bacs_hw14

110071010

2024-05-23

Setup

```
# Load the data and remove missing values
cars <- read.table("auto-data.txt", header=FALSE, na.strings = "?")
names(cars) <- c("mpg", "cylinders", "displacement", "horsepower", "wei
ght", "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(111555777) # 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 + acc
eleration +
                   model_year + factor(origin)
cars_reduced <- mpg ~ weight + acceleration + model_year + factor(origi
n)
cars_full_poly2 <- mpg ~ poly(cylinders, 2) + poly(displacement, 2) + p
oly(horsepower, 2) + poly(weight, 2) + poly(acceleration, 2) + model_ye
ar + factor(origin)
cars_reduced_poly2 <- mpg ~ poly(weight, 2) + poly(acceleration,2) + mo
del_year + factor(origin)
cars_reduced_poly6 <- mpg ~ poly(weight, 6) + poly(acceleration,6) + mo
del_year + factor(origin)

library(rpart) # for regression trees

lm_full <- lm(formula = cars_full, data=cars)
lm_reduced <- lm(formula = cars_reduced, data=cars)
lm_poly2_full <- lm(formula = cars_full_poly2, data=cars)
lm_poly2_reduced <- lm(formula = cars_reduced_poly2, data=cars)
lm_poly6_reduced <- lm(formula = cars_reduced_poly6, data=cars)
rt_full <- rpart(formula = cars_full, data=cars)
rt_reduced <- rpart(formula = cars_reduced, data=cars)
```

Question 1

Compute and report the in-sample fitting error (MSEin) of all the models described above. It will be easier to first write a function called mse_in(...) that returns the fitting error of a single estimated model; you can apply that function to each model (feel free to ask us for help!). We will discuss these results later.

```
mse_in <- function(model){
  mean(residuals(model)^2)
  # mean((cars$mpg - fitted(model))^2)
}

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)

cat("mse_lm_full : ", mse_lm_full, "\n")
cat("mse_lm_reduced : ", mse_lm_reduced, "\n")
cat("mse_lm_poly2_full : ", mse_lm_poly2_full, "\n")
cat("mse_lm_poly2_reduced : ", mse_lm_poly2_reduced, "\n")
cat("mse_lm_poly6_reduced : ", mse_lm_poly6_reduced, "\n")
cat("mse_rt_full : ", mse_rt_full, "\n")
cat("mse_rt_reduced : ", mse_rt_reduced, "\n")

## mse_lm_full :  10.68212
## mse_lm_reduced :  10.97164
## mse_lm_poly2_full :  7.91903
## mse_lm_poly2_reduced :  8.364546
## mse_lm_poly6_reduced :  8.254377
## mse_rt_full :  9.155146
## mse_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:

2a

Split the data into 70:30 for training:test (did you remember to shuffle the data earlier?)

```
set.seed(111555777)

# Split the data into 70:30 for training:test
```

```r
train_indices <- sample(1:nrow(cars), size = 0.70*nrow(cars))
train_set <- cars[train_indices,]
test_set <- cars[-train_indices,]
```

2b

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 just the training dataset
trained_model <- lm(formula = cars_reduced, data = train_set)

# Show the coefficients of the trained model
summary(trained_model)$coefficients
```

```
##                     Estimate    Std. Error     t value     Pr(>|t|)
## (Intercept)     -16.774070479 4.9272440811  -3.4043514 7.647307e-04
## weight           -0.005964948 0.0003428965 -17.3957686 6.983693e-46
## acceleration      0.013635648 0.0849423083   0.1605283 8.725858e-01
## model_year        0.747895113 0.0593500574  12.6014219 6.607599e-29
## factor(origin)2   1.939320199 0.6393551468   3.0332441 2.656790e-03
## factor(origin)3   2.168933983 0.6300771348   3.4423309 6.689422e-04
```

2c

Use the trained_model model to predict the mpg of the test dataset. What is the in-sample mean-square fitting error (MSEin) of the trained model? What is the out-of-sample mean-square prediction error (MSEout) of the test dataset?

```r
# Function to calculate in-sample MSE
mse_in <- function(model) {
  mean(residuals(model)^2)
}

# Use the trained_model model to predict the mpg of the test dataset
mpg_predicted <- predict(trained_model, newdata = test_set)

# Calculate the in-sample MSE (MSEin) of the trained model
cat("MSEin : ", mse_in(trained_model), "\n")

# Calculate the out-of-sample MSE (MSEout) of the test dataset
mpg_actual <- test_set$mpg
pred_err <- mpg_actual - mpg_predicted
cat("MSEout : ", mean((mpg_predicted - mpg_actual)^2), "\n")
```

```
## MSEin :  11.1779
## MSEout :  10.6634
```

