 1328, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7920344  0.5830822
```

```
# Predict colarc using training set
train_pred_logit <- predict(logit)
```

Naive Bayes

```
set.seed(0) # Ensure the reproducibility

# Naive Bayes
(nb <- train(x = train_x, y = train_y, method = 'nb', metric = 'Accuracy',
             trControl = trainControl(method = 'cv', number = 10),
             tuneGrid = expand.grid(usekernel = c(TRUE, FALSE),
                                    fL = 0:5,
                                    adjust = seq(0, 5, by = 1))))
```

```
## Naive Bayes
##
## 1476 samples
##   55 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1329, 1329, 1329, 1328, 1328, 1328, ...
## Resampling results across tuning parameters:
##
##   usekernel  fL  adjust  Accuracy   Kappa
##   FALSE      0   0       0.7392168  0.4782127
##   FALSE      0   1       0.7392168  0.4782127
##   FALSE      0   2       0.7392168  0.4782127
##   FALSE      0   3       0.7392168  0.4782127
##   FALSE      0   4       0.7392168  0.4782127
##   FALSE      0   5       0.7392168  0.4782127
##   FALSE      1   0       0.7392168  0.4782127
##   FALSE      1   1       0.7392168  0.4782127
##   FALSE      1   2       0.7392168  0.4782127
##   FALSE      1   3       0.7392168  0.4782127
##   FALSE      1   4       0.7392168  0.4782127
##   FALSE      1   5       0.7392168  0.4782127
```

```
##   FALSE      2   0       0.7392168  0.4782127
##   FALSE      2   1       0.7392168  0.4782127
##   FALSE      2   2       0.7392168  0.4782127
##   FALSE      2   3       0.7392168  0.4782127
##   FALSE      2   4       0.7392168  0.4782127
##   FALSE      2   5       0.7392168  0.4782127
##   FALSE      3   0       0.7392168  0.4782127
##   FALSE      3   1       0.7392168  0.4782127
##   FALSE      3   2       0.7392168  0.4782127
##   FALSE      3   3       0.7392168  0.4782127
##   FALSE      3   4       0.7392168  0.4782127
##   FALSE      3   5       0.7392168  0.4782127
##   FALSE      4   0       0.7392168  0.4782127
##   FALSE      4   1       0.7392168  0.4782127
##   FALSE      4   2       0.7392168  0.4782127
##   FALSE      4   3       0.7392168  0.4782127
##   FALSE      4   4       0.7392168  0.4782127
##   FALSE      4   5       0.7392168  0.4782127
##   FALSE      5   0       0.7392168  0.4782127
##   FALSE      5   1       0.7392168  0.4782127
##   FALSE      5   2       0.7392168  0.4782127
##   FALSE      5   3       0.7392168  0.4782127
##   FALSE      5   4       0.7392168  0.4782127
##   FALSE      5   5       0.7392168  0.4782127
##    TRUE      0   0              NaN        NaN
##    TRUE      0   1       0.7256251  0.4518571
##    TRUE      0   2       0.7249816  0.4513483
##    TRUE      0   3       0.7222743  0.4463254
##    TRUE      0   4       0.7263100  0.4549191
##    TRUE      0   5       0.7249448  0.4526199
##    TRUE      1   0              NaN        NaN
##    TRUE      1   1       0.7256251  0.4518571
##    TRUE      1   2       0.7249816  0.4513483
##    TRUE      1   3       0.7222743  0.4463254
##    TRUE      1   4       0.7263100  0.4549191
##    TRUE      1   5       0.7249448  0.4526199
##    TRUE      2   0              NaN        NaN
##    TRUE      2   1       0.7256251  0.4518571
##    TRUE      2   2       0.7249816  0.4513483
##    TRUE      2   3       0.7222743  0.4463254
##    TRUE      2   4       0.7263100  0.4549191
##    TRUE      2   5       0.7249448  0.4526199
##    TRUE      3   0              NaN        NaN
##    TRUE      3   1       0.7256251  0.4518571
##    TRUE      3   2       0.7249816  0.4513483
##    TRUE      3   3       0.7222743  0.4463254
##    TRUE      3   4       0.7263100  0.4549191
##    TRUE      3   5       0.7249448  0.4526199
##    TRUE      4   0              NaN        NaN
##    TRUE      4   1       0.7256251  0.4518571
##    TRUE      4   2       0.7249816  0.4513483
##    TRUE      4   3       0.7222743  0.4463254
##    TRUE      4   4       0.7263100  0.4549191
##    TRUE      4   5       0.7249448  0.4526199
```

```
##    TRUE      5   0             NaN        NaN
##    TRUE      5   1       0.7256251  0.4518571
##    TRUE      5   2       0.7249816  0.4513483
##    TRUE      5   3       0.7222743  0.4463254
##    TRUE      5   4       0.7263100  0.4549191
##    TRUE      5   5       0.7249448  0.4526199
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were fL = 0, usekernel = FALSE and adjust
##  = 0.
```

```
# Predict colarc using training set
train_pred_nb <- predict(nb)
```

Elastic net regression

```
set.seed(0) # Ensure the reproducibility

# Elastic net regression
(en <- train(x = train_x, y = train_y, method = 'glmnet', metric = 'Accuracy',
             trControl = trainControl(method = 'cv', number = 10),
             tuneLength = 10))
```

