Module1_Part1

Andrew Estes

3/16/2022

```r
knitr::opts_chunk$set(message=FALSE, error=FALSE, warning=FALSE)
```

```r
library(pROC)
library(tidyverse)
library(caret)
```

Intro

Sometimes it's nice to use simulated data, rather than real data when evaluating predictive models because you can control the "true" parameters than define the data and then assess how well the model captures those true parameters. Consider the fuction below, which creates a data set with a long-term trend, smaller-scale oscillation, and random noise:

```r
sim.data <- function(seed, h=3, s=3){
  set.seed(seed)
  x <- runif(200, min=0, max=6)
  y <- 8 + (x-4)^2 + 4*sin(h*x) + rnorm(200, mean=0, sd=s)
  return(data.frame(x=x, y=y))
}
```

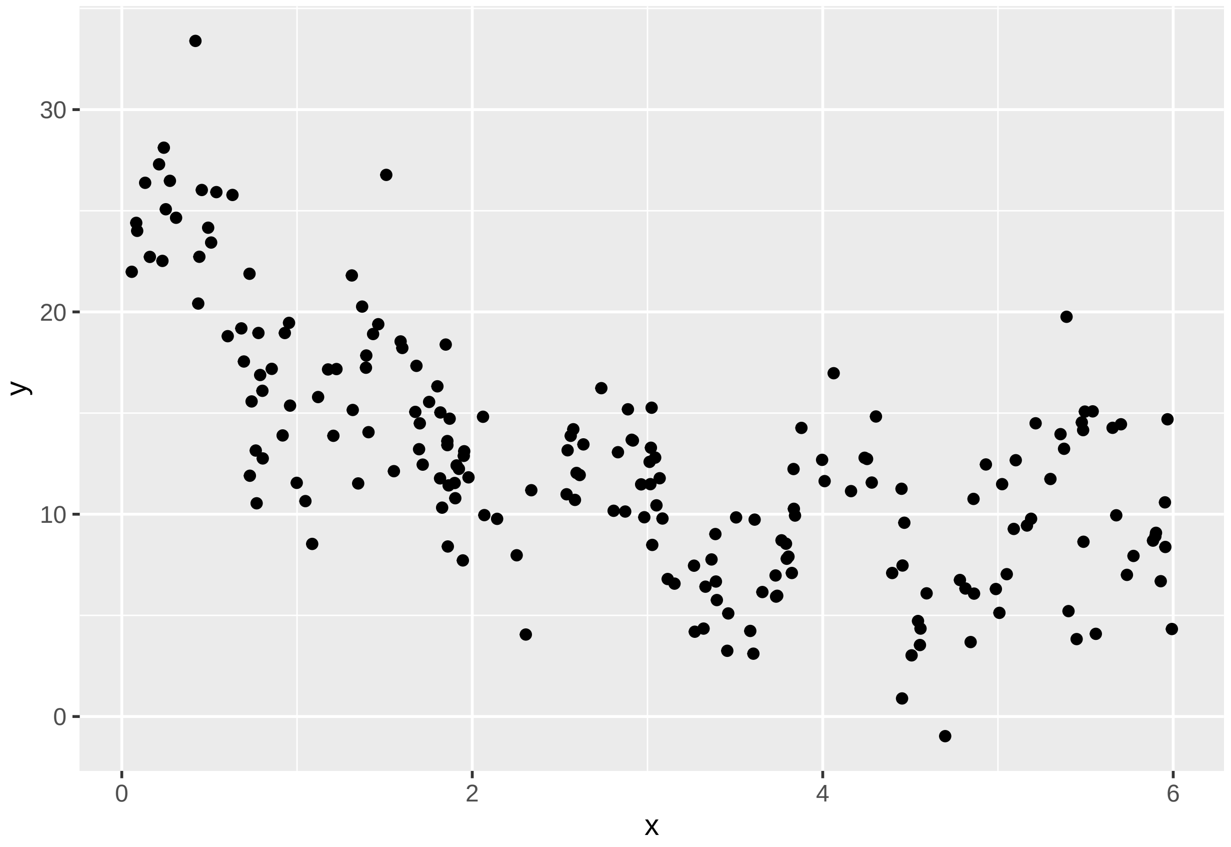
The function returns a data frame with an x and a y column. In this homework set, you'll practice function definition and loop construction by doing the following to train a LOESS regression model to fit the data set produced by this function:

Part 1

Write code that sets values for seed, h and s, creates a data frame by running the sim.data command, and plots the results. Once you've got the code working, choose values of seed, h and s that give what you feel is an interesting graph to fit with LOESS regression and use those in your code and output (you don't have to show all your tried, just the final choice for those parameters).

```r
#creating dataframe from function
df <- sim.data(1234, 5, 3)

#plotting output
ggplot(df, aes(x, y)) +
  geom_point()
```



Part 2

Write code that uses loops and functions to implement "repeated cross-validation" function by hand. Specifically, write code that defines constants k and m and that repeats k-fold cross-validation m times on your data set.

