HW16

111078513

2023-06-02

```r
# loads some pachages we need
library(rpart)
library(rpart.plot)
```

```r
# Load the data and remove missing values
cars <- read.table("auto-data.txt", header=FALSE, na.strings = "?")
names(cars) <- c("mpg", "cylinders", "displacement", "horsepower", "weight",
                 "acceleration", "model_year", "origin", "car_name")
cars$car_name <- NULL
cars <- na.omit(cars)
```

```r
# IMPORTANT: Shuffle the rows of data in advance for this project!
set.seed(27935752)
cars <- cars[sample(1:nrow(cars)),]
```

```r
# DV and IV of formulas we are interested in
cars_full <- mpg ~ cylinders + displacement + horsepower + weight +
  acceleration + model_year + factor(origin)
cars_reduced <- mpg ~ weight + acceleration + model_year + factor(origin)
cars_full_poly2 <- mpg ~ poly(cylinders, 2) + poly(displacement, 2) + poly(horsepower, 2) +
  poly(weight, 2) + poly(acceleration, 2) + model_year +
  factor(origin)
cars_reduced_poly2 <- mpg ~ poly(weight, 2) + poly(acceleration,2) + model_year +
  factor(origin)
cars_reduced_poly6 <- mpg ~ poly(weight, 6) + poly(acceleration,6) + model_year +
  factor(origin)
```

Question 1) Compute and report the in-sample fitting error (MSE_{in}) of all the models described above. It might be easier to first write a function called mse_in(...) that returns the fitting error of a single model.

```r
# Run the models.
attach(cars)
lm_full <- lm(cars_full)
lm_reduced <- lm(cars_reduced)
lm_poly2_full <- lm(cars_full_poly2)
lm_poly2_reduced <- lm(cars_reduced_poly2)
lm_poly6_reduced <- lm(cars_reduced_poly6)
rt_full <- rpart(cars_full)
rt_reduced <- rpart(cars_reduced)
```

```r
# Make the list we'll need
model_list <- list(lm_full, lm_reduced, lm_poly2_full, lm_poly2_reduced,
                   lm_poly6_reduced, rt_full, rt_reduced)
name_list <- c("lm(cars_full)", "lm(lm_reduced)", "lm(cars_full_poly2)",
               "lm(cars_reduced_poly2)", "lm(cars_reduced_poly6)",
               "rpart(cars_full)", "rpart(cars_reduced)")
formula_list <- c(cars_full, cars_reduced, cars_full_poly2,
                  cars_reduced_poly2, cars_reduced_poly6,
                  cars_full, cars_reduced)
function_list <- c(lm, lm, lm, lm, lm, rpart, rpart)
```

```r
# Build the function we'll need.
mse_in <- function(model){
  mean(residuals(model)^2)
}
```

```r
# Get all MSE of models above.
MSE_in_list <- sapply(1:7, \(i){
  model <-  model_list[[i]]
  mse_in <- mse_in(model)
})
names(MSE_in_list) <- name_list
MSE_in_list
```

```
##          lm(cars_full)         lm(lm_reduced)    lm(cars_full_poly2)
##              10.682122              10.971643               7.919030
## lm(cars_reduced_poly2) lm(cars_reduced_poly6)       rpart(cars_full)
##               8.364546               8.254377               9.155146
##    rpart(cars_reduced)
##               9.501344
```

Question 2) Let's try some simple evaluation of prediction error. Let's work with the lm_reduced model and test its predictive performance with split-sample testing:

a. Split the data into 70:30 for training:test.

```r
train_set <- cars[1:(nrow(cars)*0.7), ]
test_set <- cars[-(1:(nrow(cars)*0.7)), ]
```

b. Retrain the lm_reduced model on just the training dataset (call the new model: trained_model); Show the coefficients of the trained model.

```r
train_model <- lm(cars_reduced, data = train_set)
summary(train_model)$coefficients
```

```
##                     Estimate   Std. Error     t value     Pr(>|t|)
## (Intercept)     -25.18094941 5.0090298892  -5.0271110 9.119436e-07
## weight           -0.00551164 0.0003394218 -16.2383237 9.347739e-42
## acceleration      0.06855571 0.0855923713   0.8009559 4.238667e-01
## model_year        0.82885622 0.0631211683  13.1311926 9.537389e-31
## factor(origin)2   3.22722498 0.6527806111   4.9438125 1.352589e-06
```

```
## factor(origin)3   2.03952491 0.6289446776   3.2427731 1.333655e-03
```

c. Use the trained_model model to predict the mpg of the test dataset

(i) What is the in-sample mean-square fitting error (MSE_{in}) of the trained model?

```
mean(residuals(train_model)^2)
```

```
## [1] 11.70576
```

