HW_Week16_108020033

Che-Wei, Chang

2023-05-30 helped by 108020024

Let's return yet again to the cars dataset we now understand quite well. Recall that it had several interesting issues such as non-linearity and multicollinearity. How do these issues affect prediction? We are also interested in model complexity and the difference in fit error versus prediction error. Let's setup what we need for this assignment (note: we will not use the cars_log dataset; we will return to the original, raw data for cars):

```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)
# IMPORTANT: Shuffle the rows of data in advance for this project!
set.seed(27935752) # use your own seed, or use this one to compare to next class notes
cars <- cars[sample(1:nrow(cars)),]
# 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)
```

A simple description of each formula:

cars_full: The full formula with all IVs in our original dataset

cars_reduced: The reduced formula after stepwise-VIF to eliminate collinear terms

cars_full_poly2: The full formula with quadratic terms

cars_reduced_poly2: The reduced formula with quadratic terms

cars_reduced_poly6: The reduced formula with upto 6th degree higher-order terms

Here are seven models (formula + estimation/training method) you must create and test in this project:

lm_full: A full model (cars_full) using linear regression

lm_reduced: A reduced model (cars_reduced) using linear regression

lm_poly2_full: A full quadratic model (cars_full_poly2) using linear regression lm_poly2_reduced: A reduced quadratic model (cars_reduced_poly2) using linear regression

lm_poly6_reduced: A reduced 6th order polynomial (cars_reduced_poly6) using linear regression

rt_full: A full model (cars_full) using a regression tree

rt_reduced: A reduced model (cars_reduced) using a regression tree

Question 1) Compute and report the in-sample fitting error (MSEin) 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; you can then apply that function to each model (feel free to ask us for help!). We will discuss these results later.

```
# Function to calculate in-sample fitting error (MSEin)
mse_in <- function(model) {
  predictions <- predict(model, newdata = cars)
  mse <- mean((cars$mpg - predictions)^2)
  return(mse)
}

# Linear regression models
lm_full <- lm(cars_full, data = cars)
lm_reduced <- lm(cars_reduced, data = cars)
lm_poly2_full <- lm(cars_full_poly2, data = cars)
lm_poly2_reduced <- lm(cars_reduced_poly2, data = cars)
lm_poly6_reduced <- lm(cars_reduced_poly6, data = cars)

# Regression tree models
library(rpart)
rt_full <- rpart(cars_full, data = cars)
rt_reduced <- rpart(cars_reduced, data = cars)

# Compute MSEin for each model
mse_lm_full <- mse_in(lm_full)
mse_lm_reduced <- mse_in(lm_reduced)
mse_lm_poly2_full <- mse_in(lm_poly2_full)
mse_lm_poly2_reduced <- mse_in(lm_poly2_reduced)
mse_lm_poly6_reduced <- mse_in(lm_poly6_reduced)
mse_rt_full <- mse_in(rt_full)
mse_rt_reduced <- mse_in(rt_reduced)

# Print the MSEin for each model
cat("MSEin - lm_full:", mse_lm_full, "\n")
```

```
## MSEin - lm_full: 10.68212
```

```
cat("MSEin - lm_reduced:", mse_lm_reduced, "\n")
```

```
## MSEin - lm_reduced: 10.97164
```

```
cat("MSEin - lm_poly2_full:", mse_lm_poly2_full, "\n")
```

```
## MSEin - lm_poly2_full: 7.91903
```

```r
cat("MSEin - lm_poly2_reduced:", mse_lm_poly2_reduced, "\n")
```

```
## MSEin - lm_poly2_reduced: 8.364546
```

```r
cat("MSEin - lm_poly6_reduced:", mse_lm_poly6_reduced, "\n")
```

```
## MSEin - lm_poly6_reduced: 8.254377
```

```r
cat("MSEin - rt_full:", mse_rt_full, "\n")
```

```
## MSEin - rt_full: 9.155146
```

```r
cat("MSEin - rt_reduced:", mse_rt_reduced, "\n")
```

```
## MSEin - rt_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 (did you remember to shuffle the data earlier?)

