STAT 215 Lab 8: Overfitting and Cross-Validation

Colin Reimer Dawson

Last Revised November 3, 2017

The purpose of this lab 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 use `set.seed()` with an integer of your choice so that you can get the same results every time you re-run or re-Knit your code.

As we did in class yesterday to examine what happens to R^2 as we add predictors, we will again use a synthetic dataset generated from a known mean function. This time we will consider various polynomial models.

The other has just one predictor, but the true relationship between X and the mean of Y is a third-degree (cubic) polynomial, given by the following formula:

$$f(x) = 1 - \frac{1}{2}x - \frac{1}{2}x^2 + \frac{1}{4}x^3$$

The Y values have been generated by adding $\mathcal{N}(0, 7)$ residuals to $f(x)$.

Run the following to run scripts from my website that will generate the data:

```
source("http://colindawson.net/stat215/code/make_fake_mlr_data.R")
source("http://colindawson.net/stat215/code/gen_poly_data.R")
```

The second script will plot the true cubic function, along with the data.

Similar to what we did in class, let's fit a series of ten regression models to this data, this time using polynomial prediction functions, 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, the model for degree 2 is

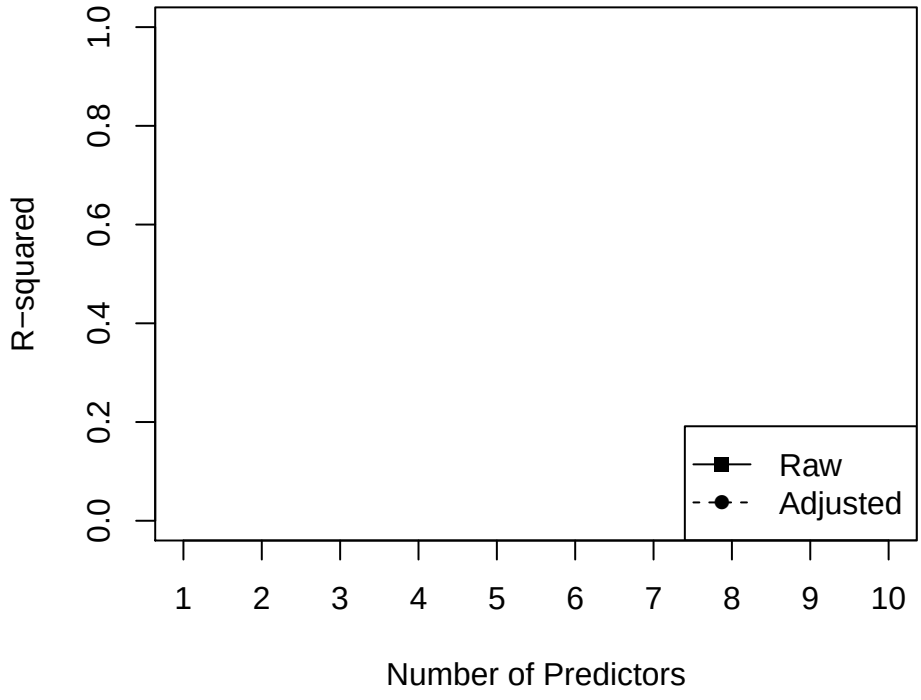
```
model2 <- lm(y ~ poly(x, degree = 2, raw = TRUE), data = FakePolyData)
```

Fit each model (degree 1 to degre n-2) and plot each model over the data, using the form

```
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 involves the following steps:

1. For each data point in the sample (i ranging from 1 to n), we will:
   (a) Fit a model on all *but* the ith data point.
   (b) Compute the predicted y value for 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 "out of sample" 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. You don't need to be able to write code like this, but try to understand what each line does.

```
n <- nrow(FakePolyData)
squared.errors <- numeric(n)  # define a container for the errors
poly.degree <- 2 # we'll use a quadratic for this example

## a for loop tells R to run the code in braces repeatedly,
## starting with i = 1, then i = 2, etc. until i = n on the last iteration
for(i in 1:n)
{
    ## We need to estimate new parameters for each held-out data point
    ## the form [-i,] excludes the ith row (i.e, "case")
    model_i <- lm(y ~ poly(x, degree = poly.degree, raw = TRUE),
                  data = FakePolyData[-i,])
    y.hat_i <- predict.lm(model_i, newdata = FakePolyData[i,])
    ## selects the ith row from the column with label "y"
    ## Note that to do this with other data, we need to change 'y'
    ## to whatever the name of the response variable is
    y_i <- FakePolyData[i,"y"]
    ## Computes the squared prediction error between the actual
    ## and predicted value and puts it in position i in the container
    ## called squared.errors
```

```
    squared.errors[i] <- (y_i - y.hat_i)^2
}
root.mse <- squared.errors %>% mean() %>% sqrt()
```

To save time, I have created a function called `get.leave.one.out.rmse` that will let you do leave-one-out cross-validation for any dataset, response variable, and model formula. It has the following form:

```
get.leave.one.out.rmse(
    formula,             # a model formula of the form Y ~ <stuff>
    data.set,            # the name of the data frame
    response.name        # the name of the response variable (in quotes)
    )
```

It returns the root mean squared error based on the specified model using leave-one-out cross-validation.

To define the function, run:

```
source("http://colindawson.net/stat215/code/define_cv_function.R")
```

We can use the function that we've created by calling it for each model formula that we want to validate. For example, for a quadratic model, we can get the leave-one-out root mean squared error by calling:

```
leave.one.out.m2 <-
    get.leave.one.out.rmse(
        formula = y ~ poly(x, degree = 2, raw = TRUE),
        data.set = FakePolyData,
        response.name = "y")
```

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 do the best under each metric (R^2, adj. R^2, leave-one-out RMSE)?