Predict the Price of Used Corolla - Bagging and Random Forest

Langtao Chen

Initial: Feb 20, 2017 Update: March 30, 2020

Contents

In this example, we'll use bagging and random forest (RF) to predict the price of used Corolla.

1.Data

Import the data from the csv file.

```
# Clean the environment
rm(list = ls())

# Read data file
df <- read.csv("ToyotaCorolla.csv")
```

```
# Show the head of the dataset
head(df)
```

```
##   Price Age    KM FuelType HP MetColor Automatic   CC Doors Weight
## 1 13500  23 46986   Diesel 90        1         0 2000     3   1165
## 2 13750  23 72937   Diesel 90        1         0 2000     3   1165
## 3 13950  24 41711   Diesel 90        1         0 2000     3   1165
## 4 14950  26 48000   Diesel 90        0         0 2000     3   1165
## 5 13750  30 38500   Diesel 90        0         0 2000     3   1170
## 6 12950  32 61000   Diesel 90        0         0 2000     3   1170
```

```
# Show the structure of the dataset
str(df)
```

```
## 'data.frame':    1436 obs. of  10 variables:
##  $ Price    : int  13500 13750 13950 14950 13750 12950 16900 18600 21500 12950 ...
##  $ Age      : int  23 23 24 26 30 32 27 30 27 23 ...
##  $ KM       : int  46986 72937 41711 48000 38500 61000 94612 75889 19700 71138 ...
##  $ FuelType : chr  "Diesel" "Diesel" "Diesel" "Diesel" ...
##  $ HP       : int  90 90 90 90 90 90 90 90 192 69 ...
##  $ MetColor : int  1 1 1 0 0 0 1 1 0 0 ...
##  $ Automatic: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ CC       : int  2000 2000 2000 2000 2000 2000 2000 2000 1800 1900 ...
##  $ Doors    : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ Weight   : int  1165 1165 1165 1165 1170 1170 1245 1245 1185 1105 ...
```

```
# Summary statistics
summary(df)
```

```
##      Price            Age              KM            FuelType
##  Min.   : 4350   Min.   : 1.00   Min.   :     1   Length:1436
##  1st Qu.: 8450   1st Qu.:44.00   1st Qu.: 43000   Class :character
##  Median : 9900   Median :61.00   Median : 63390   Mode  :character
##  Mean   :10731   Mean   :55.95   Mean   : 68533
##  3rd Qu.:11950   3rd Qu.:70.00   3rd Qu.: 87021
##  Max.   :32500   Max.   :80.00   Max.   :243000
##        HP           MetColor        Automatic             CC
##  Min.   : 69.0   Min.   :0.0000   Min.   :0.00000   Min.   :1300
```

```
##  1st Qu.: 90.0   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:1400
##  Median :110.0   Median :1.0000   Median :0.00000   Median :1600
##  Mean   :101.5   Mean   :0.6748   Mean   :0.05571   Mean   :1567
##  3rd Qu.:110.0   3rd Qu.:1.0000   3rd Qu.:0.00000   3rd Qu.:1600
##  Max.   :192.0   Max.   :1.0000   Max.   :1.00000   Max.   :2000
##      Doors           Weight
##  Min.   :2.000   Min.   :1000
##  1st Qu.:3.000   1st Qu.:1040
##  Median :4.000   Median :1070
##  Mean   :4.033   Mean   :1072
##  3rd Qu.:5.000   3rd Qu.:1085
##  Max.   :5.000   Max.   :1615
```

From the summary statistics, we found that there is no missing value.

2. Data Partitioning

We use a single 80/20% split.

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
set.seed(1234)
trainIndex <- createDataPartition(df$Price, p = .8, list = FALSE)

train_data <- df[ trainIndex,]
test_data  <- df[-trainIndex,]

nrow(train_data)
```

```
## [1] 1150
```

```
nrow(test_data)
```

```
## [1] 286
```

3. Bagging and Random Forest

We can use the randomForest() method in the randomForest package to implement the bagging and random forests. Bagging is special case of random forest with mtry = p.

3.1. Bagging

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.0.4

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
# Fit a bagged tree
bag_corolla <- randomForest(Price~., data = train_data,
                            mtry = 9, importance = TRUE)

bag_corolla
```

```
##
## Call:
##  randomForest(formula = Price ~ ., data = train_data, mtry = 9,      importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 9
##
##           Mean of squared residuals: 1168995
##                     % Var explained: 91.22
```

mtry=9 means that all 9 predictors are considered for each split of the tree. Thus, this is a bagged regression tree.