```r
# Split the data into training and test sets
set.seed(27935752)  # Use the same seed for consistency
train_indices <- sample(1:nrow(cars), 0.7 * nrow(cars))
train_data <- cars[train_indices, ]
test_data <- cars[-train_indices, ]
```

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
# Retrain the lm_reduced model on the training data
trained_model <- lm(cars_reduced, data = train_data)
summary(trained_model)$coefficients
```

```
##                     Estimate   Std. Error     t value     Pr(>|t|)
## (Intercept)     -19.386552584 4.9381209619  -3.9258967 1.099331e-04
## weight           -0.005828658 0.0003302789 -17.6476875 8.866457e-47
## acceleration      0.065048251 0.0829099431   0.7845651 4.334015e-01
## model_year        0.765849216 0.0602263948  12.7161724 2.647476e-29
## factor(origin)2   1.614507859 0.6255227814   2.5810537 1.038200e-02
## factor(origin)3   2.101306397 0.6120593407   3.4331743 6.909517e-04
```

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

```r
# Predict the mpg of the test dataset using the trained model
test_predictions <- predict(trained_model, newdata = test_data)
```

What is the in-sample mean-square fitting error (MSEin) of the trained model?

```r
# Calculate the in-sample fitting error (MSEin) of the trained model
mse_in_trained <- mse_in(trained_model)
cat("MSEin: ", mse_in_trained)
```

```
## MSEin:  11.06571
```

What is the out-of-sample mean-square prediction error (MSEout) of the test dataset?

```r
# Calculate the out-of-sample prediction error (MSEout) of the test dataset
mse_out <- mean((test_data$mpg - test_predictions)^2)
cat("MSEout: ", mse_out)
```

```
## MSEout:  11.37791
```

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

```r
# Create a data frame of the test set's actual mpg values, predicted mpg values, and predictive error
df <- data.frame(Actual = test_data$mpg, Predicted = test_predictions,
                 Error = test_data$mpg - test_predictions)

# Show the first several rows of the data frame
head(df)
```

```
##     Actual Predicted     Error
## 248   39.4  31.59557  7.804433
## 37    19.0  16.75076  2.249241
## 201   18.0  19.35238 -1.352376
## 103   26.0  28.13508 -2.135077
## 389   26.0  29.28920 -3.289201
## 100   18.0  20.39581 -2.395813
```

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 MSEout of the operation. Your function must accept a model, dataset and number of folds (k) but can also have whatever other parameters you wish.

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

```r
# Calculate mse_out across all folds
k_fold_mse <- function(dataset, k, lm_formula, actuals, int) {
  fold_pred_errors <- sapply(1:k, \(i) {
  fold_i_pe(i, k, dataset, lm_formula, actuals, int)
  })
  pred_errors <- unlist(fold_pred_errors)
  mean(pred_errors^2)
}

# Calculate prediction error for fold i out of k
fold_i_pe <- function(i, k, dataset, lm_formula, actuals, int) {
  folds <- cut(1:nrow(dataset), k, labels = FALSE)
  test_indices <- which(folds == i)
  test_set <- dataset[test_indices, ]
  train_set <- dataset[-test_indices, ]
  if (int == 0) {
    trained_model <- lm(lm_formula, train_set)
  }
  else {
    train_model <- rpart(dataset, cars)
  }
  predictions <- predict(trained_model, test_set)
  actuals[which(folds == i)] - predictions
}

# Testing for linear models
k_fold_mse(cars, k = 10, cars_full, cars$mpg, 0)
```

```
## [1] 11.26246
```

```r
k_fold_mse(cars, k = 10, cars_reduced, cars$mpg, 0)
```

```
## [1] 11.41586
```

```r
k_fold_mse(cars, k = 10, cars_full_poly2, cars$mpg, 0)
```

```
## [1] 8.599373
```

```r
k_fold_mse(cars, k = 10, cars_reduced_poly2, cars$mpg, 0)
```

```
## [1] 8.818607
```

```r
k_fold_mse(cars, k = 10, cars_reduced_poly6, cars$mpg, 0)
```

```
## [1] 9.267369
```

```r
# Testing for regression trees
k_fold_mse(cars, k = 10, cars_full, cars$mpg, 1)
```

```
## [1] 11.06571
```

```r
k_fold_mse(cars, k = 10, cars_reduced, cars$mpg, 0)
```

```
## [1] 11.41586
```