2d

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.

```
data.frame(
  "actual mpg" = mpg_actual,
  "predicted mpg" = mpg_predicted,
  "predictive error" = pred_err
) |> head()
```

```
##     actual.mpg predicted.mpg predictive.error
## 144       26.0      26.98782      -0.98782442
## 214       13.0      16.04172      -3.04172145
## 368       28.0      29.28190      -1.28189777
## 304       31.8      32.69119      -0.89118678
## 185       25.0      24.92728       0.07271717
## 109       20.0      26.65617      -6.65616742
```

Question 3

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

3a

Write a function that performs k-fold cross-validation. 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.

```
# Function to calculate prediction errors for fold i
fold_i_pe <- function(i, k, dataset, model_function, formula) {
  # Split the data into k folds
  folds <- cut(seq(1, nrow(dataset)), breaks = k, labels = FALSE)

  # Identify test and training indices for the ith fold
  test_indices <- which(folds == i, arr.ind = TRUE)
  test_set <- dataset[test_indices, ]
  train_set <- dataset[-test_indices, ]

  # Train the model on the training set
  trained_model <- model_function(formula, data = train_set)

  # Predict on the test set
  predictions <- predict(trained_model, newdata = test_set)

  # Calculate prediction errors
```

```
  actuals <- test_set$mpg
  prediction_errors <- actuals - predictions

  return(prediction_errors)
}

# Function to calculate mean squared error across all folds
k_fold_mse <- function(dataset, k = 10, model_function, formula) {
  # Calculate prediction errors for each fold
  fold_pred_errors <- sapply(1:k, function(i) {
    fold_i_pe(i, k, dataset, model_function, formula)
  })

  # Combine all prediction errors into a single vector
  pred_errors <- unlist(fold_pred_errors)

  # Calculate and return the mean squared error
  mse <- mean(pred_errors^2)
  return(mse)
}
```

i

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

```
mse_full <- k_fold_mse(dataset=cars, k=10,lm, formula=cars_full)
mse_reduced <- k_fold_mse(dataset=cars, k=10,lm, formula=cars_reduced)
mse_full_poly2 <- k_fold_mse(dataset=cars, k=10,lm, formula=cars_full_p
oly2)
mse_reduced_poly2 <- k_fold_mse(dataset=cars, k=10,lm, formula=cars_red
uced_poly2)
mse_reduced_poly6 <- k_fold_mse(dataset=cars, k=10,lm, formula=cars_red
uced_poly6)
mse_rpart_reduced_poly2 <- k_fold_mse(dataset=cars, k=10,rpart, formula
=cars_reduced_poly2)
mse_rpart_reduced_poly6 <- k_fold_mse(dataset=cars, k=10,rpart, formula
=cars_reduced_poly6)

data.frame(Model = c("lm_full", "lm_reduced", "lm_full_poly2", "lm_redu
ced_poly2", "lm_reduced_poly6", "rt_reduced_poly2", "rt_reduced_poly6"),
           MSEout = c(mse_full, mse_reduced, mse_full_poly2, mse_reduce
d_poly2,
             mse_reduced_poly6, mse_rpart_reduced_poly2, mse_rpart_redu
ced_poly6)
)

##               Model    MSEout
## 1           lm_full 11.227264
## 2        lm_reduced 11.350700
```

```
## 3    lm_full_poly2  8.677378
## 4 lm_reduced_poly2  8.807537
## 5 lm_reduced_poly6  9.379082
## 6 rt_reduced_poly2 11.865570
## 7 rt_reduced_poly6 11.806473
```

ii

Question

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

Answer

MSEout > MSEin, as there are overfitting problems for in-sample.

iii

Question

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
set.seed(111555777)

n_repeats <- 5
mse_out_repeats <- replicate(n_repeats, {
  k_fold_mse(cars, k = 10, model_function = lm, formula = cars_full)
})

# Display the results
data.frame(Repetition = 1:n_repeats, MSEout = mse_out_repeats)
```

```
##   Repetition   MSEout
## 1          1 11.22726
## 2          2 11.22726
## 3          3 11.22726
## 4          4 11.22726
## 5          5 11.22726
```

Answer

Yes, the 10-fold MSEouts remain the same after re-estimations

3b

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

i

Question

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

Answer

Test dataset: 1 observation

Training dataset: 392 - 1 = 391 observations

ii

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