```
## glmnet
##
## 1476 samples
##   55 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1329, 1329, 1329, 1328, 1328, 1328, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda        Accuracy   Kappa
##   0.1    0.0001218549  0.7933903  0.5857409
##   0.1    0.0002815006  0.7933903  0.5857409
##   0.1    0.0006503028  0.7933903  0.5857409
##   0.1    0.0015022839  0.7927101  0.5842762
##   0.1    0.0034704708  0.7927147  0.5842925
##   0.1    0.0080172380  0.7974628  0.5937819
##   0.1    0.0185208600  0.8008457  0.6005393
##   0.1    0.0427855899  0.7994898  0.5978469
##   0.1    0.0988402644  0.7960930  0.5909894
##   0.1    0.2283338359  0.7832506  0.5655286
##   0.2    0.0001218549  0.7927147  0.5843506
##   0.2    0.0002815006  0.7927147  0.5843506
##   0.2    0.0006503028  0.7933949  0.5857417
##   0.2    0.0015022839  0.7933949  0.5856954
##   0.2    0.0034704708  0.7960976  0.5911699
##   0.2    0.0080172380  0.8008503  0.6006702
```

```
##   0.2    0.0185208600  0.8008503  0.6005453
##   0.2    0.0427855899  0.7994990  0.5976592
##   0.2    0.0988402644  0.7913863  0.5814489
##   0.2    0.2283338359  0.7758090  0.5505008
##   0.3    0.0001218549  0.7920344  0.5830109
##   0.3    0.0002815006  0.7920344  0.5830109
##   0.3    0.0006503028  0.7920344  0.5830109
##   0.3    0.0015022839  0.7940706  0.5871359
##   0.3    0.0034704708  0.7981338  0.5953501
##   0.3    0.0080172380  0.7981384  0.5952501
##   0.3    0.0185208600  0.7988233  0.5965457
##   0.3    0.0427855899  0.8001839  0.5989068
##   0.3    0.0988402644  0.7812236  0.5610733
##   0.3    0.2283338359  0.7764846  0.5514829
##   0.4    0.0001218549  0.7920344  0.5830109
##   0.4    0.0002815006  0.7920344  0.5830109
##   0.4    0.0006503028  0.7933857  0.5857321
##   0.4    0.0015022839  0.7947417  0.5884624
##   0.4    0.0034704708  0.7981338  0.5953501
##   0.4    0.0080172380  0.7974628  0.5937763
##   0.4    0.0185208600  0.8022155  0.6032925
##   0.4    0.0427855899  0.7981568  0.5946701
##   0.4    0.0988402644  0.7751333  0.5489807
##   0.4    0.2283338359  0.7805433  0.5586771
##   0.5    0.0001218549  0.7927101  0.5843478
##   0.5    0.0002815006  0.7927101  0.5843478
##   0.5    0.0006503028  0.7933857  0.5857321
##   0.5    0.0015022839  0.7947417  0.5884624
##   0.5    0.0034704708  0.7981293  0.5951665
##   0.5    0.0080172380  0.7988187  0.5965075
##   0.5    0.0185208600  0.7981476  0.5949972
##   0.5    0.0427855899  0.7981568  0.5946688
##   0.5    0.0988402644  0.7758136  0.5495973
##   0.5    0.2283338359  0.7798630  0.5572804
##   0.6    0.0001218549  0.7927101  0.5843478
##   0.6    0.0002815006  0.7927101  0.5843478
##   0.6    0.0006503028  0.7940660  0.5870752
##   0.6    0.0015022839  0.7940660  0.5871275
##   0.6    0.0034704708  0.7974536  0.5937374
##   0.6    0.0080172380  0.7981430  0.5951917
##   0.6    0.0185208600  0.7974812  0.5934578
##   0.6    0.0427855899  0.7934225  0.5850766
##   0.6    0.0988402644  0.7805479  0.5588006
##   0.6    0.2283338359  0.7839309  0.5629163
##   0.7    0.0001218549  0.7927101  0.5844111
##   0.7    0.0002815006  0.7927101  0.5844111
##   0.7    0.0006503028  0.7947417  0.5884580
##   0.7    0.0015022839  0.7947417  0.5884462
##   0.7    0.0034704708  0.7988141  0.5964618
##   0.7    0.0080172380  0.8028820  0.6046770
##   0.7    0.0185208600  0.8035714  0.6055084
##   0.7    0.0427855899  0.7886790  0.5756913
##   0.7    0.0988402644  0.7825749  0.5629593
##   0.7    0.2283338359  0.7852822  0.5656587
```

