BACS - HW16

109006241, helped by 109006217

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):

```
# 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 described above and needed for the models below:

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:

i. lm_full: A full model (cars_full) using linear regression

ii. lm_reduced: A reduced model (cars_reduced) using linear regression

iii. lm_poly2_full: A full quadratic model (cars_full_poly2) using linear regression

iv. lm_poly2_reduced: A reduced quadratic model (cars_reduced_poly2) using linear regression

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

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

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

```
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)
rt_full = rpart(cars_full, data=cars)
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 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.

```
mse_in = function(model, data) {
  y_true = data$mpg
  y_hat = fitted(model)
  mse = mean((y_true - y_hat)^2)
  return (mse)
}

mse_in_tree = function(model, data) {
  y_true = data$mpg
  y_hat = predict(model, cars)
  mse = mean((y_true - y_hat)^2)
  return (mse)
}

cat(" MSEin of lm_full:", mse_in(lm_full, cars), "\n",
     "MSEin of lm_reduced:", mse_in(lm_reduced, cars), "\n",
     "MSEin of lm_poly2_full:", mse_in(lm_poly2_full, cars), "\n",
     "MSEin of lm_poly2_reduced:", mse_in(lm_poly2_reduced, cars), "\n",
     "MSEin of lm_poly6_reduced:", mse_in(lm_poly6_reduced, cars), "\n",
     "MSEin of rt_full:", mse_in_tree(rt_full, cars), "\n",
     "MSEin of rt_reduced:", mse_in_tree(rt_reduced, cars))
```

```
##  MSEin of lm_full: 10.68212
```

```
##  MSEin of lm_reduced: 10.97164
##  MSEin of lm_poly2_full: 7.91903
##  MSEin of lm_poly2_reduced: 8.364546
##  MSEin of lm_poly6_reduced: 8.254377
##  MSEin of rt_full: 9.155146
##  MSEin of 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?)

```
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.

```
trained_model = lm(cars_reduced, data=train_set)
trained_model$coefficients
```

```
##     (Intercept)          weight    acceleration      model_year factor(origin)2
##   -19.884850373    -0.005691225     0.059620087     0.772666424     2.218726662
## factor(origin)3
##     2.148123177
```

c. 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?

```
y_true = test_set$mpg
y_hat = predict(trained_model, test_set)
mse_out = mean((y_true - y_hat)^2)
cat(" In-sample mean-square fitting error (MSEin) of the trained model:", mse_in(trained_model, train_s
    "Out-of-sample mean-square prediction error (MSEout) of the test dataset:", mse_out)
```

```
##  In-sample mean-square fitting error (MSEin) of the trained model: 12.1074
##  Out-of-sample mean-square prediction error (MSEout) of the test dataset: 8.505918
```

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

```
pred_err = y_true - y_hat
df = data.frame(Actual=y_true,
                Predicted=y_hat,
                Difference=pred_err)
head(df, 5)
```

```
##     Actual Predicted  Difference
## 372     29  30.05738 -1.05737552
## 214     13  16.47532 -3.47532289
## 81      22  23.07057 -1.07057048
## 384     38  35.33296  2.66703558
## 85      27  26.92741  0.07258502
```

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 = function(i, k, dataset, model, type) {

  folds = cut(1:nrow(dataset), k, labels = FALSE)
  test_indices = which(folds == i)
  test_set = dataset[test_indices,]
  train_set = dataset[-test_indices,]

  if(type=="lm") {
    trained_model = lm(model, data=train_set)
  } else {
    trained_model = rpart(model, data=train_set)
  }

  actuals = test_set$mpg
  predictions = predict(trained_model, test_set)
  pred_err = actuals - predictions

  return (pred_err)
}

k_fold_mse = function(model, dataset, k=10, type) {
  fold_pred_errors = sapply(1:k, \(i) {
    fold_i_pe(i, k, dataset, model, type)
  })
  pred_errors = unlist(fold_pred_errors)
  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.

```
cat(" 10-fold CV MSEout of lm_full:", k_fold_mse(cars_full, cars, 10, "lm"), "\n",
     "10-fold CV MSEout of lm_reduced:", k_fold_mse(cars_reduced, cars, 10, "lm"), "\n",
     "10-fold CV MSEout of lm_poly2_full:", k_fold_mse(cars_full_poly2, cars, 10, "lm"), "\n",
     "10-fold CV MSEout of lm_poly2_reduced:", k_fold_mse(cars_reduced_poly2, cars, 10, "lm"), "\n",
     "10-fold CV MSEout of lm_poly6_reduced:", k_fold_mse(cars_reduced_poly6, cars, 10, "lm"), "\n",
     "10-fold CV MSEout of rt_full:", k_fold_mse(cars_full, cars, 10, "tree"), "\n",
     "10-fold CV MSEout of rt_reduced:", k_fold_mse(cars_reduced, cars, 10, "tree"))
```

```
##  10-fold CV MSEout of lm_full: 11.26246
##  10-fold CV MSEout of lm_reduced: 11.41586
##  10-fold CV MSEout of lm_poly2_full: 8.599373
##  10-fold CV MSEout of lm_poly2_reduced: 8.818607
##  10-fold CV MSEout of lm_poly6_reduced: 9.267369
##  10-fold CV MSEout of rt_full: 13.34222
##  10-fold CV MSEout of rt_reduced: 13.47627
```

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

> The prediction error (MSEout) is bigger than the fit error (MSEin), because the models are trained on the training data, meaning that it is more "familiar" with the training data, so, when used to predict a new data that it hasn't seen before, the prediction power will be lower.

