BACS HW (Week 14)

108020024

due on 06/04 (Sun) Helped by 108020033

Question 1) Earlier, we examined a dataset from a security survey sent to customers of e-commerce websites. However, we only used the eigenvalue > 1 criteria and the screeplot "elbow" rule to find a suitable number of components. Let's perform a parallel analysis as well this week:

```
# 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)
```

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.

```
#lm_full: A full model (cars_full) using linear regression
mean((cars$mpg - fitted(lm(cars_full,data = cars)))^2)
```

```
## [1] 10.68212
```

```r
#lm_reduced: A reduced model (cars_reduced) using linear regression

mean((cars$mpg - fitted(lm(cars_reduced,data = cars)))^2)
```

```
## [1] 10.97164
```

```r
#lm_poly2_full: A full quadratic model (cars_full_poly2) using linear regression
mean((cars$mpg - fitted(lm(cars_full_poly2,data = cars)))^2)
```

```
## [1] 7.91903
```

```r
#lm_poly2_reduced: A reduced quadratic model (cars_reduced_poly2) using linear regression
mean((cars$mpg - fitted(lm(cars_reduced_poly2,data = cars)))^2)
```

```
## [1] 8.364546
```

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

mean((cars$mpg - fitted(lm(cars_reduced_poly6,data = cars)))^2)
```

```
## [1] 8.254377
```

```r
library(rpart)

#rt_full: A full model (cars_full) using a regression tree
mean(residuals(rpart(cars_full,data = cars))^2)
```

```
## [1] 9.155146
```

```r
#rt_reduced: A reduced model (cars_reduced) using a regression tree
mean(residuals(rpart(cars_reduced,data = cars))^2)
```

```
## [1] 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
#make this example reproducible
set.seed(123)

sample <- sample(c(TRUE, FALSE), nrow(cars), replace=TRUE, prob=c(0.7,0.3))

cars_train = cars[sample, ]
cars_test =  cars[!sample, ]
```

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 = cars_train)
summary(trained_model)
```

```
##
## Call:
## lm(formula = cars_reduced, data = cars_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.2586 -2.1998  0.0647  1.7841 11.1172
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)     -2.001e+01  4.822e+00  -4.149 4.46e-05 ***
## weight          -6.007e-03  3.349e-04 -17.935  < 2e-16 ***
## acceleration     6.079e-02  8.192e-02   0.742 0.458672
## model_year       7.873e-01  5.985e-02  13.153  < 2e-16 ***
## factor(origin)2  1.891e+00  6.115e-01   3.093 0.002188 **
## factor(origin)3  2.145e+00  6.142e-01   3.493 0.000557 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.422 on 274 degrees of freedom
## Multiple R-squared:  0.8169, Adjusted R-squared:  0.8136
## F-statistic: 244.6 on 5 and 274 DF,  p-value: < 2.2e-16
```

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?

```
mean(residuals(trained_model)^2)
```

```
## [1] 11.45811
```

The in-sample mean-square fitting error is 11.45811.

```
mean( (predict(trained_model, cars_test) - cars_test$mpg)^2 )
```

```
## [1] 10.00899
```

The out-sample mean-square fitting error is 10.00899

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

```r
actual <- cars_test$mpg
predicted  <- predict(trained_model, cars_test)
difference <- abs(actual - predicted)

df <- data.frame(actual, predicted,difference)

head (df)
```

```
##     actual predicted difference
## 372   29.0  30.35676   1.356760
## 162   16.0  16.75613   0.756128
## 214   13.0  16.19914   3.199144
## 37    19.0  16.99882   2.001184
## 197   24.5  28.17246   3.672455
## 201   18.0  19.63566   1.635662
```

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.

```r
k_fold_mse <- function(dataset, k, model) {

  fold_pred_errors <- sapply(1:k, \(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) {

  dataset_temp <- dataset[sample(1:nrow(dataset)), ]

  folds <- cut(1:nrow(dataset_temp),k, labels = FALSE)
  test_indices <- which(folds == i)
  test_set <- dataset_temp[test_indices, ]
  train_set <- dataset_temp[-test_indices, ]
  train_model <- update(model, data = train_set)
  predictions <- predict(trained_model,test_set)
  test_set$mpg - predictions
}
```

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

```r
#lm_full: A full model (cars_full) using linear regression

k_fold_mse(cars, k = 10 , lm(cars_full,data = cars))
```

```
## [1] 9.638379
```

```r
#lm_reduced: A reduced model (cars_reduced) using linear regression

k_fold_mse(cars, k = 10 , lm(cars_reduced,data = cars))
```

```
## [1] 11.6136
```

```r
#lm_poly2_full: A full quadratic model (cars_full_poly2) using linear regression

k_fold_mse(cars, k = 10 , lm(cars_full_poly2,data = cars))
```

```
## [1] 12.27391
```

```r
#lm_poly2_reduced: A reduced quadratic model (cars_reduced_poly2) using linear regression

k_fold_mse(cars, k = 10 , lm(cars_reduced_poly2,data = cars))
```

```
## [1] 10.16528
```

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

k_fold_mse(cars, k = 10 , lm(cars_reduced_poly6,data = cars))
```

```
## [1] 11.80239
```

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

k_fold_mse(cars, k = 10 , rpart(cars_full,data = cars))
```

```
## [1] 10.25744
```

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

k_fold_mse(cars, k = 10 , rpart(cars_reduced,data = cars))
```

```
## [1] 11.79442
```

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

Prediction error is bigger.

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!)

It varies, since we shuffle the data each time.

```
k_fold_mse(cars, k = 10 , lm(cars_full,data = cars))
```

```
## [1] 10.05849
```

```
k_fold_mse(cars, k = 10 , lm(cars_full,data = cars))
```

```
## [1] 10.75335
```

```
k_fold_mse(cars, k = 10 , lm(cars_full,data = cars))
```

```
## [1] 11.11216
```

```
k_fold_mse(cars, k = 10 , lm(cars_full,data = cars))
```

```
## [1] 12.06255
```

```
k_fold_mse(cars, k = 10 , lm(cars_full,data = cars))
```

```
## [1] 11.94058
```

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?

There will be 391 rows in training dataset and 1 row in test dataset of each iteration of k-fold CV when k=392.

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

```
#lm_full: A full model (cars_full) using linear regression

k_fold_mse(cars, k = 392 , lm(cars_full,data = cars))
```

```
## [1] 11.20311
```

```
#lm_reduced: A reduced model (cars_reduced) using linear regression

k_fold_mse(cars, k = 392 , lm(cars_reduced,data = cars))
```

```
## [1] 9.895179
```

```r
#lm_poly2_full: A full quadratic model (cars_full_poly2) using linear regression

k_fold_mse(cars, k = 392 , lm(cars_full_poly2,data = cars))
```

```
## [1] 10.81291
```

```r
#lm_poly2_reduced: A reduced quadratic model (cars_reduced_poly2) using linear regression

k_fold_mse(cars, k = 392 , lm(cars_reduced_poly2,data = cars))
```

```
## [1] 10.53497
```

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

k_fold_mse(cars, k = 392 , lm(cars_reduced_poly6,data = cars))
```

```
## [1] 11.01732
```

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

k_fold_mse(cars, k = 392 , rpart(cars_full,data = cars))
```

```
## [1] 9.578967
```

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

k_fold_mse(cars, k = 392 , rpart(cars_reduced,data = cars))
```

```
## [1] 10.41198
```

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!)

[1] 11.29344 [1] 11.29344 [1] 11.29344

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?)

Multicollinearity present in the full models seem to not 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?)

Adding more higher-order terms hurt the predictions.