Test the performance of the bagged tree on the test dataset.

```
library(caret)

bag_yhat <- predict(bag_corolla, newdata = test_data)

postResample(bag_yhat, test_data$Price)
```

```
##         RMSE     Rsquared          MAE
## 1158.4105028    0.8937825  827.5579611
```

3.2. Random Forest

To fit a random forest, we can set a different value for the mtry parameter. The default value of mtry is p/3 in the randomForest() method. Here let's set mtry = 5.

```r
# Fit a random forest
rf_corolla <- randomForest(Price~., data = train_data,
                           mtry = 5, importance = TRUE)

rf_corolla
```

```
##
## Call:
##  randomForest(formula = Price ~ ., data = train_data, mtry = 5,      importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 1151938
##                     % Var explained: 91.34
```

Test the performance of the random forest on the test dataset.

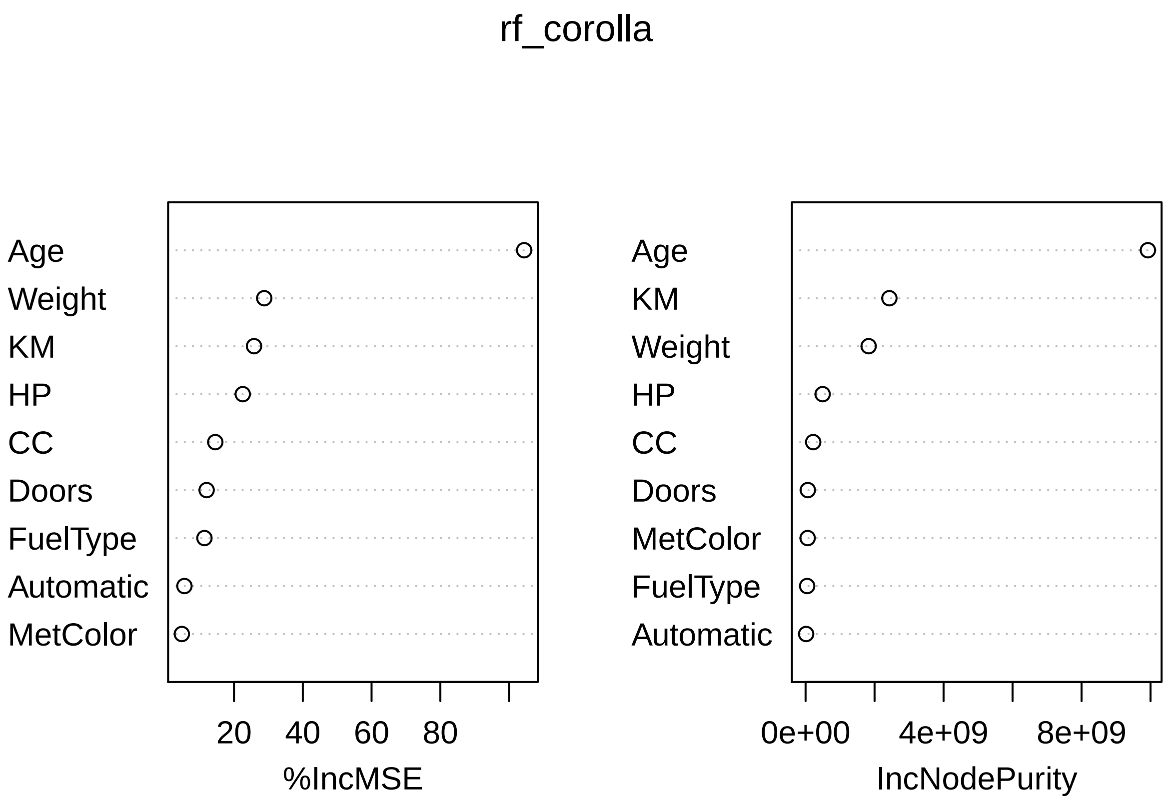
```r
rf_yhat <- predict(rf_corolla, newdata = test_data)

postResample(rf_yhat, test_data$Price)
```

```
##         RMSE     Rsquared          MAE
## 1122.1206570    0.9001429  803.3361902
```

We can see that the random forest has a better prediction performance than the bagged regression tree.

```r
# Plot the importance
varImpPlot(rf_corolla)
```



We can see that age of the used corolla is the most important feature. If age is excluded from the model, the increase in MSE and node purity will be very large. The three most important features are age, weight, and KM.