(ii). For all the models, which is bigger — the fit error (MSEin) or the prediction error (MSEout)? (optional: why do you think that is?)

```r
# By the result, we can see that MSEout of all results are bigger than MSEin of all results.
```

(iii). Does the 10-fold MSEout 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!)

```r
test_1 <- cars[sample(1:nrow(cars)), ]
test_2 <- cars[sample(1:nrow(cars)), ]
test_3 <- cars[sample(1:nrow(cars)), ]

k_fold_mse(test_1, k = 10, cars_full, test_1$mpg, 0)
```

```
## [1] 11.40131
```

```r
k_fold_mse(test_2, k = 10, cars_full, test_2$mpg, 0)
```

```
## [1] 11.41532
```

```r
k_fold_mse(test_3, k = 10, cars_full, test_3$mpg, 0)
```

```
## [1] 11.52072
```

```r
# By the result, we can find that there is a lttle different between these three values.
# Therefore, they doesn't remain stable.
```

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?

```r
# In each iteration, 391 rows are in the training dataset and 1 row is in test dataset.
```

(ii). Report the k-fold CV MSEout for all models using k=392.

```r
# Testing for linear models
k_fold_mse(cars, k = 392, cars_full, cars$mpg, 0)
```

```
## [1] 11.29344
```

```r
k_fold_mse(cars, k = 392, cars_reduced, cars$mpg, 0)
```

```
## [1] 11.38004
```

```r
k_fold_mse(cars, k = 392, cars_full_poly2, cars$mpg, 0)
```

```
## [1] 8.610385
```

```r
k_fold_mse(cars, k = 392, cars_reduced_poly2, cars$mpg, 0)
```

```
## [1] 8.787013
```

```r
k_fold_mse(cars, k = 392, cars_reduced_poly6, cars$mpg, 0)
```

```
## [1] 9.177932
```

```r
# Testing for regression trees
k_fold_mse(cars, k = 392, cars_full, cars$mpg, 1)
```

```
## [1] 11.06571
```

```r
k_fold_mse(cars, k = 392, cars_reduced, cars$mpg, 1)
```

```
## [1] 11.06571
```

(iii). When k=392, does the MSEout 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!)

```r
test_4 <- cars[sample(1:nrow(cars)), ]
test_5 <- cars[sample(1:nrow(cars)), ]
test_6 <- cars[sample(1:nrow(cars)), ]

k_fold_mse(test_4, k = 392, cars_full, test_4$mpg, 0)
```

```
## [1] 11.29344
```

```r
k_fold_mse(test_5, k = 392, cars_full, test_5$mpg, 0)
```

```
## [1] 11.29344
```

```r
k_fold_mse(test_6, k = 392, cars_full, test_6$mpg, 0)
```

```
## [1] 11.29344
```

```r
# By the result, we can find that these three values are same as each other.
# Therefore, they remain stable.
```

(iv). Looking at the fit error (MSEin) and prediction error (MSEout; 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? (optional: if not, then when/why are analysts so scared of multicollinearity?)

```r
# By the result above, we can find that MSEin and MSEout of the full models versus
# their reduced counterparts doesn't change a lot. Therefore, multicollinearity
# present in the full models doesn't hurt their fit error and prediction error.
```

(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? (optional: What does this imply? Does adding complex terms improve fit or prediction?)

```r
# By the result above, we can conclude that adding more higher-order terms hurt the
# fit and predictions.
```