BACS HW16 - 109006234

Credit: 109090035, 109006278

June 4th 2023

Setup

```
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)

set.seed(27935752)
cars <- cars[sample(1:nrow(cars)),]
```

```
cars_full <- mpg ~ cylinders + displacement +
            horsepower + weight + acceleration +
            model_year + factor(origin)
lm_full <- lm(cars_full, data=cars)

cars_reduced <- mpg ~ weight + acceleration + model_year + factor(origin)
lm_reduced <- lm(cars_reduced, data=cars)

cars_full_poly2 <- mpg ~ poly(cylinders, 2) + poly(displacement, 2) +
            poly(horsepower, 2) + poly(weight, 2) + poly(acceleration, 2) +
            model_year + factor(origin)
lm_poly2_full <- lm(cars_full_poly2, data=cars)

cars_reduced_poly2 <- mpg ~ poly(weight, 2) + poly(acceleration,2) +
            model_year + factor(origin)
lm_poly2_reduced <- lm(cars_reduced_poly2, data=cars)

cars_reduced_poly6 <- mpg ~ poly(weight, 6) + poly(acceleration,6) +
            model_year + factor(origin)
lm_poly6_reduced <- lm(cars_reduced_poly6, data=cars)

rt_full <- rpart(mpg ~ cylinders + displacement +
            horsepower + weight + acceleration +
            model_year + factor(origin), data=cars)

rt_reduced <- rpart(mpg ~ weight + acceleration +
            model_year + factor(origin), data=cars)
```

Problem 1

```
mse_in <- function(model, dataset) {
  prediction <- predict(model, newdata=dataset)
  res <- dataset$mpg - prediction
  mse <- mean(res^2)
  return(mse)
}
```

```
#Assuming that calculating MSE using the original dataset
mse_cars_full <- mse_in(lm_full, cars)
mse_cars_reduced <- mse_in(lm_reduced, cars)
mse_cars_poly2 <- mse_in(lm_poly2_full, cars)
mse_cars_reduced_poly2 <- mse_in(lm_poly2_reduced, cars)
mse_cars_reduced_poly6 <- mse_in(lm_poly6_reduced, cars)
mse_cars_rt_full <- mse_in(rt_full, cars)
mse_cars_rt_reduced <- mse_in(rt_reduced, cars)
```

```
##                       mse_in
## cars_full          10.682122
## cars_reduced       10.971643
## cars_poly2          7.919030
## cars_reduced_poly2  8.364546
## cars_reduced_poly6  8.254377
## rt_full             9.155146
## rt_reduced          9.501344
```

Problem 2

(a) Split the data into 70:30 for training:test

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

(b) Retrain the lm_reduced model on just the training dataset. Show the coefficients of the trained model.

```
trained_model <- lm(mpg ~ weight + acceleration +
            model_year + factor(origin), data=train_set)
trained_model$coefficient
```

```
##     (Intercept)          weight    acceleration      model_year factor(origin)2
##   -19.386552584    -0.005828658     0.065048251     0.765849216     1.614507859
## factor(origin)3
##     2.101306397
```

(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?

```
mpg_actual_train <- train_set$mpg
mpg_actual_test <- test_set$mpg

mpg_predicted_train <- predict(trained_model, train_set)
mpg_predicted_test <- predict(trained_model, test_set)

pred_err_train <- mpg_actual_train - mpg_predicted_train
pred_err_test <- mpg_actual_test - mpg_predicted_test

mse_is <- mean((mpg_predicted_train - mpg_actual_train)^2)
mse_is
```

```
## [1] 10.93126
```

```
mse_oos <- mean((mpg_predicted_test - mpg_actual_test)^2)
mse_oos
```

```
## [1] 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

```
result_dataframe <- cbind(mpg_actual_test, mpg_predicted_test, pred_err_test)
head(result_dataframe)
```

```
##     mpg_actual_test mpg_predicted_test pred_err_test
## 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
```

Problem 3

(a) Write a function that performs k-fold cross-validation

```
#For Linear Models
k_fold_mse_lm = function(model, dataset, k=10) {
  fold_pred_errors = sapply(1:k, function(i) {
    fold_i_pe(i, k, dataset, model)
  })
  pred_errors = unlist(fold_pred_errors)
  mean(pred_errors^2)
```

```
}
fold_i_pe = function(i, k, dataset, model) {
  folds = cut(1:nrow(dataset), k, labels = FALSE)
  test_indices = which(folds == i)
  test_set = dataset[test_indices, ]
  train_set = dataset[-test_indices, ]
  trained_model = lm(model, data = train_set)
  test_set[, 1] - predict(trained_model, test_set)
}
```

```
#Regression Tree
k_fold_mse_rt = function(model, dataset, k=10) {
  fold_pred_errors = sapply(1:k, function(i) {
    fold_i_pe_rt(i, k, dataset, model)
  })
  pred_errors = unlist(fold_pred_errors)
  mean(pred_errors^2)
}
fold_i_pe_rt = function(i, k, dataset, model) {
  folds = cut(1:nrow(dataset), k, labels = FALSE)
  test_indices = which(folds == i)
  test_set = dataset[test_indices, ]
  train_set = dataset[-test_indices, ]
  trained_model = rpart(model, data = train_set)
  test_set[, 1] - predict(trained_model, test_set)
}
```

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

