BACS HW16

109090046

2023-05-31

Settng Up

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):

```r
# Load the data and remove missing values
cars <- read.table("D:/下載/auto-data.txt", header=FALSE, na.strings = "?")
names(cars) <- c("mpg", "cylinders", "displacement", "horsepower", "weight", "acceleratio
n",
                 "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.

We will first create a function called `mse_in()` that calculates the Mean Squared Error (MSE) of a model.

```r
mse_in <- function(model) {
    predictions <- predict(model, newdata = cars)
    mse <- mean((cars$mpg - predictions)^2)
    return(mse)
}
```

Then we'll fit the models.

```
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)
```

Apply the `mse_in` function to each of them.

```
models <- list(lm_full, lm_reduced, lm_poly2_full, lm_poly2_reduced, lm_poly6_reduced, rt
_full, rt_reduced)
mse_in_values <- lapply(models, mse_in)
names(mse_in_values) <- c("lm_full", "lm_reduced", "lm_poly2_full", "lm_poly2_reduced", "
lm_poly6_reduced", "rt_full", "rt_reduced")

## The MSEin:

## $lm_full
## [1] 10.68212
##
## $lm_reduced
## [1] 10.97164
##
## $lm_poly2_full
## [1] 7.91903
##
## $lm_poly2_reduced
## [1] 8.364546
##
## $lm_poly6_reduced
## [1] 8.254377
##
## $rt_full
## [1] 9.155146
##
## $rt_reduced
## [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
split <- sample.split(cars, SplitRatio = 0.7)
train_set <- subset(cars, split == TRUE)
test_set <- subset(cars, split == FALSE)
```

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)

summary(trained_model)$coefficients

##                      Estimate    Std. Error     t value     Pr(>|t|)
## (Intercept)     -18.869401069 5.2768251523  -3.5759004 4.223095e-04
## weight           -0.005803188 0.0003402132 -17.0575042 3.255935e-43
## acceleration      0.010745886 0.0835210156   0.1286609 8.977342e-01
## model_year        0.770663204 0.0641754760  12.0086870 2.627929e-26
## factor(origin)2   2.367840573 0.6741229774   3.5124757 5.307043e-04
## factor(origin)3   2.737304914 0.6556196914   4.1751414 4.174273e-05
```

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?

```r
# Predict mpg for the test dataset
predictions <- predict(trained_model, newdata = test_set)

# Compute MSEin of the trained model
mse_in_trained <- mean((train_set$mpg - predict(trained_model, newdata = train_set))^2)

# Compute MSEout for the test dataset
mse_out <- mean((test_set$mpg - predictions)^2)

## MSEin: 10.82649

## MSEout: 11.38366
```

d.

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

```r
df <- data.frame(actual_mpg = test_set$mpg,
                 predicted_mpg = predictions,
                 error = test_set$mpg - predictions)
head(df)

##     actual_mpg predicted_mpg      error
## 372       29.0     29.843866 -0.8438658
## 257       20.5     21.522223 -1.0222227
## 81        22.0     22.891648 -0.8916482
## 158       15.0     13.314626  1.6853737
## 7         14.0      9.906655  4.0933450
## 85        27.0     27.346267 -0.3462665
```

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.

```
k_fold_mse <- function(model_formula, dataset, k = 10) {
    folds <- createFolds(dataset$mpg, k = k)
    mse_values <- vector("numeric", k)
    for(i in 1:k) {
        test_indices <- folds[[i]]
        test_set <- dataset[test_indices, ]
        train_set <- dataset[-test_indices, ]
        model <- lm(model_formula, data = train_set)
        predictions <- predict(model, newdata = test_set)
        mse_values[i] <- mean((test_set$mpg - predictions)^2)
    }
    return(mean(mse_values))
}
```

i.

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

```
mse_out_values <- sapply(list(cars_full, cars_reduced, cars_full_poly2, cars_reduced_poly
2, cars_reduced_poly6, cars_full, cars_reduced), k_fold_mse, dataset = cars)
names(mse_out_values) <- names(mse_in_values)

cat("MSE out:\n\n")

## MSE out:

mse_out_values

##          lm_full       lm_reduced    lm_poly2_full lm_poly2_reduced
##        11.331624        11.363868         8.540615         8.921559
## lm_poly6_reduced          rt_full       rt_reduced
##         9.481453        11.103191        11.299940
```

ii.

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

ANS: For all the models, the prediction error are bigger because prediction error is a more honest estimate of how the model performs on unseen data. It might be due to **overfitting and bias-varience tradeoff**.

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!)

```
for (i in 1:5) {
  print(k_fold_mse(cars_full, cars))
}
```

```
## [1] 11.40027
## [1] 11.21914
## [1] 11.35029
## [1] 11.41872
## [1] 11.26866
```

The values vary after re-estimated over and over again because every time the fold can contain different data.

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?

ANS: There will be only one row in training dataset and remaining rows (391) in the test dataset.

ii.

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

```
k_fold_mse <- function(model_formula, dataset, k = 392) {
    folds <- createFolds(dataset$mpg, k = k)
    mse_values <- vector("numeric", k)
    for(i in 1:k) {
        test_indices <- folds[[i]]
        test_set <- dataset[test_indices, ]
        train_set <- dataset[-test_indices, ]
        model <- lm(model_formula, data = train_set)
        predictions <- predict(model, newdata = test_set)
        mse_values[i] <- mean((test_set$mpg - predictions)^2)
    }
    return(mean(mse_values))
}

mse_out_values <- sapply(list(cars_full, cars_reduced, cars_full_poly2, cars_reduced_poly
2, cars_reduced_poly6, cars_full, cars_reduced), k_fold_mse, dataset = cars)
names(mse_out_values) <- names(mse_in_values)

cat("MSE out:\n\n")
```

```
## MSE out:
```

```
mse_out_values
```

```
##         lm_full       lm_reduced    lm_poly2_full lm_poly2_reduced 
##       11.293439        11.380040         8.610385         8.787013
## lm_poly6_reduced          rt_full       rt_reduced
##          9.177932        11.293439        11.380040
```

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!)

```
for (i in 1:5) {
  print(k_fold_mse(cars_full, cars))
}
```

```
## [1] 11.29344
## [1] 11.29344
## [1] 11.29344
## [1] 11.29344
## [1] 11.29344
```

It doesn't vary because the test dataset are always the same.

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?)*

ANS:

It appears that multicollinearity present in the full models does not seem to significantly hurt their fit error (MSEin). This suggests that the models are able to capture the relationships between the variables present in the training data, resulting in a relatively low fit error.

Analysts are concerned about multicollinearity due to its potential to produce unreliable coefficient estimates, inflated standard errors, overfitting, difficulties in interpretation, and sensitivity to small changes. While the impact of multicollinearity on the fit error (MSEin) may not always be significant, these other issues can undermine the accuracy, stability, and interpretability of the regression model.

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?)*

ANS:

The fit error and prediction errors are reported to be higher for the 6th order polynomial regression compared to the reduced quadratic regression. This suggests that adding more higher-order terms, in this case, the 6th order polynomial terms, did not improve the fit or prediction accuracy.

The higher errors in the 6th order polynomial regression imply that the increased complexity introduced by the additional higher-order terms did not capture the underlying relationships in the data effectively. In other words, the model became overfit to the noise or idiosyncrasies in the training data, resulting in reduced performance on unseen data.