HW14

113078506

2025-05-22

- Gen AI Usage: I use Gen AI to refine my grammar, verify my reasoning and adjust to cleaner code.
- Students who helped: 113078505, 113078502, and 113078514, helped with conclusion reasoning.

Set up:

```
# 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)
library(rpart)
# seven models:
# lm_full: A full model (cars_full) using linear regression
lm_full <- lm(cars_full, data = cars)
# lm_reduced: A reduced model (cars_reduced) using linear regression
lm_reduced <- lm(cars_reduced, data = cars)
# lm_poly2_full: A full quadratic model (cars_full_poly2) using linear regression
lm_poly2_full <- lm(cars_full_poly2, data = cars)
# lm_poly2_reduced: A reduced quadratic model (cars_reduced_poly2) using linear regression
lm_poly2_reduced <- lm(cars_reduced_poly2, data = cars)
# lm_poly6_reduced: A reduced 6th order polynomial (cars_reduced_poly6) using linear regression
lm_poly6_reduced <- lm(cars_reduced_poly6, data = cars)
# rt_full: A full model (cars_full) using a regression tree
rt_full <- rpart(cars_full, data = cars)
# rt_reduced: A reduced model (cars_reduced) using a regression tree
rt_reduced <- rpart(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, data) {
  y_true <- data$mpg
  y_pred <- predict(model, newdata = data)
  mean((y_true - y_pred)^2)
}

msein_results <- c(
  lm_full = mse_in(lm_full, cars),
  lm_reduced = mse_in(lm_reduced, cars),
  lm_poly2_full = mse_in(lm_poly2_full, cars),
  lm_poly2_reduced = mse_in(lm_poly2_reduced, cars),
  lm_poly6_reduced = mse_in(lm_poly6_reduced, cars),
  rt_full = mse_in(rt_full, cars),
  rt_reduced = mse_in(rt_reduced, cars)
)

library(knitr)

msein_df <- data.frame(
  Model = names(msein_results),
  MSEin = as.numeric(msein_results)
)

kable(msein_df, digits = 4, caption = "In-sample MSE (MSEin) for All Models")
```

Table 1: In-sample MSE (MSEin) for All Models

Model	MSEin
lm_full	10.6821
lm_reduced	10.9716
lm_poly2_full	7.9190
lm_poly2_reduced	8.3645
lm_poly6_reduced	8.2544
rt_full	9.1551
rt_reduced	9.5013

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
set.seed(27935752)
train_indices <- sample(1:nrow(cars), size=0.70*nrow(cars))
train_set <- cars[train_indices,]

test_set <- 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
trained_model <- lm(cars_reduced, data=train_set)

kable(summary(trained_model)$coefficients, digits = 4, caption = "Coefficients of the trained model")
```

Table 2: Coefficients of the trained model

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-19.3866	4.9381	-3.9259	0.0001
weight	-0.0058	0.0003	-17.6477	0.0000
acceleration	0.0650	0.0829	0.7846	0.4334
model_year	0.7658	0.0602	12.7162	0.0000
factor(origin)2	1.6145	0.6255	2.5811	0.0104
factor(origin)3	2.1013	0.6121	3.4332	0.0007

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

```r
mpg_predicted <- predict(trained_model, test_set)

# i. What is the in-sample mean-square fitting error (MSEin) of the trained model?
mse_insample <- mse_in(trained_model, train_set)

# ii. What is the out-of-sample mean-square prediction error (MSEout) of the test dataset?
mse_out <- function(model, data) {
  y_true <- data$mpg
  y_pred <- predict(model, newdata = data)
  mean((y_true - y_pred)^2)
}
mse_outsample <- mse_out(trained_model, test_set)

mse_result <- data.frame(
Metric = c("MSE_in (training)", "MSE_out (test)"), Value = round(c(mse_insample, mse_outsample), 3)
)

kable(mse_result, digits = 4, caption = "In-sample and Out-of-sample MSE")
```

Table 3: In-sample and Out-of-sample MSE

Metric	Value
MSE_in (training)	10.931
MSE_out (test)	11.378

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

```
test_results <- data.frame(
  actual_mpg = test_set$mpg,
  predicted_mpg = predict(trained_model, newdata = test_set)
)
test_results$error <- test_results$actual_mpg - test_results$predicted_mpg
kable(head(test_results), digits = 2, caption = "Test Set Predictions and Errors (First 6 Rows)")
```

Table 4: Test Set Predictions and Errors (First 6 Rows)

	actual_mpg	predicted_mpg	error
248	39.4	31.60	7.80
37	19.0	16.75	2.25
201	18.0	19.35	-1.35
103	26.0	28.14	-2.14
389	26.0	29.29	-3.29
100	18.0	20.40	-2.40

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.