```
##   0.8    0.0001218549  0.7927101  0.5844111
##   0.8    0.0002815006  0.7927101  0.5844111
##   0.8    0.0006503028  0.7940660  0.5871414
##   0.8    0.0015022839  0.7960930  0.5912134
##   0.8    0.0034704708  0.7988141  0.5964188
##   0.8    0.0080172380  0.8022063  0.6033386
##   0.8    0.0185208600  0.8049228  0.6082329
##   0.8    0.0427855899  0.7812282  0.5605837
##   0.8    0.0988402644  0.7818946  0.5615626
##   0.8    0.2283338359  0.7805111  0.5543474
##   0.9    0.0001218549  0.7927101  0.5844111
##   0.9    0.0002815006  0.7927101  0.5844111
##   0.9    0.0006503028  0.7940660  0.5871414
##   0.9    0.0015022839  0.7974490  0.5939395
##   0.9    0.0034704708  0.8001747  0.5990587
##   0.9    0.0080172380  0.8028866  0.6046630
##   0.9    0.0185208600  0.8035760  0.6055527
##   0.9    0.0427855899  0.7825795  0.5629930
##   0.9    0.0988402644  0.7819038  0.5616298
##   0.9    0.2283338359  0.7778038  0.5486873
##   1.0    0.0001218549  0.7927101  0.5844111
##   1.0    0.0002815006  0.7927101  0.5844111
##   1.0    0.0006503028  0.7940660  0.5871414
##   1.0    0.0015022839  0.7981247  0.5952758
##   1.0    0.0034704708  0.7988141  0.5963705
##   1.0    0.0080172380  0.8022201  0.6032450
##   1.0    0.0185208600  0.8035760  0.6054566
##   1.0    0.0427855899  0.7832598  0.5644229
##   1.0    0.0988402644  0.7825703  0.5614066
##   1.0    0.2283338359  0.7602133  0.5099432
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.8 and lambda = 0.01852086.
```

```
# Predict colarc using training set
train_pred_en <- predict(en)
```

Decision tree (CART)

```
set.seed(0) # Ensure the reproducibility

# Decision tree (CART)
(cart <- train(x = train_x, y = train_y, method = 'rpart', metric = 'Accuracy',
               trControl = trainControl(method = 'cv', number = 10),
               tuneLength = 10))
```

```
## CART
##
## 1476 samples
##   55 predictor
##    2 classes: '0', '1'
```

```
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1329, 1329, 1329, 1328, 1328, 1328, ...
## Resampling results across tuning parameters:
##
##   cp           Accuracy   Kappa
##   0.003328578  0.7893501  0.5761988
##   0.003566334  0.7893501  0.5761988
##   0.004279601  0.7920757  0.5820361
##   0.005706134  0.7968239  0.5914447
##   0.006419401  0.7893547  0.5765298
##   0.007132668  0.7907106  0.5791672
##   0.008559201  0.7853006  0.5684257
##   0.012838802  0.7710655  0.5384775
##   0.023537803  0.7683582  0.5300474
##   0.499286733  0.6022385  0.1702890
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.005706134.
```

```
# Predict colarc using training set
train_pred_cart <- predict(cart)
```

Bagging (Bagged CART)

```
set.seed(0) # Ensure the reproducibility

# Bagging
(bagging <- train(x = train_x, y = train_y,  method = 'treebag', metric = 'Accuracy',
                  trControl = trainControl(method = 'cv', number = 10)))
```

```
## Bagged CART
##
## 1476 samples
##   55 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1329, 1329, 1329, 1328, 1328, 1328, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7893501  0.5761145
```

```
                  #No tuning parameters of this model (treebag)

# Predict colarc using training set
train_pred_bagging <- predict(bagging)
```

Random forest

```
set.seed(1) # Ensure the reproducibility

# Random forest
(rf <- train(x = train_x, y = train_y,  method = 'rf', metric = 'Accuracy',
             trControl = trainControl(method = 'cv', number = 10),
             tuneGrid = expand.grid(mtry = seq(20, 30, by = 2))))
```

```
## Random Forest
##
## 1476 samples
##   55 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1329, 1328, 1329, 1327, 1328, 1328, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   20    0.8008482  0.5984145
##   22    0.8028707  0.6024959
##   24    0.8001772  0.5970570
##   26    0.8008299  0.5985031
##   28    0.8042313  0.6054340
##   30    0.7981318  0.5931330
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 28.
```

```
# Predict colarc using training set
train_pred_rf <- predict(rf)
```

Boosting

```
set.seed(0) # Ensure the reproducibility

# Boosting
(boosting <- train(x = train_x, y = train_y,  method = 'gbm', metric = 'Accuracy',
                   trControl = trainControl(method = 'cv', number = 10, verboseIter = FALSE),
                   verbose = FALSE,
                   tuneLength = 5))
```

```
## Stochastic Gradient Boosting
##
## 1476 samples
##   55 predictor
##    2 classes: '0', '1'
##
```

```
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1329, 1329, 1329, 1328, 1328, 1328, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7866382  0.5684367
##   1                  100      0.7941028  0.5846924
##   1                  150      0.8022247  0.6012773
##   1                  200      0.8042517  0.6056019
##   1                  250      0.8028866  0.6033055
##   2                   50      0.7961252  0.5881993
##   2                  100      0.8076209  0.6123829
##   2                  150      0.8049136  0.6071047
##   2                  200      0.8014984  0.6004914
##   2                  250      0.8001333  0.5982316
##   3                   50      0.8008687  0.5989572
##   3                  100      0.8008366  0.5993184
##   3                  150      0.8028590  0.6031891
##   3                  200      0.8021695  0.6021779
##   3                  250      0.7981109  0.5944652
##   4                   50      0.7961160  0.5889700
##   4                  100      0.7927238  0.5831022
##   4                  150      0.7933811  0.5845125
##   4                  200      0.7954174  0.5882328
##   4                  250      0.7960930  0.5896907
##   5                   50      0.8008687  0.5989614
##   5                  100      0.8076025  0.6127791
##   5                  150      0.8035209  0.6045981
##   5                  200      0.8048906  0.6075947
##   5                  250      0.8048676  0.6075708
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 100, interaction.depth =
##  2, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
# Predict colarc using training set
train_pred_boosting <- predict(boosting)
```

Evaluate the models

Question 3

Cross-validated error rate