(ii) What is the out-of-sample mean-square prediction error (MSE_{out}) of the test dataset?

```
pred <- predict(train_model, test_set)
mean((test_set$mpg - pred)^2)
```

```
## [1] 10.2117
```

d. Show a data frame of the test set's actual mpg values, the predicted mpg values, and the difference of the two (ε_{out} = predictive error); Just show us the first several rows of this dataframe.

```
mpg_actual_pred <- data.frame("pred" = pred, "actual" = test_set$mpg)
head(mpg_actual_pred, 5)
```

```
##         pred actual
## 89  14.04156   14.0
## 215 17.51041   13.0
## 288 19.40509   16.5
## 179 25.07802   23.0
## 398 29.12358   31.0
```

Question 3) Let's use k-fold cross validation (k-fold CV) to see how all these models perform predictively!

a. Write a function that performs k-fold cross-validation (see class notes and ask us online for hints!). Name your function k_fold_mse(model, dataset, k=10, ...) – it should return the MSE_{out} of the operation. Your function must accept a model, dataset and number of folds (k) but can also have whatever other parameters you wish.

```
# Calculate prediction error for fold i out of k
# n for nrow(dataset), formula for formula, func for lm() or rpart()
fold_i_pe <- function(i, k, dataset, n, formula, func) {
  folds <- cut(1:n, breaks = k, labels = FALSE)
  test_indices <- which(folds == i)
  test_set <- dataset[test_indices, ]
  train_set <- dataset[-test_indices, ]
  trained_model <- func(formula, data = train_set)
  predictions <- predict(trained_model, test_set)
  test_set$mpg - predictions
}
```

```r
# Calculate mse_out across all folds
k_fold_mse <- function(dataset, k=10, formula, func) {
  fold_pred_errors <- sapply(1:k, \(i) {
  fold_i_pe(i, k, dataset, nrow(dataset), formula, func)})
  pred_errors <- unlist(fold_pred_errors)
  mean(pred_errors^2)
}
```

(i) Use your k_fold_mse function to find and report the 10-fold CV MSE_{out} for all models.

```r
MSE_out_list <- sapply(1:7, \(i){k_fold_mse(cars, k = 10,
                                  formula = formula_list[[i]],
                                  func = function_list[[i]])})
names(MSE_out_list) <- name_list
MSE_out_list
```

```
##          lm(cars_full)         lm(lm_reduced)    lm(cars_full_poly2)
##              11.262460              11.415855               8.599373
## lm(cars_reduced_poly2) lm(cars_reduced_poly6)       rpart(cars_full)
##               8.818607               9.267369              13.342221
##    rpart(cars_reduced)
##              13.476272
```

(ii) For all the models, which is bigger — the fit error (MSE_{in}) or the prediction error (MSE_{out})?

```r
MSE_df <- data.frame(method = name_list, MSE_in = MSE_in_list,
                     MSE_out = MSE_out_list)
MSE_df[, c("MSE_in", "MSE_out")]
```

```
##                           MSE_in   MSE_out
## lm(cars_full)          10.682122 11.262460
## lm(lm_reduced)         10.971643 11.415855
## lm(cars_full_poly2)     7.919030  8.599373
## lm(cars_reduced_poly2)  8.364546  8.818607
## lm(cars_reduced_poly6)  8.254377  9.267369
## rpart(cars_full)        9.155146 13.342221
## rpart(cars_reduced)     9.501344 13.476272
```

The MSE_{out} values of all models are higher than the MSE_{in} values.