The RMSE calculation from each fold of each run show be stored in a numeric vector that has a length of k*m, and at the end of the run, the average RMSE should be output, along with standard error, calculated by for formula sd(rmse.vec)/sqrt(length(rmse.vec)) The function can be written specifically for this assignment and data set (as was done in the coding lab), but should take inputs for k, m and span at least.

```
rmse.df <- function(method=loess, degree=1, span=0.50, k=10, m=5, ...){
  set.seed(1234)
  rmse.vec <- numeric(k*m)
  for(i in 1:m) {
    for(j in 1:k) {
      folds <- rep(1:k, each=ceiling(nrow(df)/k))
      folds <- sample(folds, nrow(df))

      fit <- method(y ~ x, data=df[folds != j, ], degree=1, span=span)

      rmse.vec[j+(k*i)-k] <- RMSE(df$y[folds==j],
                                  predict(fit, newdata=df[folds==j, ]),
                                  na.rm=TRUE)
    }

    return(mean(rmse.vec))
}
}

rmse.df(loess, span=.5)
```

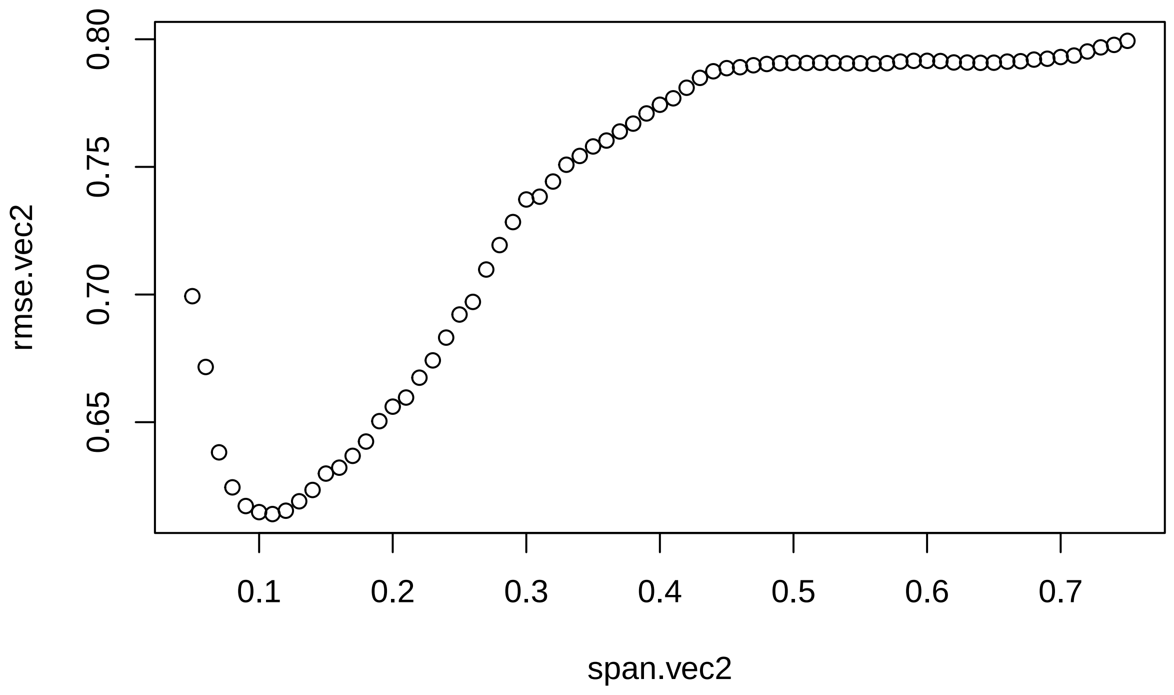
```
## [1] 0.7907758
```

Part 3

Write code to do model tuning for the span parameter "by hand." Your code should call the function you defined in the last part, and store the cross-validated RMSE for each span value in a vector. It would likely be good to try values of span from 0.05 to 0.50, but depending on the curve you choose, the best span might be outside that range, so make sure to look at your results carefully. Plot a graph of CV-RMSE vs. span value and clearly state which value of span minimizes RMSE.

```r
span.vec2 <- seq(0.05, 0.75, by=0.01)
rmse.vec2 <- map_dbl(span.vec2, ~rmse.df(method=loess, degree=1, k=10, m=5, span=.x))


#plotting span vs RMSE
plot(x=span.vec2, y=rmse.vec2)
```



```r
#finding best span vector
span.best <- span.vec2[which.min(rmse.vec2)]
span.best
```

```
## [1] 0.11
```

