yc4384_Yangyang_Chen_HW4

Yangyang Chen (UNI: yc4384)

2024-04-10

Contents

Load packages

```
library(tidyverse)
library(caret)
library(rpart.plot)
library(ranger)
library(gbm)
library(knitr)
library(party)
library(ISLR)
library(pROC)
```

Problem 1: How Much is Your Out-of-State Tuition?

Load and split data into training and testing sets

```
set.seed(2024)

# import and tidy
data = read_csv("College.csv") |>
  janitor::clean_names() |>
  select(-college)

# partition data into training and testing sets as randomized 4:1 splits
train_index = createDataPartition(y = data$outstate, p = 0.8, list = F)
train_data = data[train_index, ]
test_data = data[-train_index, ]

# testing set response for RMSE calculation
test_resp = test_data$outstate
```

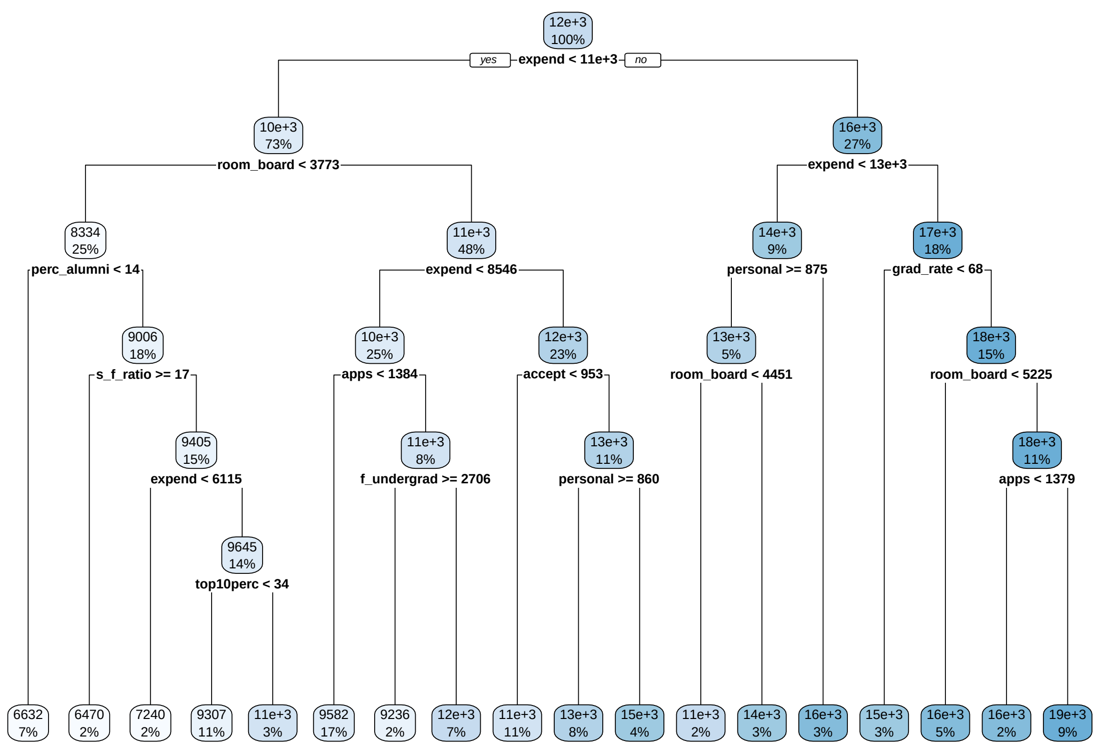
Set cross validation methods

```
# for regression tree
ctrl_re = trainControl(method = "repeatedcv", number = 2, repeats = 5)

# for classification tree under the minimal MSE rule
ctrl = trainControl(method = "repeatedcv", number = 2, repeats = 5,
                       summaryFunction = twoClassSummary,
                       classProbs = TRUE)

# for classification tree under the 1SE rule
ctrl_1se = trainControl(method = "repeatedcv", number = 2, repeats = 5,
                        summaryFunction = twoClassSummary,
                        classProbs = TRUE,
                        selectionFunction = "oneSE")
```

a) **Fit and plot a regression tree model**

Use the regression tree (CART) approach to graph an optimally pruned tree. At the top (root) of the tree, it is shown that splitting at `expend` over or under 11K provides significantly more accurate predictions for out-of-state tuitions than any other.

```r
set.seed(2024)

rpart_grid = data.frame(cp = exp(seq(-8,-5, length = 100)))
rpart_fit = train(outstate ~ . ,
                  data,
                  subset = train_index,
                  method = "rpart",
                  tuneGrid = rpart_grid,
                  trControl = ctrl_re)
# ggplot(rpart_fit, highlight = TRUE)

rpart.plot(rpart_fit$finalModel)
```