Let's further tune the parameter mtry by using a repeated 10-fold cross-validation. Here, we use the train() method in the caret package.

```
tuneGrid <- data.frame(mtry =1:9)

tuneGrid
```

```
##   mtry
## 1    1
## 2    2
## 3    3
## 4    4
## 5    5
## 6    6
## 7    7
## 8    8
## 9    9
```

```
library(lubridate)
```

```
## Warning: package 'lubridate' was built under R version 4.0.4
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
control <- trainControl(method = 'repeatedcv',
                        number = 10,
                        repeats = 3)
set.seed(123)

# print out system time before training
start_t <- Sys.time()
cat("",cat("Training started at:",format(start_t, "%a %b %d %X %Y")))
```

```
## Training started at: Tue Apr 06 3:50:54 PM 2021
```

```
rf_tuned <- train(Price ~ ., data = train_data,
                  method = 'rf',
                  trControl = control,
                  tuneGrid = tuneGrid)

# print out system time after training
finish_t <- Sys.time()
cat("",cat("Training finished at:",format(finish_t, "%a %b %d %X %Y")))
```

```
## Training finished at: Tue Apr 06 3:56:01 PM 2021
```

```
cat("The training process finished in",difftime(finish_t,start_t,units="mins"), "minutes")
```

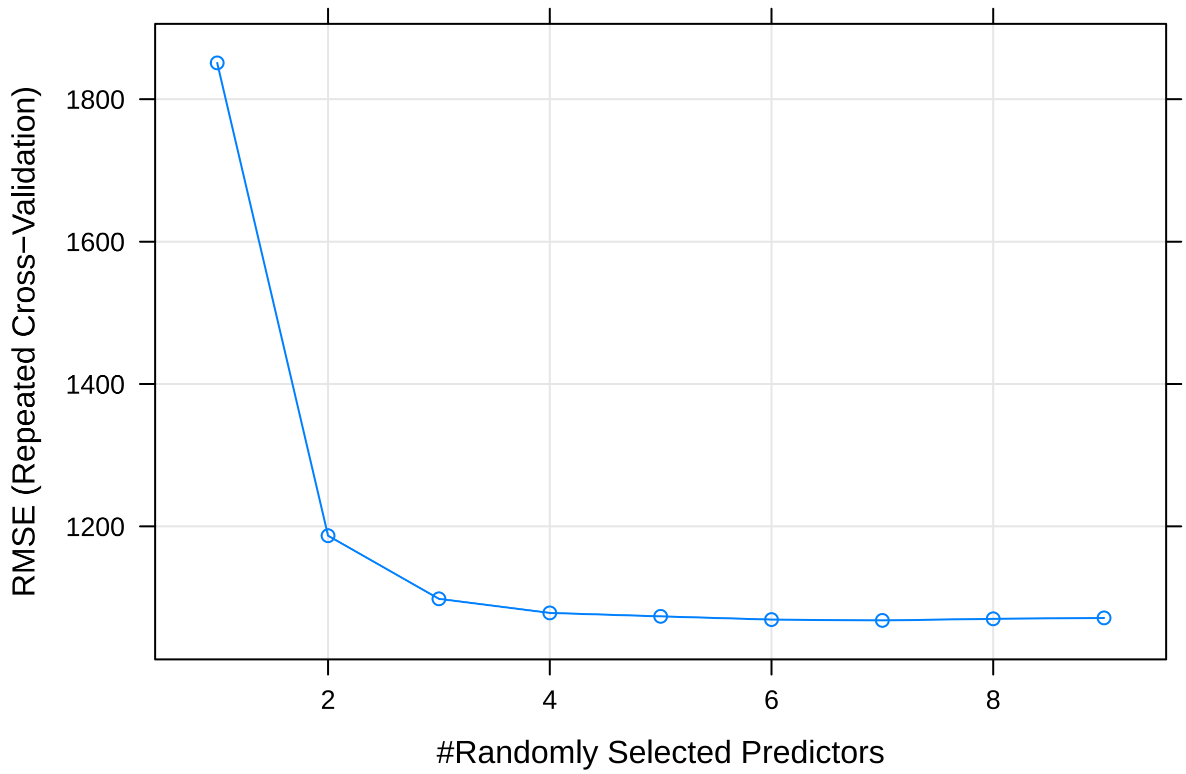
```
## The training process finished in 5.122552 minutes
```

```
print(rf_tuned)
```

```
## Random Forest
##
## 1150 samples
##    9 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 1035, 1035, 1035, 1036, 1035, 1034, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   1     1850.990  0.8480542  1398.2980
##   2     1187.009  0.8997856   892.0086
##   3     1098.338  0.9094517   831.9544
##   4     1078.569  0.9118771   824.1121
##   5     1073.734  0.9124241   824.8960
```

```
##   6      1069.176  0.9130877   824.5403
##   7      1068.019  0.9131663   825.5026
##   8      1070.270  0.9127050   828.1985
##   9      1071.552  0.9125353   828.7931
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 7.
```

```
plot(rf_tuned)
```



The cross-validation shows that mtry = 7 leads to the best model. However, the difference between mtry = 5 and mtry = 7 is not very large. Our initial choice mtry = 5 is not bad!