LECTURE 5, PART II: REGRESSION SPLINES AND SMOOTHING SPLINES

Text references: Chapter 8 in Shalizi

So how do you use splines for regression? Before we discuss regression splines and smoothing splines, let's first review polynomial regression. [Some text adapted from Larry Wasserman]

Review: Polynomial Regression and Penalized Sum of Squares

Consider polynomial regression

$$Y = \sum_{j=0}^{p} \beta_j x^j + \epsilon.$$

or

$$\widehat{r}(x) = \sum_{j=0}^{p} \widehat{\beta}_j x^j.$$

Polynomial regression is a **basis function** approach. Instead of fitting a linear model in x, we fit a standard linear model with predictors $b_0(x) = 1$, $b_1(x) = x$, $b_2(x) = x^2$, ..., $b_p(x) = x^p$, where the basis functions $b_0(\cdot)$, $b_1(\cdot), \ldots, b_p(\cdot)$ are polynomial functions. Hence, we can use *least squares* to estimate the unknown regression functions in the model above, and use all of the inference tools available for linear models.

Specifically, the design matrix for polynomial regression is given by

$$\mathbb{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^p \\ 1 & x_2 & x_2^2 & \dots & x_2^p \\ & & & \dots & \\ 1 & x_n & x_n^2 & \dots & x_n^p \end{bmatrix}.$$

Least squares minimizes

$$(\vec{Y} - \mathbb{X}\beta)^t(\vec{Y} - \mathbb{X}\beta),$$

which implies $\widehat{\beta} = (\mathbb{X}^t\mathbb{X})^{-1}\mathbb{X}^t\vec{Y}$. If we introduce a *ridge regression penalty* (see posted 401 notes) and aim to minimize

$$(\vec{Y} - \mathbb{X}\beta)^t(\vec{Y} - \mathbb{X}\beta) + \lambda\beta^t I\beta,$$

then $\tilde{\beta} = (\mathbb{X}^t\mathbb{X} + \lambda I)^{-1}\mathbb{X}^t\vec{Y}$. Spline regression follows a similar pattern, except that we replace $\mathbb{X}$ with the B-spline basis matrix $\mathbb{B}$ (see below).

Regression Splines

The most commonly used basis for splines is the *cubic B-spline*; see R for details. Figure 1 shows the cubic B-spline basis using nine equally spaced knots on (0,1). B-spline basis functions have compact support which makes it possible to speed up calculations (a slower alternative is to use a truncated power basis).

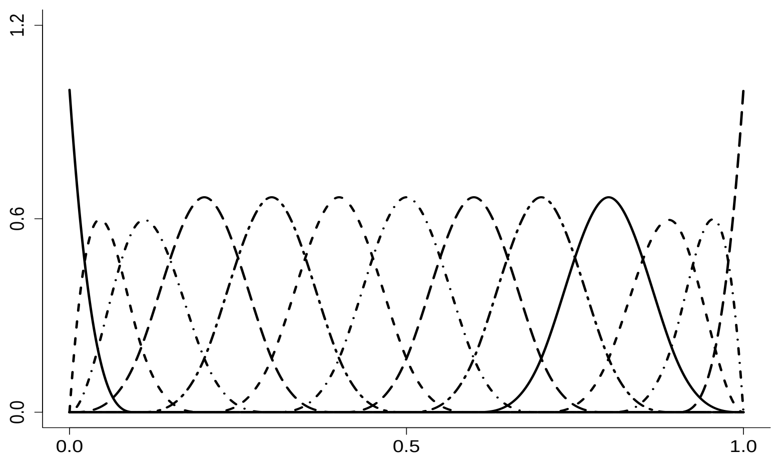

Figure 1: Cubic B-spline basis using nine equally spaced knots on (0,1).

A cubic spline with m knots can be modeled as:

where $B_j(x), j = 1, \ldots, N$ are the basis vectors for the B-spline and $N = m + 4$. (Note the basis is determined by the observed values of x_j.) We follow the pattern of polynomial regression, but replace $\mathbb{X}$ with $\mathbb{B}$, where

$$\mathbb{B} = \begin{bmatrix} B_1(x_1) & B_2(x_1) & \ldots & B_N(x_1) \\ B_1(x_2) & B_2(x_2) & \ldots & B_N(x_2) \\ & & \ldots & \\ B_1(x_n) & B_2(x_n) & \ldots & B_N(x_n) \end{bmatrix}$$

To find our regression spline estimator we only need to find the coefficients $\widehat{\beta} = (\widehat{\beta}_1, \ldots, \widehat{\beta}_N)^T$. For this, we can do *ordinary linear regression*. What are the optimal (i.e. least squares) coefficients?

