STAT 213: Overfitting and Cross-Validation

Colin Reimer Dawson

Last Revised April 12, 2016

The purpose of this worksheet is two-fold: (1) for you to get a conceptual understanding and a feel for the phenomenon of overfitting, and (2) for you to learn how to do cross-validation in R.

I suggest running this in a script or a Markdown document. First type `set.seed(SOME_NUMBER)` (replacing `SOME_NUMBER` with an integer of your choice) so that you can get the same results every time you re-run or re-Knit your code.

We will create a synthetic dataset where the true relationship between X and Y is a cubic polynomial. We will then see what happens to our in-sample and out-of-sample predictive accuracy as we fit increasingly higher-order polynomials to the data.

First, make sure the mosaic package is loaded.

```
set.seed(1234) ## use your own value for the seed
library("mosaic")
```

Let's define the "population" polynomial, which will be a cubic.

```
beta0 <- 1; beta1 <- -1/2; beta2 <- -1/2; beta3 <- 1/4
## defines a function that we can call to transform xs to y-hats
f <- function(x) { beta0 + beta1 * x + beta2 * x^2 + beta3 * x^3 }
## defines the error standard deviation
sigma.epsilon = 7
```

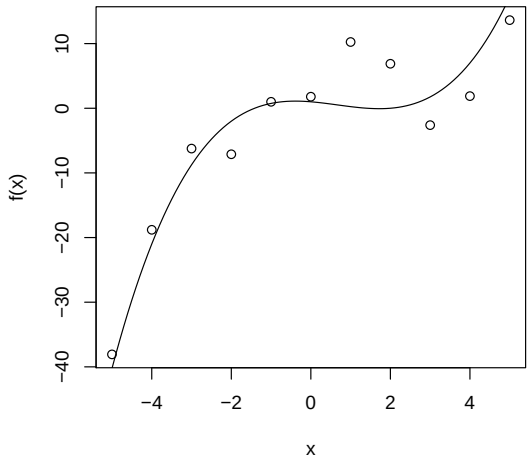
Now, let's generate some data.

```
n <- 11
x <- seq(-5, 5, length = n) ## evenly spaced integers from -5 to 5
f.x <- f(x) ## the y points exactly on the curve
```

```
epsilon <- rnorm(n, mean = 0, sd = sigma.epsilon) ## normally distributed resids.
y <- f.x + epsilon
TheData <- data.frame(x = x, y = y)
```

Let's plot the data with the population curve overlaid (your curve will be the same, but your data will be different, since you are using your own random seed)

```
curve(f(x), from = -5, to = 5, ylim = range(~y, data = TheData))
points(y ~ x, data = TheData)
```



Now, let's fit a series of nine polynomial regression models to this data, ranging from order 1 (linear) to order $n-2$ (we stop short of $n-1$ so that we have at least one error degree of freedom). The easiest thing to do is probably to use the `poly()` function for this. For example

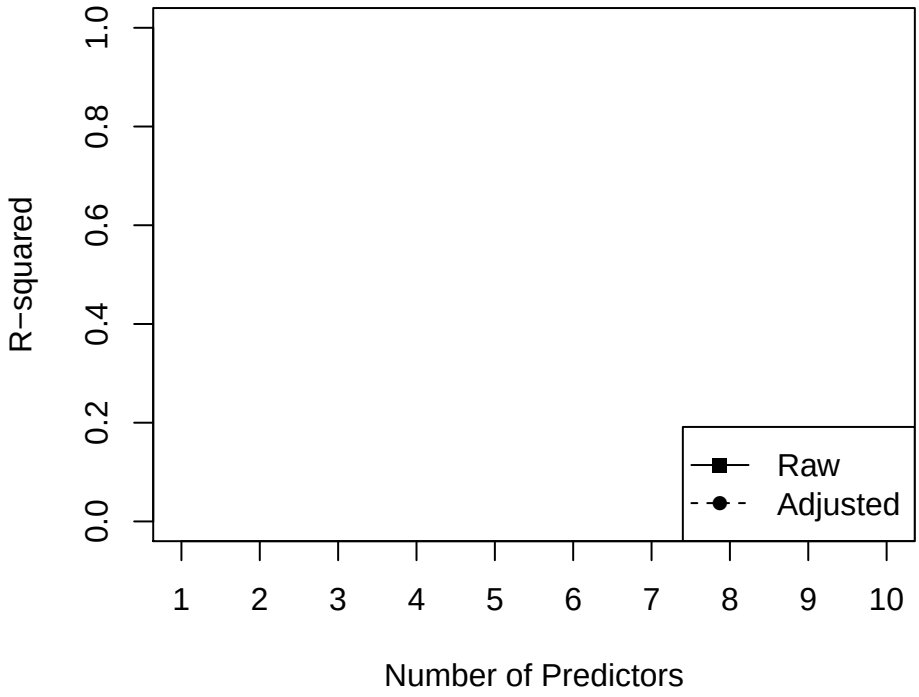
```
model2 <- lm(y ~ poly(x, degree = 2, raw = TRUE), data = TheData)
```