```
set.seed(0) # Ensure the reproducibility

# Calculate the CV error rate
```

```r
model <- c("Logit", "Naive bayes", "Elastic net", "CART", "Bagging", "Random forest", "Boosting")
kable(data.frame(model = model,
           error.rate = c(1 - logit$results$Accuracy,
                          1 - max(nb$results$Accuracy, na.rm = TRUE),
                          1 - max(cart$results$Accuracy),
                          1 - max(bagging$results$Accuracy),
                          1 - max(en$results$Accuracy),
                          1 - max(rf$results$Accuracy),
                          1 - max(boosting$results$Accuracy))), digits = 3)
```

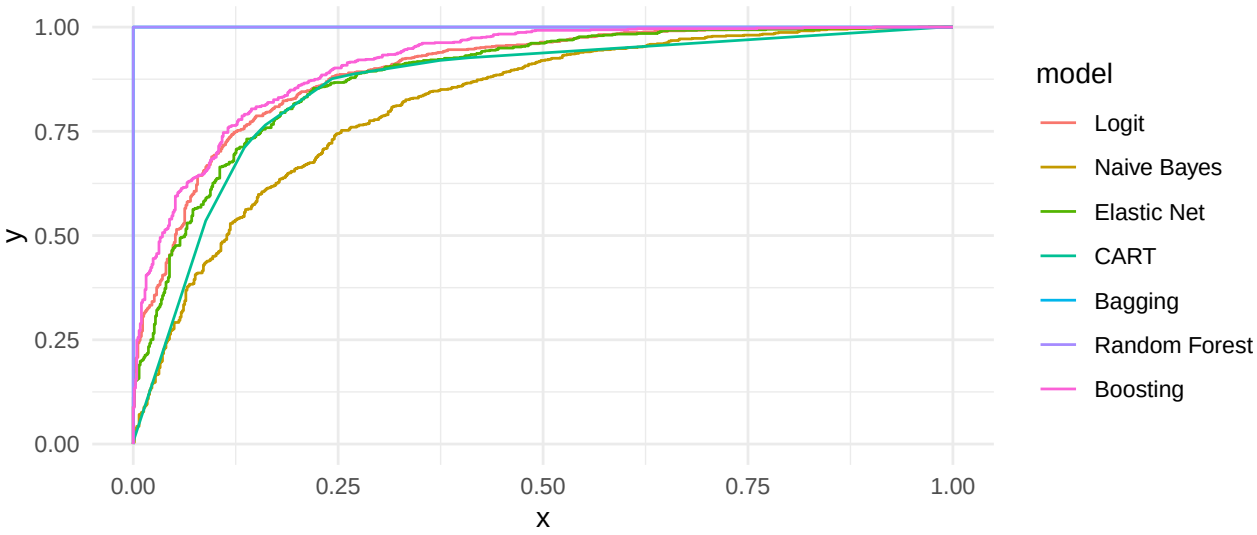
model	error.rate
Logit	0.208
Naive bayes	0.261
Elastic net	0.203
CART	0.211
Bagging	0.195
Random forest	0.196
Boosting	0.192

In terms of Cross-validated error rate, boosting is the best performance model because it has the minimum error rate.

ROC/AUC

```r
set.seed(0) # Ensure the reproducibility

# Calculate ROC/AUC by each model using training data
train_pred_logit_p <- predict(logit, type = "prob")
train_logit_roc <- prediction(train_pred_logit_p["1"], as.factor(train_y))
train_logit_auc <- as.numeric(performance(train_logit_roc, 'auc')@y.values)
train_logit_perf <- performance(train_logit_roc, 'tpr', 'fpr')
train_logit_data <- data.frame(x = slot(train_logit_perf, 'x.values'),
                               y = slot(train_logit_perf, 'y.values'),
                               model = 'Logit')
colnames(train_logit_data) <- c("x", "y", "model")

train_pred_nb_p <- predict(nb, type = "prob")
train_nb_roc <- prediction(train_pred_nb_p["1"], as.factor(train_y))
train_nb_auc <- as.numeric(performance(train_nb_roc, 'auc')@y.values)
train_nb_perf <- performance(train_nb_roc, 'tpr', 'fpr')
train_nb_data <- data.frame(x = slot(train_nb_perf, 'x.values'),
                            y = slot(train_nb_perf, 'y.values'),
                            model = 'Naive Bayes')
colnames(train_nb_data) <- c("x", "y", "model")

train_pred_en_p <- predict(en, type = "prob")
train_en_roc <- prediction(train_pred_en_p["1"], as.factor(train_y))
train_en_auc <- as.numeric(performance(train_en_roc, 'auc')@y.values)
train_en_perf <- performance(train_en_roc, 'tpr', 'fpr')
```