Regression splines are *linear smoothers* (check this!). What are the fitted values for this estimator?

How do we control the amount of smoothing when using regression splines?

Remark 1: Natural Splines. There is a variant of the B-spline basis for natural splines (which are linear beyond the boundary knots). Because of the additional boundary conditions, a natural cubic spline with m knots only needs $N = m$ basis vectors.

Remark 2: Comparison to Polynomial Regression. "Regression splines often give superior results to polynomial regression. This is because unlike polynomials, which must use a high degree (exponent in the highest monomial term, e.g. X^{15}) to produce flexible fits, splines introduce flexibility by increasing the number of knots but keeping the degree fixed [to, for example, k=3]. Generally, this approach produces more stable estimates. Splines also allow us to place more knots, and hence flexibility, over regions where the function f seems to be changing rapidly, and fewer knots where f appears more stable." [Ref: ISL]

Exercise 1: As an example, consider the code in the file `bsplinempg.R` on the Canvas website. In that code, we use the first training set of (engine-displacement, mpg) pairs from Homework #1. The plot makes it look like a knot at engine-displacement= 200 might be useful. We fit linear, quadratic, and cubic splines as well as a single cubic polynomial. The cubic spline and single cubic polynomial have very similar fits. Can you run the code and see if one of them seems to fit better than the other?

Choosing the (number and) *locations* of the knots $t_1, \ldots, t_m$ can however be tricky. This is the beauty behind smoothing splines — with them, we don't have to choose knots!

Smoothing Splines

Smoothing splines perform a **regularized regression over the natural spline basis**, placing knots at **all** points $x_1, \dots x_n$. Smoothing splines circumvent the problem of knot selection (as they just use the inputs as knots), and simultaneously, they control for overfitting by shrinking the coefficients of the estimated function (in its basis expansion).

We will focus on natural cubic splines, defined in the previous lecture (Part I). These splines arise naturally in the penalized regression framework as the following theorem shows.

Theorem 1 *The function $\widehat{r}(x)$ that minimizes the* **penalized sum of squares**

$$\mathcal{L}(\lambda) = \sum_{i=1}^{n} (Y_i - r(X_i))^2 + \lambda \, J(r) \tag{1}$$

where

$$J(r) = \int (r''(x))^2 dx \tag{2}$$

is a natural cubic spline with knots at the data points. The estimator $\widehat{r}$ is called a **smoothing spline**.

The theorem above however does not give an explicit form for $\widehat{r}$ but we can work out its form by expanding the regression function $r(x)$ in the basis, calculating the 2nd derivatives, and then rewriting the minimization....

Hence, we write

$$r(x) = \sum_{j=1}^{N} \beta_j B_j(x) \tag{3}$$

where $B_j(x)$, $j = 1, \ldots, N$, are the basis vectors for the natural B-spline, $N = n$.

By expanding r in the basis, and calculating the 2nd derivatives, we can now rewrite the minimization as follows:

where $\mathbb{B}$ is the $N \times N$ matrix with (i, j) element $B_{ij} = B_j(X_i)$ and $\Omega_{jk} = \int B_j''(x) B_k''(x) dx$. The exact form of the penalty matrix Ω is actually not so important. What you should pay attention to is that there is an extra term $\lambda \beta^T \Omega \beta$ in the loss function compared to the previous expression for regression splines; this is called a *regularization* term, and it has the effect of shrinking the components of the solution $\hat{\beta}$ towards zero. The parameter $\lambda \geq 0$ is a tuning parameter, often called the smoothing parameter, and the higher the value of λ, the more shrinkage.

Following the pattern we saw for ridge regression, the value of β that minimizes the penalized sum of squares is

Smoothing splines are another example of *linear smoothers* $\widehat{\vec{Y}} = L\vec{Y}$:

If we had done ordinary linear regression of Y with basis $\mathbb{B}$, the "hat matrix" would be $L = \mathbb{B}(\mathbb{B}^T\mathbb{B})^{-1}\mathbb{B}^T$ and the fitted values would interpolate the observed data. Again: The effect of the term $\lambda\beta^T\Omega\beta$ in the penalty is to shrink the regression coefficients towards a subspace, which results in a smoother fit. We choose the smoothing parameter λ by minimizing some form of cross-validation score.