For comparison, the following is the code using the conditional inference tree (CIT) approach. The code generates an overly cluttered graph but `expend` is still atop the decision tree.

```r
set.seed(2024)

ctree_grid = data.frame(mincriterion = 1-exp(seq(-2, 0, length = 100)))
ctree_fit = train(outstate ~ . ,
                  data,
                  subset = train_index,
                  method = "ctree",
                  tuneGrid = ctree_grid,
```

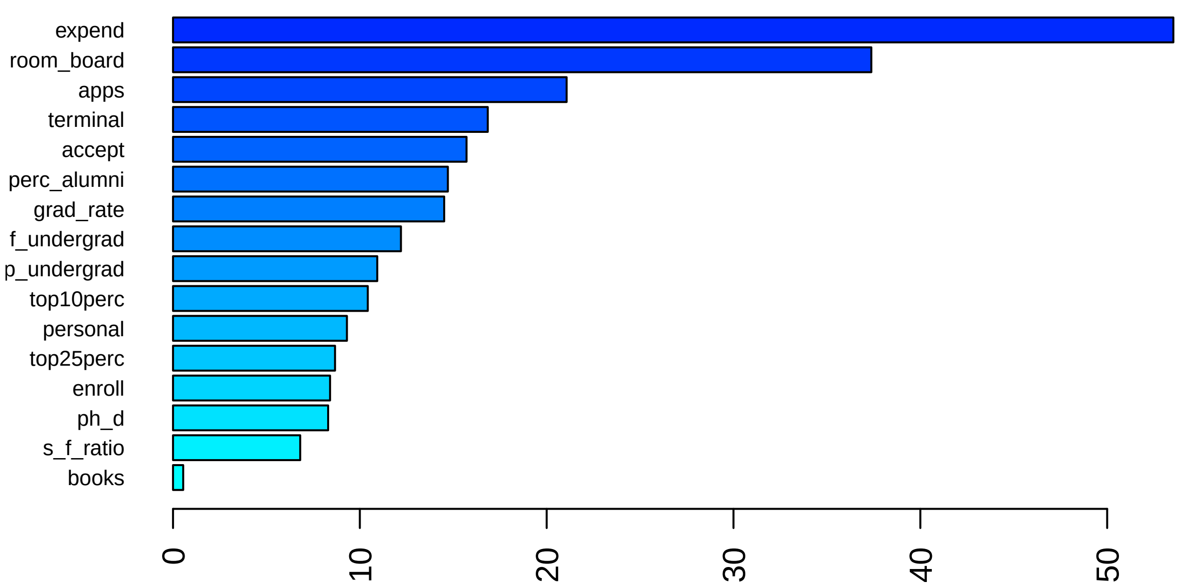
```r
                    trControl = ctrl_re)
ggplot(ctree_fit, highlight = TRUE)

plot(ctree_fit$finalModel)

RMSE(predict(ctree_fit, newdata = test_data), test_resp)
```

b) **Fit and evaluate a random forest regression model**

```
set.seed(2024)

rf_grid = expand.grid(mtry = 1:16,
                      splitrule = "variance",
                      min.node.size = 1:6)
rf_fit = train(outstate ~ . ,
               data,
               subset = train_index,
               method = "ranger",
               tuneGrid = rf_grid,
               trControl = ctrl_re)
# ggplot(rf_fit, highlight = TRUE)
```