(iii) Does the 10-fold MSE_{out} of a model remain stable (same value) if you re-estimate it over and over again, or does it vary?

```r
tentimes_kfold <- sapply(1:7, \(i){
  mean(replicate(10, k_fold_mse(cars[sample(1:nrow(cars)),],
                                k = 10, formula = formula_list[[i]],
                                func = function_list[[i]])))})
MSE_df$"10 times k-fold MSE_out" = tentimes_kfold
MSE_df[, c(3, 4)]
```

```
##                          MSE_out 10 times k-fold MSE_out
## lm(cars_full)          11.262460               11.321864
## lm(lm_reduced)         11.415855               11.403597
```

```
## lm(cars_full_poly2)     8.599373              8.645329
## lm(cars_reduced_poly2)  8.818607              8.861234
## lm(cars_reduced_poly6)  9.267369              9.285558
## rpart(cars_full)       13.342221             12.816573
## rpart(cars_reduced)    13.476272             12.716515
```

b. Make sure your k_fold_mse() function can accept as many folds as there are rows (i.e., k=392).

(i) How many rows are in the training dataset and test dataset of each iteration of k-fold CV when k=392?

```
n = nrow(cars)
k = 392
folds <- cut(1:n, breaks = k, labels = FALSE)
test_indices <- which(folds == 1)
train_set <- cars[-test_indices, ]
print(paste("There are ", nrow(train_set), " in train_set", sep = ""))
```

```
## [1] "There are 391 in train_set"
```

```
test_set <- cars[test_indices, ]
print(paste("There are ", nrow(test_set), " in test_set", sep = ""))
```

```
## [1] "There are 1 in test_set"
```

(ii) Report the k-fold CV MSE_{out} for all models using k=392.

```
MSE_out_list_392 <- sapply(1:7, \(i)
                        {k_fold_mse(cars, k = 392,
                                    formula = formula_list[[i]],
                                    func = function_list[[i]])})
MSE_df$"MSE_out_392" <- MSE_out_list_392
names(MSE_out_list_392) <- name_list
MSE_out_list_392
```

```
##            lm(cars_full)          lm(lm_reduced)     lm(cars_full_poly2)
##                11.293439               11.380040                8.610385
## lm(cars_reduced_poly2) lm(cars_reduced_poly6)        rpart(cars_full)
##                 8.787013                9.177932               12.769791
##    rpart(cars_reduced)
##                13.145150
```

(iii) When k=392, does the MSE_{out} of a model remain stable (same value) if you re-estimate it over and over again, or does it vary? (show a few repetitions for any model and decide!)

```
tentimes_kfold_392 <- sapply(1:7, \(i){
  mean(replicate(10, k_fold_mse(cars[sample(1:nrow(cars)),],
                                k = 392, formula = formula_list[[i]],
                                func = function_list[[i]])))})
MSE_df$"10 times k-fold MSE_out_392" <- tentimes_kfold_392
names(tentimes_kfold_392) <- name_list
tentimes_kfold_392
```

```
##            lm(cars_full)          lm(lm_reduced)     lm(cars_full_poly2)
```

```
##             11.293439             11.380040             8.610385
## lm(cars_reduced_poly2) lm(cars_reduced_poly6)       rpart(cars_full)
##              8.787013              9.177932             12.769791
##    rpart(cars_reduced)
##             13.145150
```

(iv) Looking at the fit error (MSE_{in}) and prediction error (MSE_{out}; k=392) of the full models versus their reduced counterparts (with the same training technique), does multicollinearity present in the full models seem to hurt their fit error and/or prediction error?

```r
MSE_df[c(1, 2, 6, 7), c("MSE_in", "10 times k-fold MSE_out_392")]
```

```
##                        MSE_in 10 times k-fold MSE_out_392
## lm(cars_full)       10.682122                    11.29344
## lm(lm_reduced)      10.971643                    11.38004
## rpart(cars_full)     9.155146                    12.76979
## rpart(cars_reduced)  9.501344                    13.14515
```

Although the differences may seem small, both in the lm() and rpart() cases, the reduce model's MSE_{in} and MSE_out are slightly higher than those of the full model.

(v) Look at the fit error and prediction error (k=392) of the reduced quadratic versus 6th order polynomial regressions — did adding more higher-order terms hurt the fit and/or predictions?

```r
MSE_df[c(4, 5), c("MSE_in", "10 times k-fold MSE_out_392")]
```

```
##                          MSE_in 10 times k-fold MSE_out_392
## lm(cars_reduced_poly2) 8.364546                    8.787013
## lm(cars_reduced_poly6) 8.254377                    9.177932
```

In the case of the sixth-degree polynomial model, the MSE_{in} is slightly lower compared to the quadratic polynomial model. However, in terms of MSE_out, the quadratic polynomial model has a lower value.