In-Class Computing Task 16

Math 253: Statistical Computing & Machine Learning

Building your own spline fitter

In this programming activity, you're going to build your own cubic spline fitter.

The knots

Recall that in specifying a cubic spline, the *only* parameter to be selected is the number of knots, k. It's usual to place the knots at evenly spaced points inside the range of the input variable x. A sensible definition of "evenly spaced" is that each of the intervals between knots contain roughly the same number of x values. In other words, the knots will be placed at evenly spaced quantiles of x. For instance, if $k = 2$, the knots are placed at the $\frac{1}{3}$ and $\frac{2}{3}$ quantiles.

Write a function `my_knots(x, k)` which takes the x data and the number of knots as arguments. The evenly spaced quantiles will be $\frac{1}{k+1}, \frac{2}{k+1}, \ldots, \frac{k}{k+1}$. Use the `quantile()` function to calculate the values at which these quantiles occur.

The basis set

For fitting the spline, you are going to construct a model matrix based on x. The model matrix will consist of a set of vectors, $v_1, v_2, \ldots$ each of the same length as x. Vector v_i is computed by applying some function $f_i()$ to x. Here are the functions $f_i()$ for a cubic spline:

- The constant function $f_1(x) = 1$.
- The linear function, $f_2(x) = x$.
- The quadratic function $f_3(x) = x^2$.
- The cubic function $f_4(x) = x^3$
- Another k functions, each of which is $(x - \xi_j)^3_+$, where ξ_j is the jth knot.

Recall that $(x - \xi)^3_+$ means that the function has value 0 when $x < \xi$ and value $(x - \xi)^3$ otherwise.

Constructing the first four vectors in the model matrix is easy. Call the model matrix `MM`.

```
MM <- cbind(1, x, x^2, x^3)
```

To add in the k vectors containing $(x - \xi_j)^3_+$, use a loop over each of the knots, adding the new vector to the model matrix like this:

```
MM <- cbind(MM, new_vector)
```

Package up the computations to create the model matrix into a function `spline_model_matrix(x, knot_locations)`.

Finding the best linear combination

Once you have your `knots()` and `spline_model_matrix()` functions, finding the best least squares fit is straightforward, using `lm()` like this:

```
spline_coefs <- coef(lm(y ~ MM))
```

Create a function `fit_spline()` that does the calculations. The `fit_spline()` function will have a formula interface. It will return a list of items relevant to calculating the value of the spline at any point x. Here's the function:

```
fit_spline <- function(formula, k=2, data=parent.frame()) {
  y <- eval(formula[[2]], envir=data)
  x <- eval(formula[[3]], envir=data)
  knot_locations <- my_knots(x, k)
  MM <- spline_model_matrix(x, knot_locations)
  mod <- lm(y ~ MM - 1)
  res <- list(coef = coef(mod), knots = knot_locations, cov = vcov(mod))
  class(res) <- "my_spline"

  return(res)
}
```

I've written this function for you because it contains the statements to make use of a formula as well as something new, use of the `class()` function. More about that later. The component `cov` gives the covariance matrix of the model coefficients. Those will be used later to find standard errors on the spline values, confidence intervals, and prediction intervals.

The predict function

Write a corresponding predict function called `predict.my_spline()`.[1]

[1] This name allows you to refer to the function simply with `predict()`. R, seeing that the first argument is an object of class `my_spline`, will automatically call `predict.my_spline()`.

```
predict.my_spline <-
  function(mod, newx, level = 0.95,
           intervals=c("none", "confidence", "prediction")) {
  intervals <- match.arg(intervals)
  MM <- spline_model_matrix(newx, mod$knots)
  vals <- MM %*% mod$coef
  se <- sqrt(rowSums(MM %*% mod$cov * MM))
  if (intervals == "none") return(vals)
  else return(NULL) # for future use
}
```

Again, I've given you a template. You'll add on to it later.

Used together, `fit_spline()` and `predict()` (that is, `predict.my_spline()`) enable you construct a spline-function model $f(\hat{x})$

Try out your function on the `Wage` data from the ISLR package, finding a smooth model of wage versus age.

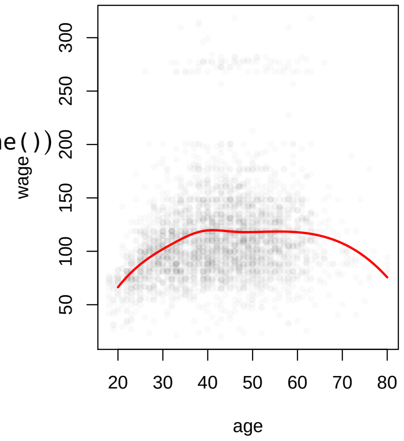

Figure 1: Figure 1. A cubic spline with 4 knots modeling wage versus age.

Confidence intervals

Modify `predict.my_spline()` so that if the argument `intervals` is set to `"confidence"`, you will return a data frame with three columns: `vals`, `upper` and `lower`. Following the usual procedure for constructing 95% confidence intervals, `upper` will be `vals` plus 2 times the standard error `se`, and similarly with `lower`.[2]

Revise your plot (as in Figure 1) to show the upper and lower confidence intervals.

[2] Strictly speaking, the multiplier for a confidence level of `level` is `qt((1-level)/2, df=dfr)`, where `dfr` is the degrees of freedom of the residual.

Prediction intervals

Modify `predict.my_spline()` again so that if the argument `intervals` is set to `"prediction"`, you will return the three-column data frame, but giving the 95% prediction intervals. These will be based on `se` and the standard error of the residuals `rse` from the fit. The prediction standard error is `sqrt(se^2 + rse^2)`. You'll have to go back to `fit_spline()` and arrange for it to include the `rse`. This is the sum of square residuals divided by the degrees of freedom in the residual.[3]

[3] The residual degrees of freedom is the number of points n in the data used for fitting minus the number of columns in the model matrix.

How many knots?

1. Examine the spline fit to wage vs age as you change the number of knots to get a sense for how the spline's smoothness and prediction intervals vary.
2. Divide `Wage` into training and testing sets. Find the number of knots that minimizes the mean square error on the testing set.