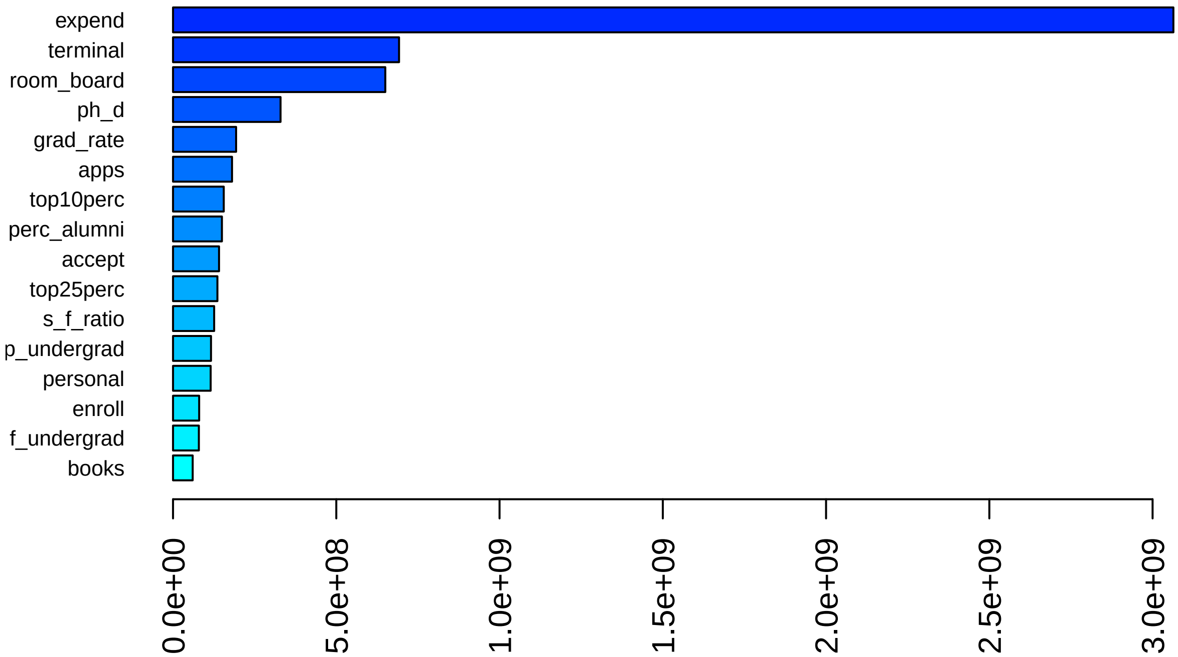
Calculate and graph variable importance using permutation and impurity metrics. Similarly, both evaluations suggest `expend` as the most important predictor for regressing out-of-state tuition, followed by `room-board`.

```
set.seed(2024)

rf_perm = ranger(outstate ~ . ,
                 train_data,
                 mtry = rf_fit$bestTune[[1]],
                 splitrule = "variance",
                 min.node.size = rf_fit$bestTune[[3]],
                 importance = "permutation",
                 scale.permutation.importance = TRUE)
barplot(sort(importance(rf_perm), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("cyan","blue"))(19))
```




```
rf_imp = ranger(outstate ~ . ,
                train_data,
                mtry = rf_fit$bestTune[[1]],
```

```
                splitrule = "variance",
                min.node.size = rf_fit$bestTune[[3]],
                importance = "impurity")
barplot(sort(importance(rf_imp), decreasing = FALSE),
        las = 2, horiz = TRUE, cex.names = 0.7,
        col = colorRampPalette(colors = c("cyan","blue"))(19))
```



