HW16

Paula

Question 1) Let's work with the cars_log model and test some basic prediction. Split the data into train and test sets (70:30) and try to predict log.mpg. for the smaller test set:

```
# 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)

# Shuffle the rows of cars
set.seed(27935752)
cars <- cars[sample(1:nrow(cars)),]

# Create a log transformed dataset also
cars_log <- with(cars, data.frame(log(mpg), log(cylinders), log(displacement),
                                  log(horsepower), log(weight), log(acceleration),
                                  model_year, origin))
# Linear model of mpg over all the variables that don't have multicollinearity
cars_lm <- lm(mpg ~ weight + acceleration + model_year + factor(origin), data=cars)
# Linear model of log mpg over all the log variables that don't have multicollinearity
cars_log_lm <- lm(log.mpg. ~ log.weight. + log.acceleration. + model_year + factor(origin),
                  data=cars_log)
# Linear model of log mpg over all the log variables, including multicollinear terms!
cars_log_full_lm <- lm(log.mpg. ~ log.cylinders. + log.displacement. + log.horsepower. +
                         log.weight. + log.acceleration. + model_year + factor(origin),
                       data=cars_log)
```

a. Retrain the cars_log_lm model on just the training dataset (call the new model: lm_trained); Show the coefficients of the trained model

```
set.seed(27935752)
train_indices <- sample(1:nrow(cars_log), size = 0.70*nrow(cars_log))
train_set <- cars_log[train_indices,]
lm_trained <- lm(log.mpg. ~ log.weight. + log.acceleration. +
                   model_year + factor(origin), data = train_set)
lm_trained
```

```
##
## Call:
## lm(formula = log.mpg. ~ log.weight. + log.acceleration. + model_year +
##     factor(origin), data = train_set)
##
## Coefficients:
##       (Intercept)        log.weight.  log.acceleration.         model_year
##           7.32743           -0.87233            0.08277            0.03244
##   factor(origin)2    factor(origin)3
##           0.05215            0.02768
```

b. Use the lm_trained model to predict the log.mpg. of the test dataset

```r
test_set <- cars_log[-train_indices,]
mpg_predicted <- predict(lm_trained, test_set)
mpg_predicted
```

```
##        3        8       16       18       19       20       25       29
## 3.466664 2.789469 2.907741 3.391088 3.354312 2.956784 3.488651 3.236924
##       37       39       46       48       49       50       52       54
## 2.929451 3.209352 3.386614 3.612186 3.578163 2.456565 2.872826 2.998920
##       59       63       64       65       68       70       75       76
## 2.498748 2.573365 2.932996 3.568585 3.525348 3.233248 2.482109 3.407145
##       82       88       89       90       93       94      114      115
## 2.577797 2.997752 2.686801 3.362369 2.498084 2.932767 2.972080 2.816982
##      116      117      119      125      130      133      135      136
## 2.484505 2.909025 3.341183 3.523058 2.782365 3.479172 3.378933 3.617172
##      137      139      140      142      146      148      163      166
## 3.039455 2.540403 2.634703 2.776261 3.470983 3.000045 3.372427 2.566704
##      168      169      175      176      177      181      186      189
## 2.813438 2.803243 3.116331 3.391460 3.151940 3.392594 2.602973 3.122610
##      193      197      198      202      203      205      207      213
## 2.792055 3.183865 3.223766 3.539862 2.575643 2.917953 2.951859 2.465390
##      223      225      227      228      229      231      241      244
## 3.467948 3.269299 3.193899 2.729333 2.754007 3.003903 3.640224 3.353894
##      245      249      250      253      263      264      266      268
## 3.461850 2.592510 2.684052 2.596727 3.143670 3.071153 3.064387 3.040913
##      273      274      276      277      284      285      286      291
## 3.562472 3.171061 2.810481 2.878261 3.235325 3.258988 3.188372 3.566448
##      294      295      300      301      311      313      314      320
## 3.145054 3.422898 2.751471 2.874997 2.693946 2.553277 3.021472 3.382442
##      321      329      336      338      339      345      346      349
## 3.158310 2.972534 2.457794 3.146178 3.245573 3.218505 2.511524 2.932616
##      350      352      354      355      359      361      370      376
## 2.828110 3.351906 2.880312 3.451603 3.101744 3.264240 2.784263 3.185716
##      381      385      386      388      390      392
## 3.389904 3.231582 3.436298 3.428726 3.352106 3.643497
```