```r
train_en_data <- data.frame(x = slot(train_en_perf, 'x.values'),
                            y = slot(train_en_perf, 'y.values'),
                            model = 'Elastic Net')
colnames(train_en_data) <- c("x", "y", "model")

train_pred_cart_p <- predict(cart, type = "prob")
train_cart_roc <- prediction(train_pred_cart_p["1"], as.factor(train_y))
train_cart_auc <- as.numeric(performance(train_cart_roc, 'auc')@y.values)
train_cart_perf <- performance(train_cart_roc, 'tpr', 'fpr')
train_cart_data <- data.frame(x = slot(train_cart_perf, 'x.values'),
                              y = slot(train_cart_perf, 'y.values'),
                              model = 'CART')
colnames(train_cart_data) <- c("x", "y", "model")

train_pred_bagging_p <- predict(bagging, type = "prob")
train_bagging_roc <- prediction(train_pred_bagging_p["1"], as.factor(train_y))
train_bagging_auc <- as.numeric(performance(train_bagging_roc, 'auc')@y.values)
train_bagging_perf <- performance(train_bagging_roc, 'tpr', 'fpr')
train_bagging_data <- data.frame(x = slot(train_bagging_perf, 'x.values'),
                                 y = slot(train_bagging_perf, 'y.values'),
                                 model = 'Bagging')
colnames(train_bagging_data) <- c("x", "y", "model")

train_pred_rf_p <- predict(rf, type = "prob")
train_rf_roc <- prediction(train_pred_rf_p["1"], as.factor(train_y))
train_rf_auc <- as.numeric(performance(train_rf_roc, 'auc')@y.values)
train_rf_perf <- performance(train_rf_roc, 'tpr', 'fpr')
train_rf_data <- data.frame(x = slot(train_rf_perf, 'x.values'),
                            y = slot(train_rf_perf, 'y.values'),
                            model = 'Random Forest')
colnames(train_rf_data) <- c("x", "y", "model")

train_pred_boosting_p <- predict(boosting, type = "prob")
train_boosting_roc <- prediction(train_pred_boosting_p["1"], as.factor(train_y))
train_boosting_auc <- as.numeric(performance(train_boosting_roc, 'auc')@y.values)
train_boosting_perf <- performance(train_boosting_roc, 'tpr', 'fpr')
train_boosting_data <- data.frame(x = slot(train_boosting_perf, 'x.values'),
                                  y = slot(train_boosting_perf, 'y.values'),
                                  model = 'Boosting')
colnames(train_boosting_data) <- c("x", "y", "model")

# Show the ROC curve
rbind(train_logit_data, train_nb_data, train_en_data, train_cart_data,
      train_bagging_data, train_rf_data,train_boosting_data) %>%
  ggplot(aes(x = x, y = y, color = model)) +
  geom_line()
```




```
# Calculate the AUC
kable(data.frame(model = model,
           auc = c(train_logit_auc, train_nb_auc, train_en_auc, train_cart_auc,
                  train_bagging_auc, train_rf_auc, train_boosting_auc)), digits = 2)
```

model	auc
Logit	0.90
Naive bayes	0.82
Elastic net	0.88
CART	0.86
Bagging	1.00
Random forest	1.00
Boosting	0.91

In terms of ROC/AUC, bagging and random forest are the best performance models because they have the largest AUC.

Question 4

In this problem set, our goal is predicting the attitudes towards racist college professors. In Question 3, we evaluated each model's training performance by using training data. Since there is a possibility of overfitting the model, it is hard to say the "best" model by this analysis unless we use the test data to check the error rate or AUC. Especially, since bagging and random forest's AUC is close to 1, their models surely face the issue of the overfitting. In terms of reducing the error rate for the classification, I think the boosting is the best model because of the minimum cross-valiated error rate.

Evaluate the best model

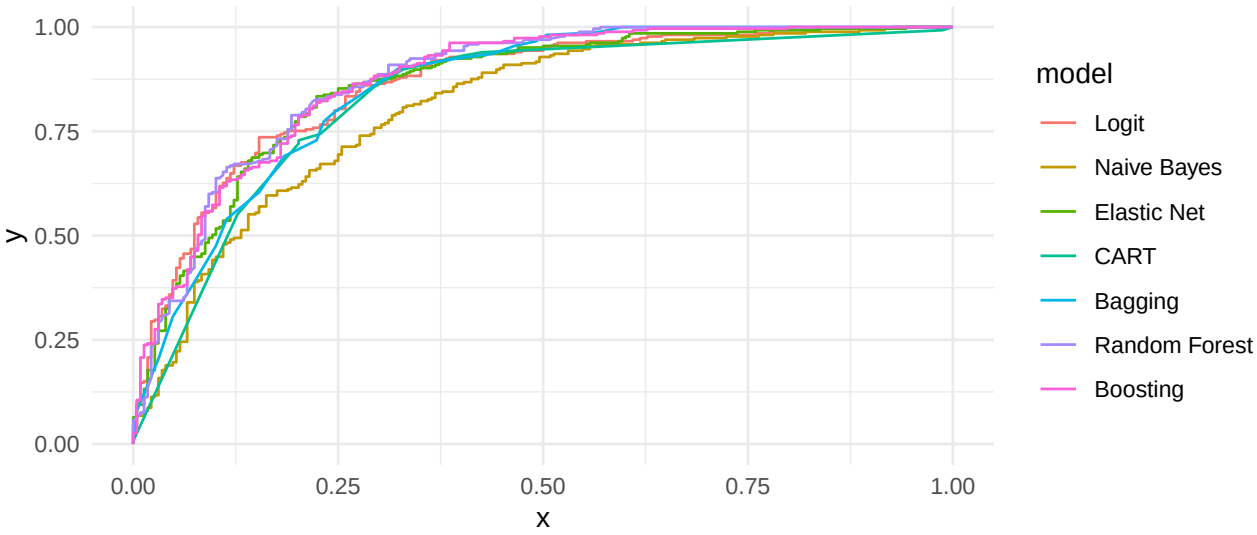
Question 5