```
# fold_i_pe
fold_i_pe <- function(i, k, dataset, formula, method = "lm") {
  fold_ids <- cut(sample(1:nrow(dataset)), breaks = k, labels = FALSE)
  test_indices <- which(fold_ids == i)
  test_set <- dataset[test_indices, ]
  train_set <- dataset[-test_indices, ]

  if (method == "lm") {
    trained_model <- lm(formula, data = train_set)
  } else if (method == "rpart") {
    trained_model <- rpart(formula, data = train_set)
  } else {
```

```
    stop("Unsupported method")
  }

  predictions <- predict(trained_model, newdata = test_set)
  actuals <- test_set$mpg
  actuals - predictions  # out
}


# k_fold_mse
k_fold_mse <- function(dataset, formula, k = 10, method = "lm") {
  fold_pred_errors <- sapply(1:k, function(i) {
    fold_i_pe(i, k, dataset, formula, method)
  })
  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 MSEout for all models.

```
cv_lm_full <- k_fold_mse(cars, cars_full, k = 10, method = "lm")
cv_lm_reduced <- k_fold_mse(cars, cars_reduced, k = 10, method = "lm")
cv_lm_poly2_full <- k_fold_mse(cars, cars_full_poly2, k = 10, method = "lm")
cv_lm_poly2_reduced <- k_fold_mse(cars, cars_reduced_poly2, k = 10, method = "lm")
cv_lm_poly6_reduced <- k_fold_mse(cars, cars_reduced_poly6, k = 10, method = "lm")
cv_rt_full <- k_fold_mse(cars, cars_full, k = 10, method = "rpart")
cv_rt_reduced <- k_fold_mse(cars, cars_reduced, k = 10, method = "rpart")

cv_results <- data.frame(
  Model = c("lm_full", "lm_reduced", "lm_poly2_full", "lm_poly2_reduced", "lm_poly6_reduced", "rt_full"
  MSEout_10fold = c(cv_lm_full, cv_lm_reduced, cv_lm_poly2_full, cv_lm_poly2_reduced, cv_lm_poly6_reduc
)

library(knitr)
kable(cv_results, digits = 4, caption = "10-Fold Cross-Validation MSEout for All Models")
```

Table 5: 10-Fold Cross-Validation MSEout for All Models

Model	MSEout_10fold
lm_full	11.1887
lm_reduced	12.5063
lm_poly2_full	7.6853
lm_poly2_reduced	9.9752
lm_poly6_reduced	8.8964
rt_full	12.1031
rt_reduced	11.5063

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

In all models, MSEout is generally larger than MSEin. MSEin reflects training error, while MSEout reflects prediction error on unseen data, which is often worse due to overfitting or variance in the model.

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!)

```
set.seed(123)
repeat_mse <- replicate(5, k_fold_mse(cars, cars_reduced, k = 10, method = "lm"))
repeat_mse
```

```
## [1] 13.536429  9.375274 12.353257 11.712107  9.610050
```

```
sd(repeat_mse)
```

```
## [1] 1.791632
```

No, the 10-fold MSEout is not stable when re-estimated multiple times. In our five repetitions using the lm_reduced model, the MSEout values varied significantly: [13.54, 9.38, 12.35, 11.71, 9.61]. The standard deviation was 1.79, which indicates a noticeable amount of fluctuation. This variation is due to the randomness in fold assignments during cross-validation. It shows that model evaluation using k-fold CV may be sensitive to how the data is split. A standard deviation of 1.79 is considered relatively unstable when the average MSEout is around 11. This means that the cross-validation results can vary by 15–20% depending on how the folds are randomly assigned. For model evaluation, we prefer smaller variance to ensure reliable comparisons.

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?