```
rt_full_formula <- mpg ~ cylinders + displacement +
          horsepower + weight + acceleration +
          model_year + factor(origin)
rt_reduced_formula <- mpg ~ weight + acceleration +
          model_year + factor(origin)

kfold_cars_full <- k_fold_mse_lm(lm_full, cars, 10)
kfold_cars_reduced <- k_fold_mse_lm(lm_reduced, cars, 10)
kfold_cars_poly2 <- k_fold_mse_lm(lm_poly2_full, cars, 10)
kfold_cars_reduced_poly2 <- k_fold_mse_lm(lm_poly2_reduced, cars, 10)
kfold_cars_reduced_poly6 <- k_fold_mse_lm(lm_poly6_reduced, cars, 10)
kfold_cars_rt_full <- k_fold_mse_rt(rt_full_formula, cars, 10)
kfold_cars_rt_reduced <- k_fold_mse_rt(rt_reduced_formula, cars, 10)
```

```
##                       mse_out
## cars_full           11.262460
## cars_reduced        11.415855
## cars_poly2           8.599373
## cars_reduced_poly2   8.818607
## cars_reduced_poly6   9.267369
## rt_full             13.342221
## rt_reduced          13.476272
```

(ii) For all the models, which is bigger — the fit error (MSEin) or the prediction error (MSEout)?

The fit error, also referred to as the training error, evaluates the model's ability to match the patterns in the data it was trained on. It compares the model's predictions with the actual values in the training set, quantifying the disparity between them. It reflects how well the model has grasped the underlying relationships and patterns in the training data.
On the contrary, the prediction error assesses the model's performance in handling new, unseen data. It measures the deviation between the model's predictions and the actual values in a separate validation or test dataset. The prediction error gauges the model's proficiency in making accurate predictions on examples that were not included in the training process. Since the fit error is calculated using the same data the model was trained on, it is generally lower than the prediction error.

(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?

When re-estimating the model over and over again using the same dataset and the same cross-validation setup, the 10-fold MSE values may vary. The variability can be attributed to the random splitting of the data into folds during each iteration of the cross-validation process. As we have set seed in the random shuffling process, it will remain stable.

(b) Make sure your k_fold_mse() function can accept as many folds as there are rows

k_fold_mse function can do as many folds as there are rows in the data.

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

```r
#Using a modified k_fold_mse()
k_fold_mse_mod <- function(model, data, k){
  data_shuffle_indices <- sample(1:nrow(data),replace = F)
  data_shuffle <- data[data_shuffle_indices,]
  fold_indices <- cut(1:nrow(data), k, labels = FALSE)
  f <- format(terms(model)) %>% paste(., collapse = " ") %>% as.formula()

  fold_pred_errors <- fold_indices
  for(i in 1:k){
    test_set <- data_shuffle[fold_indices == i,]
    train_set <- data_shuffle[fold_indices != i,]
    k_model <- lm(f,train_set)
    predicted <- predict(k_model, test_set)
    formula <- k_model$model %>% names()
    real_value <- test_set[,colnames(test_set) == formula[1]]
    fold_pred_errors[fold_pred_errors==i] <- (real_value - predicted)
  }
  kfold_392 <- mean(fold_pred_errors^2)

  kfold <- c(nrow(train_set), nrow(test_set), kfold_392)
  names(kfold) <- c("train set", "test set", "kfold 392")
  kfold
}
k_fold_mse_mod(lm_full, cars, 392)
```

```
## train set  test set kfold 392
## 391.00000   1.00000  11.29344
```

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

```
kfold_392_full <- k_fold_mse_lm(lm_full, cars, 392)
kfold_392_reduced <- k_fold_mse_lm(lm_reduced, cars, 392)
kfold_392_poly2 <- k_fold_mse_lm(lm_poly2_full, cars, 392)
kfold_392_reduced_poly2 <- k_fold_mse_lm(lm_poly2_reduced, cars, 392)
kfold_392_reduced_poly6 <- k_fold_mse_lm(lm_poly6_reduced, cars, 392)
kfold_392_rt_full <- k_fold_mse_rt(rt_full_formula, cars, 392)
kfold_392_rt_reduced <- k_fold_mse_rt(rt_reduced_formula, cars, 392)
```

```
##                     mse_out_392
## cars_full             11.293439
## cars_reduced          11.380040
## cars_poly2             8.610385
## cars_reduced_poly2     8.787013
## cars_reduced_poly6     9.177932
## rt_full               12.769791
## rt_reduced            13.145150
```

(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?

In general, if we re-estimate a model using the same dataset and the same hyperparameters, the MSEout value will be exactly the same because only 1 data is used for testing and the rest is used as the training.

(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?

```
##                       mse_in mse_out_392
## cars_full          10.682122   11.293439
## cars_reduced       10.971643   11.380040
## cars_poly2          7.919030    8.610385
## cars_reduced_poly2  8.364546    8.787013
## cars_reduced_poly6  8.254377    9.177932
## rt_full             9.155146   12.769791
## rt_reduced          9.501344   13.145150
```

When predictor variables are highly correlated, it becomes challenging for the model to distinguish the individual effects of each variable accurately. As a result, the model may struggle to generalize well to unseen data, resulting in a higher prediction error (MSEout). Hence, I think multicollinearity do not hurt fit error, but it might hurt prediction error. However, to determine multicollinearity, solely relying on MSEin and MSEout as the main factor is not recommended.

(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?

```
##                     mse_in_compare mse_out_392_compare
## cars_reduced_poly2        8.364546            8.787013
## cars_reduced_poly6        8.254377            9.177932
```

If the fit error (MSEin) of the 6th order polynomial regression is significantly lower than that of the reduced quadratic regression, it suggests that the higher-order terms have improved the model's ability to fit the training data. This indicates that the 6th order polynomial regression captures more complex relationships and patterns present in the data. However, if the prediction error (MSEout) of the 6th order polynomial regression is higher than that of the reduced quadratic regression, it implies that the additional higher-order terms have led to overfitting. The model may have become too complex and captured noise or random variations in the training data, resulting in poor performance when making predictions on unseen data. Hence, I think adding more higher-order terms do not hurt fit error, but it might hurt prediction error.