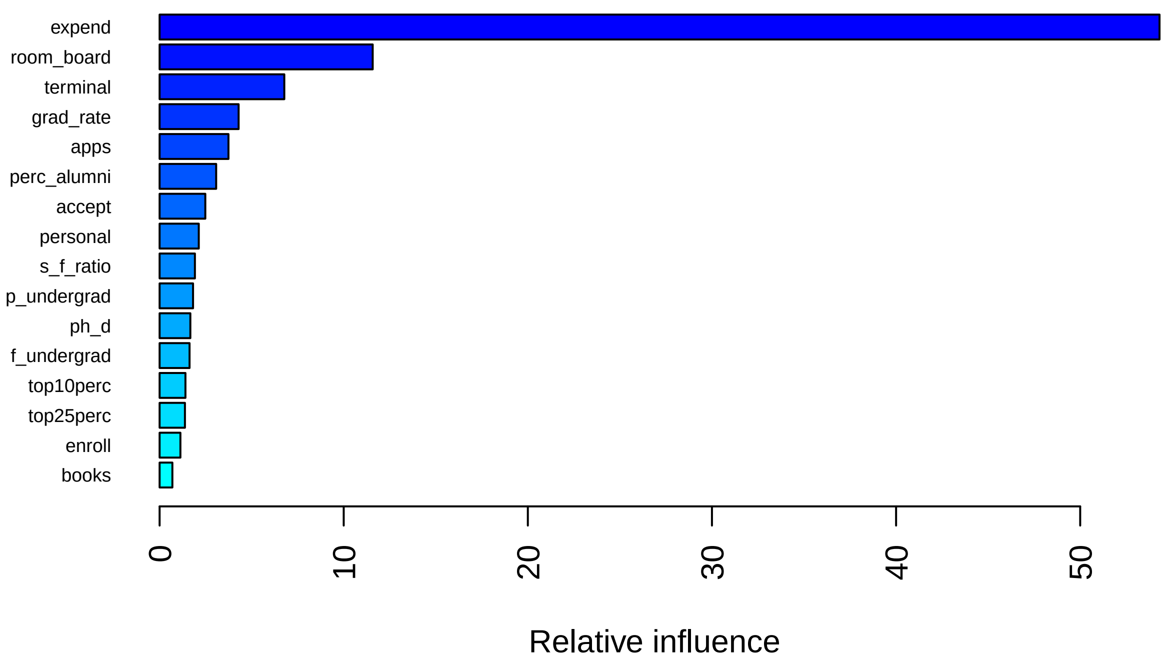
For the random forest model test error and its interpretation, see the end of part C).

c) **Fit and evaluate a gradient boosting regression model**

```r
set.seed(2024)

gbm_grid = expand.grid(n.trees = c(2000, 3000, 4000, 5000),
                       interaction.depth = 1:5,
                       shrinkage = c(0.001,0.003,0.005),
                       n.minobsinnode = c(1, 10))
gbm_fit = train(outstate ~ . ,
                train_data,
                method = "gbm",
                tuneGrid = gbm_grid,
                trControl = ctrl_re,
                verbose = FALSE)
# ggplot(gbm_fit, highlight = TRUE)
```

Calculate, list and graph variable importance. Again, boosting suggests `expend` and `room-board` as the 2 most important predictors for regressing out-of-state tuition.

```r
summary(gbm_fit$finalModel, las = 2, cBars = 19, cex.names = 0.6)
```




```
##                     var    rel.inf
## expend           expend 54.3082745
## room_board   room_board 11.5734622
## terminal       terminal  6.7729533
## grad_rate     grad_rate  4.2909105
## apps               apps  3.7389891
## perc_alumni perc_alumni  3.0720762
## accept           accept  2.4825233
## personal       personal  2.1280995
## s_f_ratio     s_f_ratio  1.9171942
## p_undergrad p_undergrad  1.8155129
```

```
## ph_d                 ph_d  1.6699978
## f_undergrad f_undergrad  1.6269904
## top10perc     top10perc  1.4040538
## top25perc     top25perc  1.3765600
## enroll           enroll  1.1292374
## books             books  0.6931648
```

Show test errors for both the random forest and boosting models, and compare them with their cross-validation errors. The boosting model has a lower test error and cross-validation error than those of the random forest model. Notice both their test RMSEs fall in the 4th quartile of their cross-validation errors, which is rather high but still within expectation, and both models could be applied to other new testing sets.

```r
rf_test_rmse = RMSE(predict(rf_fit, newdata = test_data), test_resp)
boost_test_rmse = RMSE(predict(gbm_fit, newdata = test_data), test_resp)
kable(c(rf = rf_test_rmse, boost = boost_test_rmse), col.names = "RMSE", "simple")
```

	RMSE
rf	1678.162
boost	1619.765

```r
summary(resamples(list(rf = rf_fit, boost = gbm_fit)))
```

```
##
## Call:
## summary.resamples(object = resamples(list(rf = rf_fit, boost = gbm_fit)))
##
## Models: rf, boost
## Number of resamples: 10
##
## MAE
##           Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## rf    1325.305 1365.448 1379.930 1386.057 1407.894 1458.292    0
## boost 1279.403 1323.424 1330.867 1337.483 1340.203 1449.421    0
##
## RMSE
##           Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## rf    1754.762 1786.446 1830.181 1841.339 1891.290 1967.757    0
## boost 1706.987 1740.645 1764.806 1786.805 1816.898 1969.182    0
##
## Rsquared
##            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## rf    0.7492317 0.7527947 0.7573357 0.7620531 0.7676637 0.7875193    0
## boost 0.7450418 0.7670033 0.7774699 0.7742701 0.7859785 0.7915404    0
```