If k=392, then i=1. Training set will be 391 rows Test set will be 1 row

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

```
cv392_lm_full <- k_fold_mse(cars, cars_full, k = 392, method = "lm")
cv392_lm_reduced <- k_fold_mse(cars, cars_reduced, k = 392, method = "lm")
cv392_lm_poly2_full <- k_fold_mse(cars, cars_full_poly2, k = 392, method = "lm")
cv392_lm_poly2_reduced <- k_fold_mse(cars, cars_reduced_poly2, k = 392, method = "lm")
cv392_lm_poly6_reduced <- k_fold_mse(cars, cars_reduced_poly6, k = 392, method = "lm")
cv392_rt_full <- k_fold_mse(cars, cars_full, k = 392, method = "rpart")
cv392_rt_reduced <- k_fold_mse(cars, cars_reduced, k = 392, method = "rpart")

cv_results_392 <- data.frame(
  Model = c("lm_full", "lm_reduced", "lm_poly2_full", "lm_poly2_reduced", "lm_poly6_reduced", "rt_full"
```

```
  MSEout_LOOCV = c(cv392_lm_full, cv392_lm_reduced, cv392_lm_poly2_full, cv392_lm_poly2_reduced,
                   cv392_lm_poly6_reduced, cv392_rt_full, cv392_rt_reduced)
)
kable(cv_results_392, digits = 4, caption = "LOOCV (k=392) MSEout for All Models")
```

Table 6: LOOCV (k=392) MSEout for All Models

Model	MSEout_LOOCV
lm_full	10.2935
lm_reduced	12.5971
lm_poly2_full	10.1585
lm_poly2_reduced	8.5468
lm_poly6_reduced	9.9110
rt_full	11.3692
rt_reduced	12.2866

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!)

```
repeat_loocv <- replicate(5, k_fold_mse(cars, cars_reduced, k = 392, method = "lm"))
repeat_loocv
```

```
## [1] 10.12274 11.40793 12.53299 10.07144 10.59203
```

```
sd(repeat_loocv)
```

```
## [1] 1.036801
```

No, the MSEout is not perfectly stable even when using LOOCV (k = 392). In our five repetitions, the MSEout values ranged from 10.07 to 12.53, with a standard deviation of 1.04. Although LOOCV reduces bias due to using almost all data for training, it still shows variability because the fold assignment is randomized each time. This implies that even LOOCV has non-trivial variance and should not be assumed perfectly consistent.

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?)

```
compare_mc <- data.frame(
  Model = c("lm_full", "lm_reduced", "rt_full", "rt_reduced"),
  MSEin = c(mse_in(lm_full, cars), mse_in(lm_reduced, cars),
            mse_in(rt_full, cars), mse_in(rt_reduced, cars)),
  MSEout = c(cv392_lm_full, cv392_lm_reduced, cv392_rt_full, cv392_rt_reduced)
)
kable(compare_mc, digits = 4, caption = "Effect of Multicollinearity (Full vs. Reduced Models)")
```

Table 7: Effect of Multicollinearity (Full vs. Reduced Models)

Model	MSEin	MSEout
lm_full	10.6821	10.2935
lm_reduced	10.9716	12.5971
rt_full	9.1551	11.3692
rt_reduced	9.5013	12.2866

Multicollinearity doesn't always hurt prediction accuracy (MSEout). In fact, the full model here performs better. But analysts fear multicollinearity because it causes unstable coefficients, unreliable statistical significance, and poor interpretability, even when prediction error is low.

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?)

```
compare_poly <- data.frame(
  Model = c("lm_poly2_reduced", "lm_poly6_reduced"),
  MSEin = c(mse_in(lm_poly2_reduced, cars), mse_in(lm_poly6_reduced, cars)),
  MSEout = c(cv392_lm_poly2_reduced, cv392_lm_poly6_reduced)
)
kable(compare_poly, digits = 4, caption = "Effect of Adding Higher-Order Polynomial Terms")
```

Table 8: Effect of Adding Higher-Order Polynomial Terms

Model	MSEin	MSEout
lm_poly2_reduced	8.3645	8.5468
lm_poly6_reduced	8.2544	9.9110

Adding more complex terms (like 6th-degree polynomials) may slightly improve the fit on training data but can hurt prediction accuracy due to overfitting. This suggests that more complex models do not always generalize better and may model noise rather than signal.