```r
set.seed(0) # Ensure the reproducibility

# # Calculate ROC/AUC by each model using test data
test_pred_logit <- predict(logit, newdata = test_x)
test_pred_nb <- predict(nb, newdata = test_x)
test_pred_en <- predict(en, newdata = test_x)
test_pred_cart <- predict(cart, newdata = test_x)
test_pred_bagging <- predict(bagging, newdata = test_x)
test_pred_rf <- predict(rf, newdata = test_x)
test_pred_boosting <- predict(boosting, newdata = test_x)

test_pred_logit_p <- predict(logit, newdata = test_x, type = "prob")
test_logit_roc <- prediction(test_pred_logit_p["1"], as.factor(test_y))
test_logit_auc <- as.numeric(performance(test_logit_roc, 'auc')@y.values)
test_logit_perf <- performance(test_logit_roc, 'tpr', 'fpr')
test_logit_data <- data.frame(x = slot(test_logit_perf, 'x.values'),
                              y = slot(test_logit_perf, 'y.values'),
                              model = 'Logit')
colnames(test_logit_data) <- c("x", "y", "model")


test_pred_nb_p <- predict(nb, newdata = test_x, type = "prob")
test_nb_roc <- prediction(test_pred_nb_p["1"], as.factor(test_y))
test_nb_auc <- as.numeric(performance(test_nb_roc, 'auc')@y.values)
test_nb_perf <- performance(test_nb_roc, 'tpr', 'fpr')
test_nb_data <- data.frame(x = slot(test_nb_perf, 'x.values'),
                           y = slot(test_nb_perf, 'y.values'),
                           model = 'Naive Bayes')
colnames(test_nb_data) <- c("x", "y", "model")

test_pred_en_p <- predict(en, newdata = test_x, type = "prob")
test_en_roc <- prediction(test_pred_en_p["1"], as.factor(test_y))
test_en_auc <- as.numeric(performance(test_en_roc, 'auc')@y.values)
test_en_perf <- performance(test_en_roc, 'tpr', 'fpr')
test_en_data <- data.frame(x = slot(test_en_perf, 'x.values'),
                           y = slot(test_en_perf, 'y.values'),
                           model = 'Elastic Net')
colnames(test_en_data) <- c("x", "y", "model")

test_pred_cart_p <- predict(cart, newdata = test_x, type = "prob")
test_cart_roc <- prediction(test_pred_cart_p["1"], as.factor(test_y))
test_cart_auc <- as.numeric(performance(test_cart_roc, 'auc')@y.values)
test_cart_perf <- performance(test_cart_roc, 'tpr', 'fpr')
test_cart_data <- data.frame(x = slot(test_cart_perf, 'x.values'),
                             y = slot(test_cart_perf, 'y.values'),
                             model = 'CART')
colnames(test_cart_data) <- c("x", "y", "model")

test_pred_bagging_p <- predict(bagging, newdata = test_x, type = "prob")
test_bagging_roc <- prediction(test_pred_bagging_p["1"], as.factor(test_y))
test_bagging_auc <- as.numeric(performance(test_bagging_roc, 'auc')@y.values)
test_bagging_perf <- performance(test_bagging_roc, 'tpr', 'fpr')
test_bagging_data <- data.frame(x = slot(test_bagging_perf, 'x.values'),
```

```
                                  y = slot(test_bagging_perf, 'y.values'),
                                  model = 'Bagging')
colnames(test_bagging_data) <- c("x", "y", "model")

test_pred_rf_p <- predict(rf, newdata = test_x, type = "prob")
test_rf_roc <- prediction(test_pred_rf_p["1"], as.factor(test_y))
test_rf_auc <- as.numeric(performance(test_rf_roc, 'auc')@y.values)
test_rf_perf <- performance(test_rf_roc, 'tpr', 'fpr')
test_rf_data <- data.frame(x = slot(test_rf_perf, 'x.values'),
                           y = slot(test_rf_perf, 'y.values'),
                           model = 'Random Forest')
colnames(test_rf_data) <- c("x", "y", "model")

test_pred_boosting_p <- predict(boosting, newdata = test_x, type = "prob")
test_boosting_roc <- prediction(test_pred_boosting_p["1"], as.factor(test_y))
test_boosting_auc <- as.numeric(performance(test_boosting_roc, 'auc')@y.values)
test_boosting_perf <- performance(test_boosting_roc, 'tpr', 'fpr')
test_boosting_data <- data.frame(x = slot(test_boosting_perf, 'x.values'),
                                 y = slot(test_boosting_perf, 'y.values'),
                                 model = 'Boosting')
colnames(test_boosting_data) <- c("x", "y", "model")