Problem 2: Classification models using the auto.csv dataset

This problem uses the auto data in the in Homework 3. We have 392 observations with 8 parameters: 7 predictors, including 4 continuous variables (`displacement`, `horsepower`, `weight`, `acceleration`) and 3 categorical variables (`cylinders`, `year`, `origin`), along with one binary outcome variable, `mpg_cat`, which takes values "high" and "low". Half our observations have the "high" label while the other half have the "low" label.

```
# Load data, clean column names, eliminate rows containing NA entries
auto_df = read_csv("auto.csv") |>
  janitor::clean_names() |>
  na.omit() |>
  distinct() |>
  mutate(
    cylinders = as.factor(cylinders),
    year = as.factor(year),
    origin = case_when(origin == "1" ~ "American",
                       origin == "2" ~ "European",
                       origin == "3" ~ "Japanese"),
    origin = as.factor(origin),
    mpg_cat = as.factor(mpg_cat),
    mpg_cat = fct_relevel(mpg_cat, "low")
  ) |>
  as.data.frame()
```

```
## Rows: 392 Columns: 8
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): mpg_cat
## dbl (7): cylinders, displacement, horsepower, weight, acceleration, year, or...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

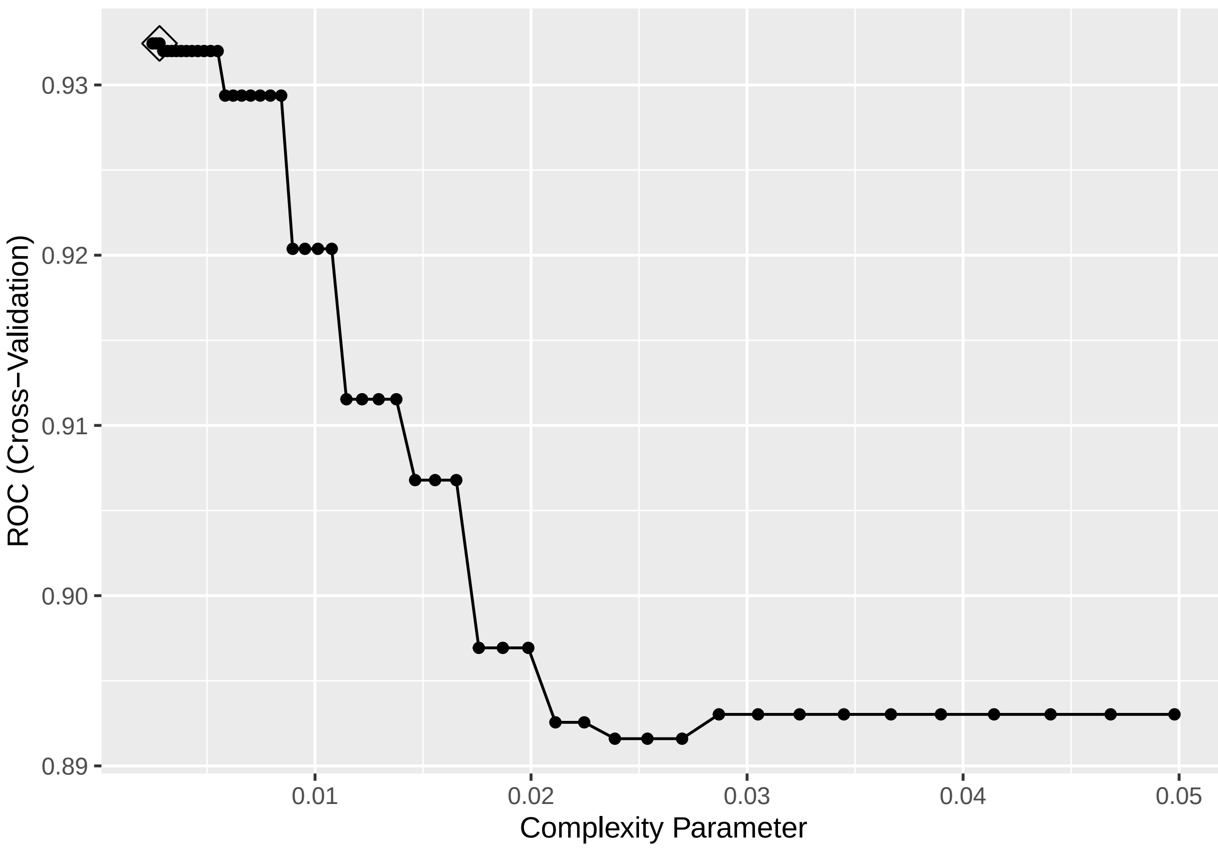
Create a training set containing a random sample of 700 observations, and a test set containing the remaining observations.