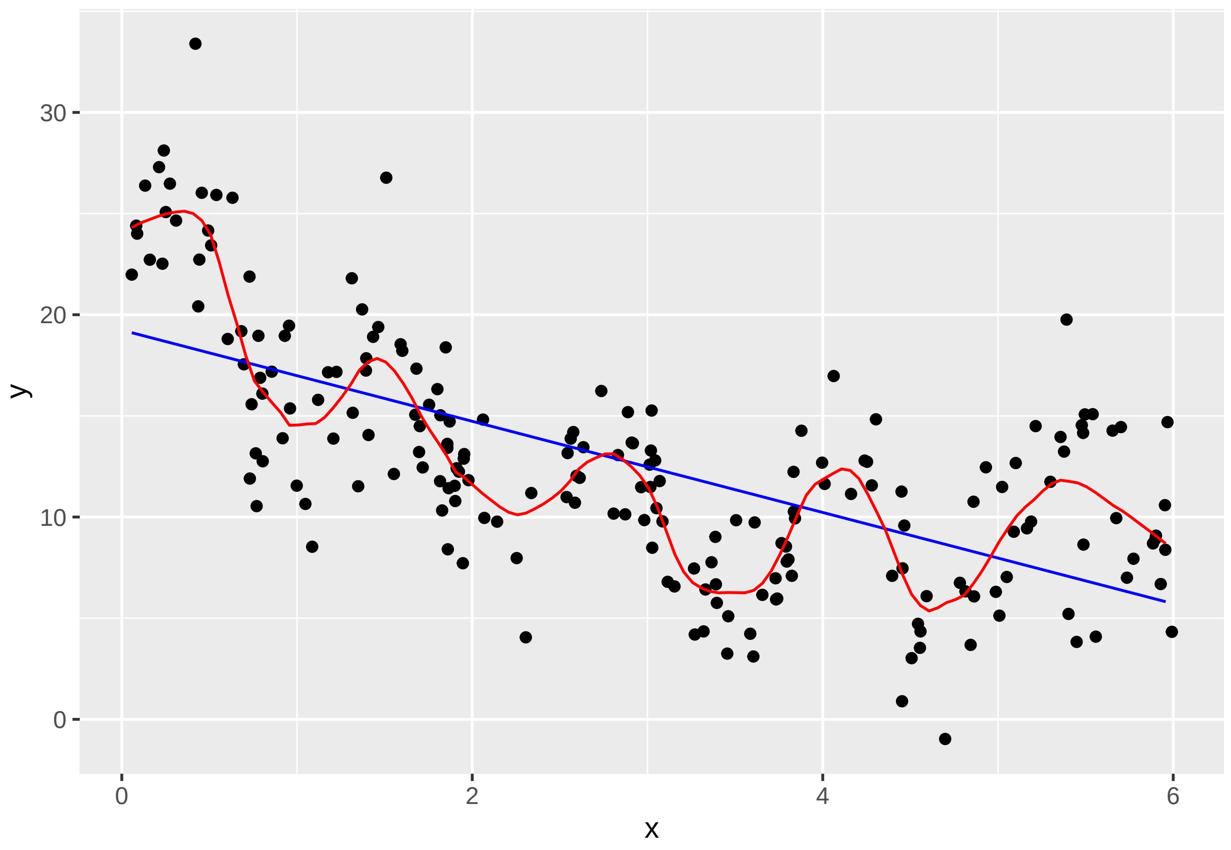
0.14 is the best span, but anything ranged 0.13 to 0.19 would work similarly well.

```r
df.plot <- data.frame(
  x = seq(min(df$x, na.rm=TRUE), max(df$x, na.rm=TRUE), by=0.05)
)


fit.lm <- lm(y ~ x, data=df)
fit.loess <- loess(y ~ x, data=df, degree=1, span=span.best)


df.plot$lm.y <- predict(fit.lm, newdata=df.plot)
df.plot$loess.y <- predict(fit.loess, newdata=df.plot)


ggplot(df) +
  geom_point(aes(x=x, y=y)) +
  geom_line(data=df.plot, aes(x=x, y=lm.y), color="blue") +
  geom_line(data=df.plot, aes(x=x, y=loess.y), color="red")
```



I think the loess line is close to matching the optimal curve. It seems to be affected by the extreme points.