# Calcurate the error rate
accuracy_rate <- c(confusionMatrix(test_pred_logit, as.factor(test_y))[3]$overall['Accuracy'],
                   confusionMatrix(test_pred_nb, as.factor(test_y))[3]$overall['Accuracy'],
                   confusionMatrix(test_pred_en, as.factor(test_y))[3]$overall['Accuracy'],
                   confusionMatrix(test_pred_cart, as.factor(test_y))[3]$overall['Accuracy'],
                   confusionMatrix(test_pred_bagging, as.factor(test_y))[3]$overall['Accuracy'],
                   confusionMatrix(test_pred_rf, as.factor(test_y))[3]$overall['Accuracy'],
                   confusionMatrix(test_pred_boosting, as.factor(test_y))[3]$overall['Accuracy'])

kable(data.frame(model = model,
                 error.rate = 1 - accuracy_rate), digits = 3)
```

model	error.rate
Logit	0.209
Naive bayes	0.272
Elastic net	0.199
CART	0.213
Bagging	0.209
Random forest	0.199
Boosting	0.207

```
# Show the ROC curve
rbind(test_logit_data, test_nb_data, test_en_data, test_cart_data,
      test_bagging_data, test_rf_data,test_boosting_data) %>%
  ggplot(aes(x = x, y = y, color = model)) +
  geom_line()
```




```
# Calculate the AUC
kable(data.frame(model = model,
           auc = c(test_logit_auc, test_nb_auc, test_en_auc, test_cart_auc,
                  test_bagging_auc, test_rf_auc, test_boosting_auc)), digits = 3)
```

model	auc
Logit	0.862
Naive bayes	0.806
Elastic net	0.859
CART	0.830
Bagging	0.852
Random forest	0.873
Boosting	0.872

Just in case, I evaluated all the models using test data, not only the boosting model (i.e, the "best" model in question 4). I think that the boosting model can generate well because of the relatively small error rate (20.7%) and large AUC (87.2%). However, I think the "best" model using test set is not the boosting but the random forest because of the minimum error rate and the maximum AUC. Thus, we cannot surely say that the boosting is the "best". As for the boosting, there might be a possibility of overgitting for the training data. Note: It is difficult for us to surely conclude which model is the best. If we use finer hyperparameter tuning, we might have a different result (the boosting could be the "best" model even using test data).

Bonus: PDPs/ICE

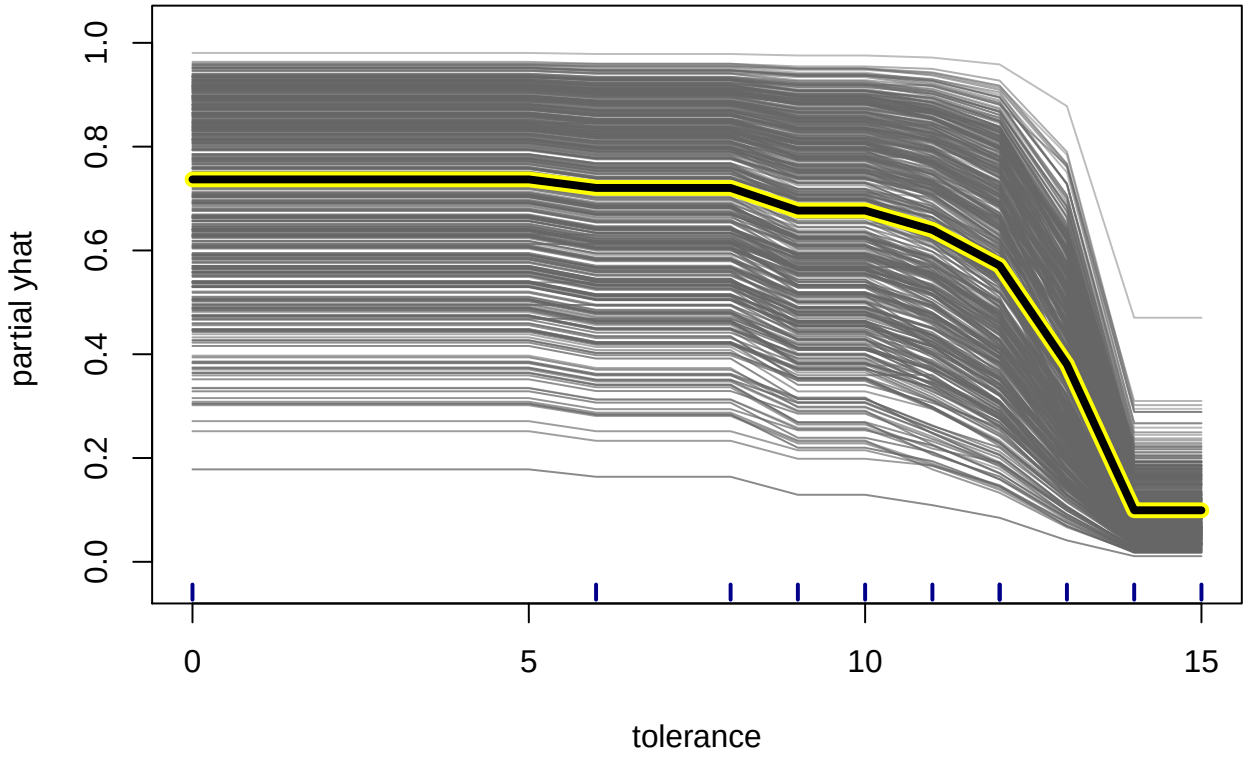
Question 6