```
# Partition data into training/test sets (70% split)
indexTrain = createDataPartition(y = auto_df$mpg_cat, p = 0.7,
list = FALSE)

training_df = auto_df[indexTrain, ]
testing_df = auto_df[-indexTrain,]
```

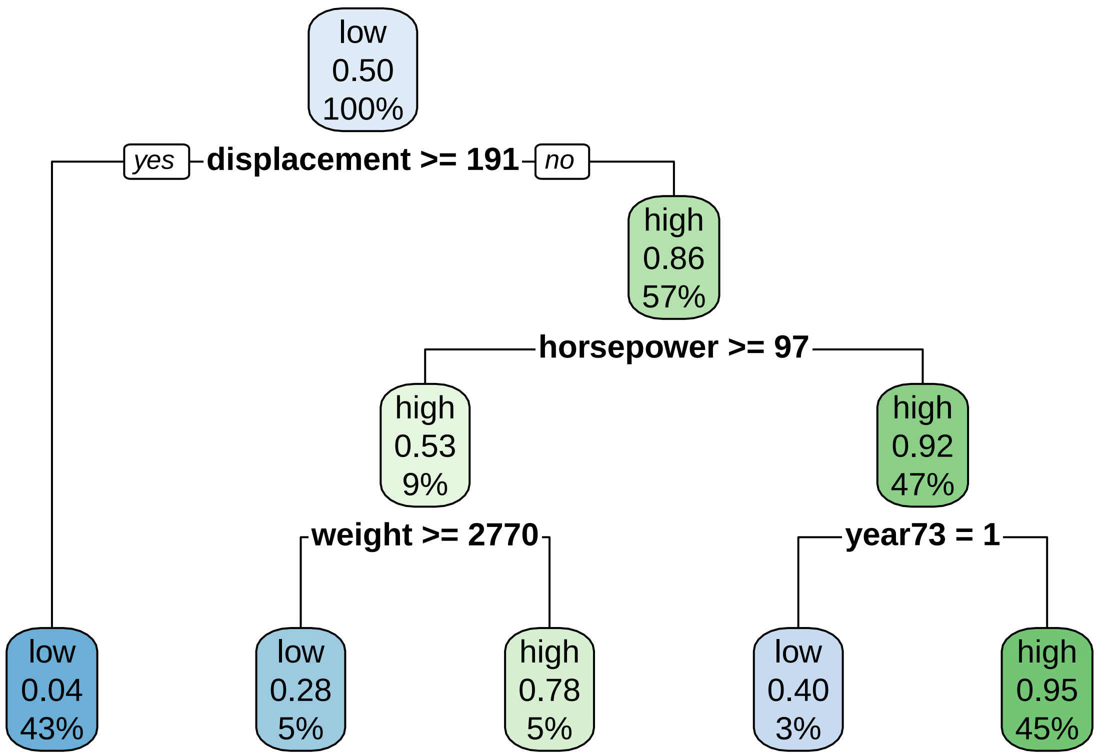
(a) Build a classification tree using the training data, with mpg cat as the response and the other variables as predictors.