```
mse_full <- k_fold_mse(dataset=cars, k=392,lm, formula=cars_full)
mse_reduced <- k_fold_mse(dataset=cars, k=392,lm, formula=cars_reduced)
mse_full_poly2 <- k_fold_mse(dataset=cars, k=392,lm, formula=cars_full_
poly2)
mse_reduced_poly2 <- k_fold_mse(dataset=cars, k=392,lm, formula=cars_re
duced_poly2)
mse_reduced_poly6 <- k_fold_mse(dataset=cars, k=392,lm, formula=cars_re
duced_poly6)
mse_rpart_reduced_poly2 <- k_fold_mse(dataset=cars, k=392,rpart, formul
a=cars_reduced_poly2)
mse_rpart_reduced_poly6 <- k_fold_mse(dataset=cars, k=392,rpart, formul
a=cars_reduced_poly6)

data.frame(Model = c("lm_full", "lm_reduced", "lm_full_poly2", "lm_redu
ced_poly2", "lm_reduced_poly6", "rt_reduced_poly2", "rt_reduced_poly6"),
           MSEout = c(mse_full, mse_reduced, mse_full_poly2, mse_reduce
d_poly2,
             mse_reduced_poly6, mse_rpart_reduced_poly2, mse_rpart_redu
ced_poly6)
)

##               Model    MSEout
## 1           lm_full 11.293439
## 2        lm_reduced 11.380040
## 3     lm_full_poly2  8.610385
## 4 lm_reduced_poly2  8.787013
## 5 lm_reduced_poly6  9.177932
## 6 rt_reduced_poly2 13.303589
## 7 rt_reduced_poly6 13.270311
```

iii

Question

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!)

```
set.seed(111555777)

n_repeats <- 5
mse_out_repeats <- replicate(n_repeats, {
  k_fold_mse(cars, k = 392, model_function = lm, formula = cars_full)
})

# Display the results
data.frame(Repetition = 1:n_repeats, MSEout = mse_out_repeats)

##   Repetition   MSEout
## 1          1 11.29344
## 2          2 11.29344
## 3          3 11.29344
## 4          4 11.29344
## 5          5 11.29344
```

Answer

Yes, the 392-fold MSEouts remain the same after re-estimations

iv

Question

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?

```
# Calculate in-sample MSE for full and reduced models
mse_lm_full_in <- mse_in(lm_full)
mse_lm_reduced_in <- mse_in(lm_reduced)

# Set seed for reproducibility
set.seed(111555777)

# Calculate out-of-sample MSE for full and reduced models
mse_lm_full_out <- k_fold_mse(cars, 392, lm, cars_full)
mse_lm_reduced_out <- k_fold_mse(cars, 392, lm, cars_reduced)

# Display results
data.frame(Model = c("Full Model", "Reduced Model"),
           MSEin = c(mse_lm_full_in, mse_lm_reduced_in),
           MSEout = c(mse_lm_full_out, mse_lm_reduced_out)
)
```

```
##           Model    MSEin   MSEout
## 1    Full Model 10.68212 11.29344
## 2 Reduced Model 10.97164 11.38004
```

Answer

The reduced model has a lower in-sample MSE compared to the full model, suggesting that removing collinear terms improved the fit of the model on the training data. The reduced model also has a lower out-of-sample MSE compared to the full model. This indicates that the reduced model performs better in predicting new data.

v

Question

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?

```
# Calculate in-sample MSE for both models
mse_poly2_reduced_in <- mse_in(lm_poly2_reduced)
mse_poly6_reduced_in <- mse_in(lm_poly6_reduced)

# Set seed for reproducibility
set.seed(111555777)

# Calculate out-of-sample MSE for both models using k-fold CV
mse_poly2_reduced_out <- k_fold_mse(cars, 392, lm, cars_reduced_poly2)
mse_poly6_reduced_out <- k_fold_mse(cars, 392, lm, cars_reduced_poly6)

# Display results
data.frame(Model = c("Reduced Quadratic", "Reduced 6th Order Polynomial
"),
           MSEin = c(mse_poly2_reduced_in, mse_poly6_reduced_in),
           MSEout = c(mse_poly2_reduced_out, mse_poly6_reduced_out)
)
```

```
##                          Model    MSEin   MSEout
## 1            Reduced Quadratic 8.364546 8.787013
## 2 Reduced 6th Order Polynomial 8.254377 9.177932
```

Answer

Adding more higher-order terms may slightly improve the fit to the training data (lower MSEin) but can hurt the model's ability to generalize to new data (higher MSEout). The higher MSEout of the 6th order polynomial might stem from overfitting.