i. What is the in-sample mean-square fitting error (MSEIS) of the trained model?

```r
mse_is <- mean((train_set$log.mpg - fitted(lm_trained))^2)
mse_is <- mean(residuals(lm_trained)^2)
mse_is
```

```
## [1] 0.01249181
```

ii. What is the out-of-sample mean-square prediction error (MSEOOS) of the test dataset?

```r
mpg_actual <- test_set$log.mpg.
mse_oos <- mean((mpg_predicted - mpg_actual)^2)
mse_oos
```

```
## [1] 0.01559438
```

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

```r
head(mpg_actual)
```

```
## [1] 3.673766 2.944439 2.890372 3.258097 3.258097 2.890372
```

```r
head(mpg_predicted)
```

```
##        3        8       16       18       19       20
## 3.466664 2.789469 2.907741 3.391088 3.354312 2.956784
```

```r
pred_err <- mpg_actual - mpg_predicted
head(pred_err)
```

```
##           3           8          16          18          19          20
##  0.20710219  0.15496961 -0.01736939 -0.13299180 -0.09621497 -0.06641226
```

Question 2) Let's see how our three large models described in the setup at the top perform predictively!

a. Report the MSEIS of the cars_lm, cars_log_lm, and cars_log_full_lm; Which model has the best (lowest) mean-square fitting error? Which has the worst?

```r
cars_lm_mse_is <- mean((cars$mpg - fitted(cars_lm))^2)
cars_lm_mse_is  <- mean(residuals(cars_lm)^2)
cars_lm_mse_is
```

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

```r
cars_log_lm_mse_is <- mean((cars_log$log.mpg. - fitted(cars_log_lm))^2)
cars_log_lm_mse_is  <- mean(residuals(cars_log_lm)^2)
cars_log_lm_mse_is
```

```
## [1] 0.01332245
```

```r
cars_log_full_lm_mse_is <- mean((cars_log$log.mpg. - fitted(cars_log_full_lm))^2)
cars_log_full_lm_mse_is  <- mean(residuals(cars_log_full_lm)^2)
cars_log_full_lm_mse_is
```

```
## [1] 0.01246619
```

`cars_log_full_lm has the best, and cars_lm_mse has the worst.`

b. Try writing a function that performs k-fold cross-validation (see class notes and ask in Teams for hints!). Name your function k_fold_mse(dataset, k=10, …) – it should return the MSEOOS of the operation. Your function may must accept a dataset and number of folds (k) but can also have whatever other parameters you wish.

```r
library(magrittr)
k_fold_mse <- function(data, k, model){
  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()
  #to reuse the formula in the model#to reuse the formula in the model
  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)
  }
  return(mean(fold_pred_errors^2))
}
```

i. Use/modify your k-fold cross-validation function to find and report the MSEOOS for cars_lm – recall that this non-transformed data/model has non-linearities

```r
k_fold_mse(cars, 10, cars_lm)
```

```
## [1] 11.48344
```

ii. Use/modify your k-fold cross-validation function to find and report the MSEOOS for cars_log_lm – does it predict better than cars_lm? Was non-linearity harming predictions?

```
k_fold_mse(cars_log, 10, cars_log_lm)
```

```
## [1] 0.01387623
```

```
The number is smaller than cars' mse_oos, so it predict better.    No, it didn't harm the
predictions.
```

iii. Use/modify your k-fold cross-validation function to find and report the MSEOOS for cars_log_lm_full – this model has collinear terms; so does multicollinearity seem to harm the predictions?

```
k_fold_mse(cars_log, 10, cars_log_full_lm)
```

```
## [1] 0.01309214
```

```
The number is slightly smaller tham cars_log's mse_oos, and multicollinearity still not
seem to harm the prediction.
```

c. Check if your k_fold_mse function can do as many folds as there are rows in the data (i.e., k=392). Report the MSEOOS for the cars_log_lm model with k=392.

```
k_fold_mse(cars_log, 392, cars_log_lm)
```

```
## [1] 0.01379209
```