Plot each model over the data.

```
plotModel(model2)
```

Notice that, as we did with our adjusted R^2 activity, for the the first three models, we are adding predictors that are actually related to the response, according to our known population model. For the terms of degree greater than 3, there is no relationship between the predictors we are adding and the response.

For your nine models, find the R^2, and the adjusted R^2, and plot them both below as a function of the number of predictors.



So far we are evaluating the fit of each model on the full dataset; that is, we are using the same data to fit and evaluate the models.

Let's take a different approach: **leave-one-out cross-validation**. The process is the following:

1. For i ranging from 1 to n:
   (a) Fit a model on all but the ith data point.
   (b) Compute the predicted y value on the ith data point: $\hat{y}_i = \hat{f}(x_i)$.
   (c) Calculate the squared difference between the actual y value and the predicted y value for the ith data point.
2. Compute the average squared prediction error across all n models, then take the square root to bring it back to the scale of the data.

We can use the following R code to execute this loop for a particular model form:

```
squared.errors <- numeric(n)  # define a container for the errors
for(i in 1:n)  ## recall we have defined n above
{
    ## We need to estimate new parameters for each held-out data point
    ## the form [-i,] excludes the ith row
    model2_i <- lm(y ~ poly(x, degree = 2, raw = TRUE),
                   data = TheData[-i,])
    y.hat <- predict.lm(model2_i, newdata = TheData[i,])
    ## selects the ith row from the column with label "y"
    y <- TheData[i,"y"]
    squared.errors[i] <- (y - y.hat)^2
}
root.mse <- squared.errors %>% mean() %>% sqrt()
```

As a matter of concise coding, since we are repeating this code over and over, varying only the degree of the polynomial, we can define a function that takes the model formula and the data arguments, and returns the square root of the average of the squared errors:

```
get.leave.one.out.mse <- function(formula, the.data)
{
    ### We will assume there is a variable called 'y'
    n <- nrow(the.data)
    squared.errors <- numeric(n)
    for(i in 1:n)
    {
        model_i <- lm(formula, data = the.data)
        ## all the rest of the code in the loop is the same,
        ## except we replace the specific data with the function name
        y.hat <- predict.lm(model2_i, newdata = the.data[i,])
        ## selects the ith row from the column with label "y"
        y <- the.data[i,"y"]
        squared.errors[i] <- (y - y.hat)^2
    }
    squared.errors %>% mean() %>% sqrt() %>% return()
}
```

We can use the function that we've created by calling it for each model formula that we want to validate. For example:

```
leave.one.out.m2 <-
    get.leave.one.out.mse(
        formula = y ~ poly(x, degree = 2, raw = TRUE),
        the.data = TheData)
```

Compute the leave-one-out cross-validation root mean squared error for each polynomial degree, and add the values to the plot (create a custom axis on the right-hand side to account for the fact that the values are on a different scale). Which model appears to be do the best under each metric (R^2, adj. R^2, leave-one-out RMSE)?