```
set.seed(0) # Ensure the reproducibility

# Plot PDP/ICE curves over the range of "tolerance"
ice.tolerance <- ice(object = boosting,
                     X = as.data.frame(test_x),
                     predictor = "tolerance",
```

```
                      predictfcn = function(object, newdata){
                        predict(object, newdata, type = "prob")[, 2]})
```

```
## .................
## y not passed, so range_y is range of ice curves and sd_y is sd of predictions on real observations
```

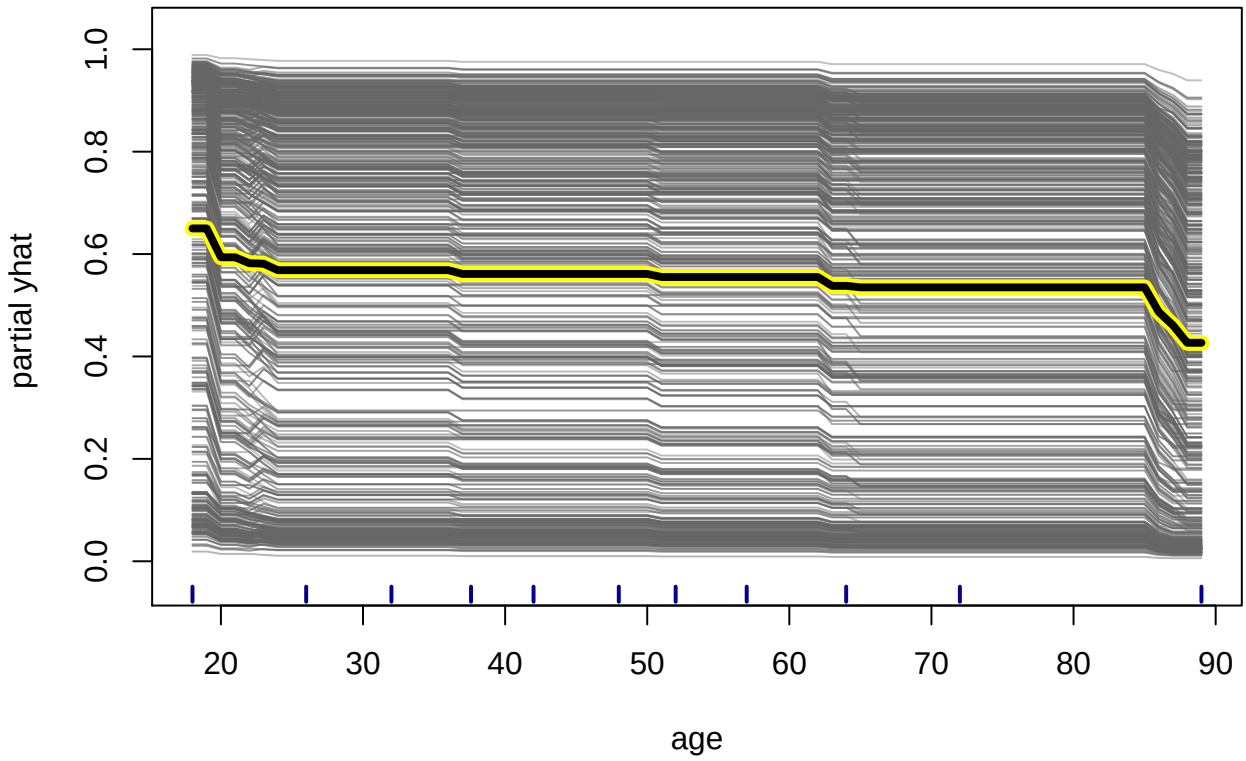
```
plot(ice.tolerance, x_quantile = FALSE, pts_preds_size = 0)
```




```
# Plot PDP/ICE curves over the range of "age"
ice.age <- ice(object = boosting,
                      X = as.data.frame(test_x),
                      predictor = "age",
                      predictfcn = function(object, newdata){
                        predict(object, newdata, type = "prob")[, 2]})
```

```
## ..................................................................
## y not passed, so range_y is range of ice curves and sd_y is sd of predictions on real observations
```

```
plot(ice.age, x_quantile = FALSE, pts_preds_size = 0)
```



I use the boosting model (the "best" model selected in question 4). The following plots are PDP/ICE curves over the range of "tolerance" and "age". PDP is a yellow line and ICE is multiple gray lines.

- "tolerance": The overall trend is that people with a higher tolerance rate are expected to allow the racist professor. More accurately, when the tolerance rate is from 0 to 12, the probability of allowing the racist professor is from 60% to 70%. However, when the tolerance rate is from 12 to 14, the probability is rapidly going down. Finally, when the tolerance rate is high from 14 to 15, the probability is about 10%. Since the variance of the ICE curves is shrinking, the tolerance rate can be said as a good predictor especially in case of the tolerance rate from 14 to 15.
- "age": The overall trend is that age and the probability are not correlated so much, while older people have a little bit larger allowance for the racist professor. More accurately, as for the people of age 20 to 85, the probability of allowing the racist professor is about 60%. People under 20 have a little less probability (65%) while people over 85 have a little more probability (50%). Since the variance of ICE curves is so large, age CANNOT be said as a good predictor.