```
# create a cross-validation object
ctrl = trainControl(method = "cv",
                    summaryFunction = twoClassSummary,
```

```r
                     classProbs = TRUE)

set.seed(2024)

# build a classification tree using the training data
rpart.fit = train(mpg_cat ~ . ,
                  data = auto_df, # training data
                  method = "rpart",
                  tuneGrid = data.frame(cp = exp(seq(-6,-3, len = 50))), # candidate values for the cp
                  trControl = ctrl,
                  metric = "ROC")

# create a plot of the complexity parameter selection
ggplot(rpart.fit, highlight = TRUE) # highlight the optimal cp value
```




```r
# plot the tree with the lowest cross-validation error
rpart.plot(rpart.fit$finalModel)
```



Which tree size corresponds to the lowest cross-validation error? Is this the same as the tree size obtained using the 1 SE rule?

The code below prints a table of the complexity parameter (cp) values correspnding to the lowest cross-validation error and the 1 standard error rule.

Lowest cross-validation error

```
rpart.fit$bestTune$cp # reports only the best cp value
```

```
## [1] 0.002801638
```

The tree with cp = 0.0028 corresponds to the lowest cross-validation error.

1 SE rule

The tree size obtained using the 1 SE rule is the one with the smalled value of cp that is within one standard error of the minimum cross-validation error (ROC).

```
# find the tree size obtained using the 1 SE rule
cp.table = data.frame(rpart.fit$results)
cp.min = which.min(cp.table$ROC) # finds the index of the row that corresponds to the min ROC
cp.1se = cp.table$cp[which.min(abs(cp.table$ROC[1:cp.min] - (cp.table$ROC[cp.min] + cp.table$ROCSD[cp.mi
cp.1se
```

```
## [1] 0.002978527
```

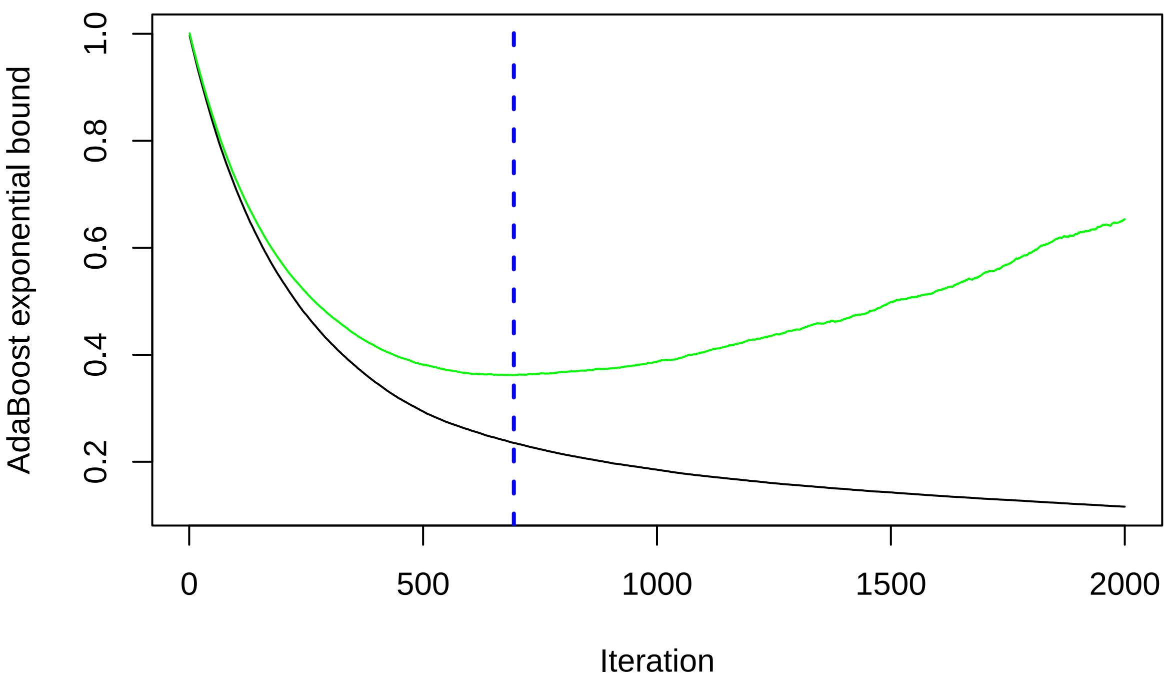
The value of cp that is within one standard error of the minimum cross-validation error is 0.0029. The tree with cp = 0.0029 is obtained using the 1 SE rule.

Therefore, the tree size obtained using the 1 SE rule is not the same as the tree size that corresponds to the lowest cross-validation error. The tree size with the lowest cross-validation error has cp = 0.0028, while the tree size obtained using the 1 SE rule has cp = 0.0029.

(b) Perform boosting on the training data and report the variable importance. Report the test data performance.