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(NULL) # Unset seed
mse_values = c()
for(i in 1:10) {
  data = cars[sample(1:nrow(cars)),]
  mse = k_fold_mse(cars_full_poly2, data, 10, "lm")
  mse_values = append(mse_values, mse)
  cat("MSEout of iteration", i, ":", mse, "\n")
}
```

```
## MSEout of iteration 1 : 8.679855
## MSEout of iteration 2 : 8.772636
## MSEout of iteration 3 : 8.589363
## MSEout of iteration 4 : 8.659903
## MSEout of iteration 5 : 8.439724
## MSEout of iteration 6 : 8.651239
## MSEout of iteration 7 : 8.631559
## MSEout of iteration 8 : 8.788056
## MSEout of iteration 9 : 8.511721
## MSEout of iteration 10 : 8.601463
```

```
cat(" Mean MSEout =", mean(mse_values), "\n",
     "Standard deviation =", sd(mse_values))
```

```
##  Mean MSEout = 8.632552
##  Standard deviation = 0.1063306
```

They are pretty much similar, but not exactly the same.

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?

Training dataset: 391 rows, test dataset: 1 row.

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

```
set.seed(27935752)
cat(" 392-fold CV MSEout of lm_full:", k_fold_mse(cars_full, cars, 392, "lm"), "\n",
     "392-fold CV MSEout of lm_reduced:", k_fold_mse(cars_reduced, cars, 392, "lm"), "\n",
     "392-fold CV MSEout of lm_poly2_full:", k_fold_mse(cars_full_poly2, cars, 392, "lm"), "\n",
     "392-fold CV MSEout of lm_poly2_reduced:", k_fold_mse(cars_reduced_poly2, cars, 392, "lm"), "\n",
     "392-fold CV MSEout of lm_poly6_reduced:", k_fold_mse(cars_reduced_poly6, cars, 392, "lm"), "\n",
     "392-fold CV MSEout of rt_full:", k_fold_mse(cars_full, cars, 392, "tree"), "\n",
     "392-fold CV MSEout of rt_reduced:", k_fold_mse(cars_reduced, cars, 392, "tree"))
```

```
##  392-fold CV MSEout of lm_full: 11.29344
##  392-fold CV MSEout of lm_reduced: 11.38004
##  392-fold CV MSEout of lm_poly2_full: 8.610385
##  392-fold CV MSEout of lm_poly2_reduced: 8.787013
##  392-fold CV MSEout of lm_poly6_reduced: 9.177932
##  392-fold CV MSEout of rt_full: 12.76979
##  392-fold CV MSEout of rt_reduced: 13.14515
```

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!)

```
set.seed(NULL) # Unset seed
for(i in 1:5) {
  data = cars[sample(1:nrow(cars)),]
  cat("MSEout of iteration", i, ":", k_fold_mse(cars_full_poly2, data, 392, "lm"), "\n")
}
```

```
## MSEout of iteration 1 : 8.610385
## MSEout of iteration 2 : 8.610385
## MSEout of iteration 3 : 8.610385
## MSEout of iteration 4 : 8.610385
## MSEout of iteration 5 : 8.610385
```

Yes, they will be the exact same. This is because when the number of folds is the same as the number of rows (k=392), all of the iterations will eventually be getting the same exact combination of test sets in the 392 folds (each with a size of 1, meaning that every row will be used at least once for the test data on the 392 folds).

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?)

```
cat(" MSEin of Full Model =", mse_in(lm_full, cars), "\n",
    "MSEin of Reduced Model =", mse_in(lm_reduced, cars), "\n",
    "MSEout of Full Model =", k_fold_mse(cars_full, cars, 392, "lm"), "\n",
    "MSEout of Reduced Model =", k_fold_mse(cars_reduced, cars, 392, "lm"), "\n")
```

```
##  MSEin of Full Model = 10.68212
##  MSEin of Reduced Model = 10.97164
##  MSEout of Full Model = 11.29344
##  MSEout of Reduced Model = 11.38004
```

No, it doesn't. The MSE values, both MSEin and MSEout, of the full models are lower than the reduced counterparts, indicating that the multicollinearity in the full models doesn't seem to hurt the models' fit error and/or prediction error.

Analysts are so scared of multicollinearity because it makes it hard to estimate the individual effects of predictor variables accurately. It also makes it hard to interpret the coefficients, increases standard errors, reduces model stability, and complicates variable selection and model building.

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?)

```
cat(" MSEin of 2nd Order Polynomial Model =", mse_in(lm_poly2_reduced, cars), "\n",
    "MSEin of 6th Order Polynomial Model =", mse_in(lm_poly6_reduced, cars), "\n",
    "MSEout of 2nd Order Polynomial Model =", k_fold_mse(cars_reduced_poly2, cars, 392, "lm"), "\n",
    "MSEout of 6th Order Polynomial Model =", k_fold_mse(cars_reduced_poly6, cars, 392, "lm"), "\n")
```

```
##  MSEin of 2nd Order Polynomial Model = 8.364546
##  MSEin of 6th Order Polynomial Model = 8.254377
##  MSEout of 2nd Order Polynomial Model = 8.787013
##  MSEout of 6th Order Polynomial Model = 9.177932
```

Yes, it does. We can see that the in-sample MSE of the 6th order polynomial model is lower than the in-sample MSE of the 2nd order polynomial model. This means that A higher order model will be able to fit the training data better. But, if we use the models to predict the test data, we can see that the out-of-sample MSE of the 6th order polynomial model is higher than the out-of-sample MSE of the 2nd order polynomial model.

This implies that adding more complex terms and using more complex models can improve the fitting power of the model to the training dataset, but it may also decrease the model's prediction power on the testing dataset. If our model is too complex, it may overfit the data, whereas if our model is too simple, it may underfit the data. So, we need to find the perfect balance between them.