```
training_df$mpg_cat <- as.numeric(training_df$mpg_cat == "high")
testing_df$mpg_cat <- as.numeric(testing_df$mpg_cat == "high")
```

```
set.seed(2024)

bst.fit <- gbm(mpg_cat ~ .,
           training_df,
           distribution = "adaboost",
           n.trees = 2000,
           interaction.depth = 2,
           shrinkage = 0.005,
           cv.folds = 10,
           n.cores = 2)

gbm.perf(bst.fit, method = "cv")
```

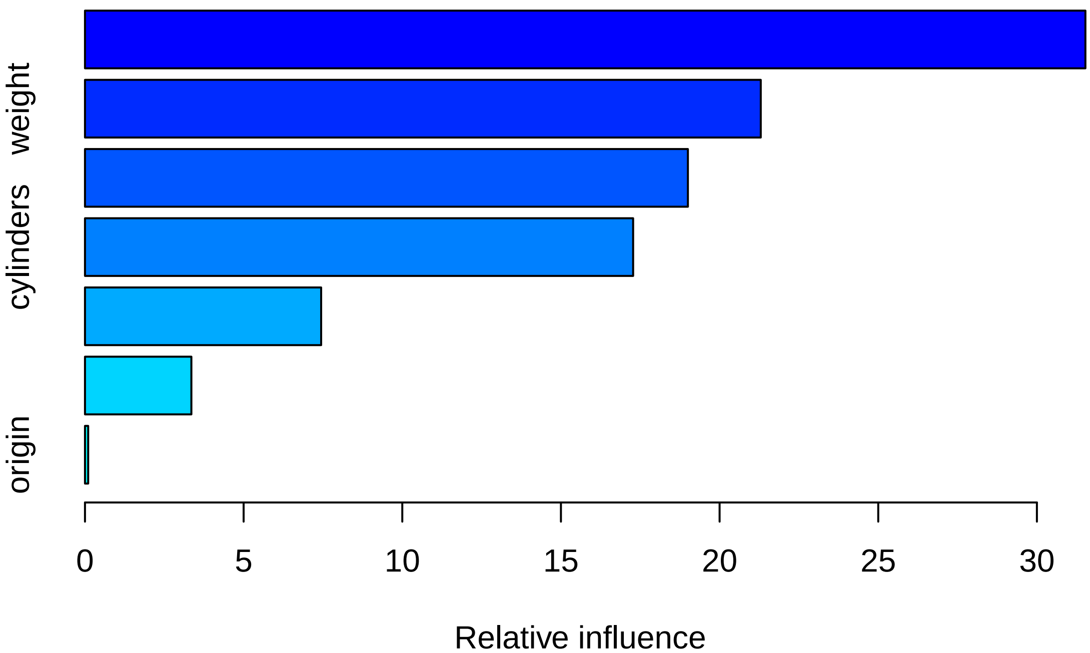



```
## [1] 694
```

Variable importance

The code below plots and prints the relative influence variable importance values for each of the predictors in the in the boosting model (bst.fit). Higher values for relative influence indicate more important variables in predicting the outcome class, `mpg_cat`, which takes values “high” and “low”.

```r
# report variable importance
var_imp <- summary(bst.fit)
```




```r
kable(var_imp)
```

	var	rel.inf
displacement	displacement	31.5287936
weight	weight	21.2971622
year	year	19.0015808
cylinders	cylinders	17.2751803
horsepower	horsepower	7.4426580
acceleration	acceleration	3.3536237
origin	origin	0.1010014

In the boosting model, the **displacement**, **weight**, and **year** variables were the most important predictors in in predicting the outcome class.

Test error

The code below used the trained boosted model (bst.fit) to make predictions on the test dataset. The predict() function is used to generate predictions for the outcome variable `mpg_cat` based on the predictor variables in the test dataset. Then, the test error (RMSE) is calculated.

```r
set.seed(2024)

# predict on test data
pred.bst <- predict(bst.fit, newdata = testing_df, n.trees = 5000) # test data
```

```
## Warning in predict.gbm(bst.fit, newdata = testing_df, n.trees = 5000): Number
## of trees not specified or exceeded number fit so far. Using 2000.
```

```r
# calculate the test error (RMSE)
RMSE <- sqrt(mean((testing_df$mpg_cat - pred.bst)^2))
RMSE
```

```
## [1] 2.933166
```

The test error of the model is 2.933166.