On the Usage of Smooth Splines

In R, the call `out = smooth.spline(x,y)` will fit a smoothing spline of y on x, and internally choose the tuning parameter value for the smoothing spline using a an approximate form of cross-validation. However, you should check to see exactly what R does, and what the function outputs! Alternatively, you could pick a value of λ, or pick several values of λ and compare them via cross-validation. The function outputs the unique values of the predictor sorted from smallest to largest, and the corresponding fitted values in `out$x` and `out$y` respectively if the output object is `out` as above. Then `lines(out$x,out$y)` will add the fitted curve to a plot of the data. However, if there are large gaps between `x` values, the curve won't be as smooth as the spline itself. Alternatively, you could use the function `predict` to compute the fitted spline on a dense regular grid and plot that. For example, if `x0` contains equally-spaced values that cover the range of the predictor (possibly extending beyond at both ends,) then

```
plot(x0,predict(out,x=x0)$y)
```

will add the fitted spline to an existing plot.

You can choose λ via cross-validation by looping through many values of `lambda` and then using the λ corresponding to the smallest cross-validation error. The problem is that λ can be anything from 0 on up, and it is difficult to know where to start looking. We will talk more about that later. (A natural place to look is near the internally-chosen smoothing parameter mentioned earlier.)

As an example, we take the Old Faithful eruption data. As described in Shalizi Section 6.4.1, this classic data set contains the time between 299 eruptions of the Old Faithful geyser in Yellowstone, and the length of the subsequent eruptions; these variables are called `waiting` and `duration`. The data are plotted in Figure 2 with `waiting` versus `duration`, along with a linear and smoothing spline fit using the internally chosen smoothing parameter, which happened to be $\lambda = 0.0517$.

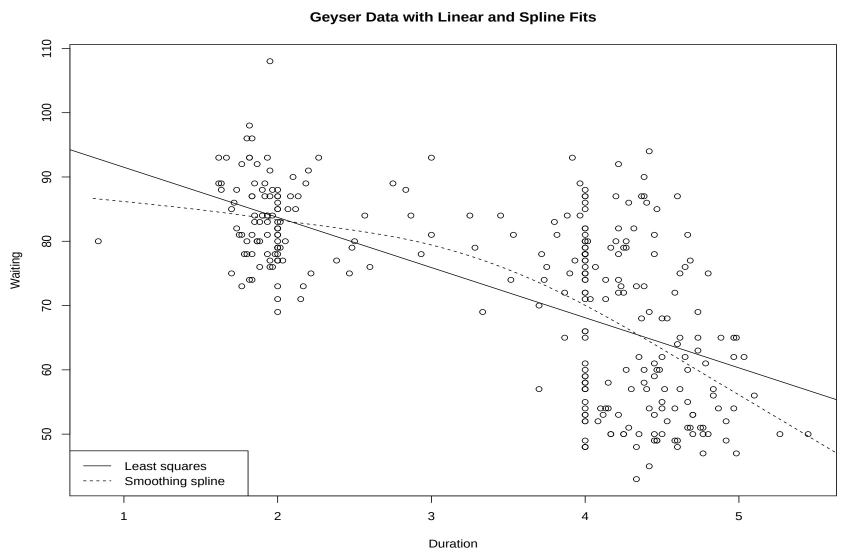

Figure 2: geyser data with two fitted regression functions.

Now that we have a guess at λ, we could choose some values in the vicinity of that guess and perform 5-fold cross-validation to choose a final λ.

Bootstrap for Spline Models

Given that the precise formulas for the predictors (the B-spline functions) are not easy to work with, computing uncertainty measures (confidence intervals for $\mathbb{E}[\widehat{r}(x)]$, tests, prediction errors, etc.) will be a challenge. It should not be surprising to learn that the bootstrap can be helpful, as it was in other difficult-to-analyze cases.

How can we use the bootstrap to get a confidence interval for $\mathbb{E}[\widehat{r}(x)]$ for one or more values of x?

Once again, we have (at least) three ways to bootstrap a regression problem. The three we learned earlier are:

Regarding resampling cases, one usually resamples the entire fitting procedure. That means that even the choice of tuning parameter is repeated for each bootstrap sample. If the tuning parameter is chosen by a lengthy cross-validation procedure, that procedure needs to be repeated for each bootstrap sample. This can add a large amount of uncertainty to the results produced, but that is not necessarily bad: it basically tells you that there is a great deal of uncertainty about whatever inference you are trying to make.

(Note however that in HW #7, Problem 2, we are looking at a special case of bootstrap via resampling cases, where we use the same tuning parameter λ in each bootstrap sample.)

Once you have chosen which of the four bootstraps to do, you then compute the confidence intervals in the appropriate way. For methods that do not specify a specific distribution for the noise terms, the pivotal confidence intervals may be most appropriate when the residuals appear skewed.

Here is a plot of the residuals from the spline fit:

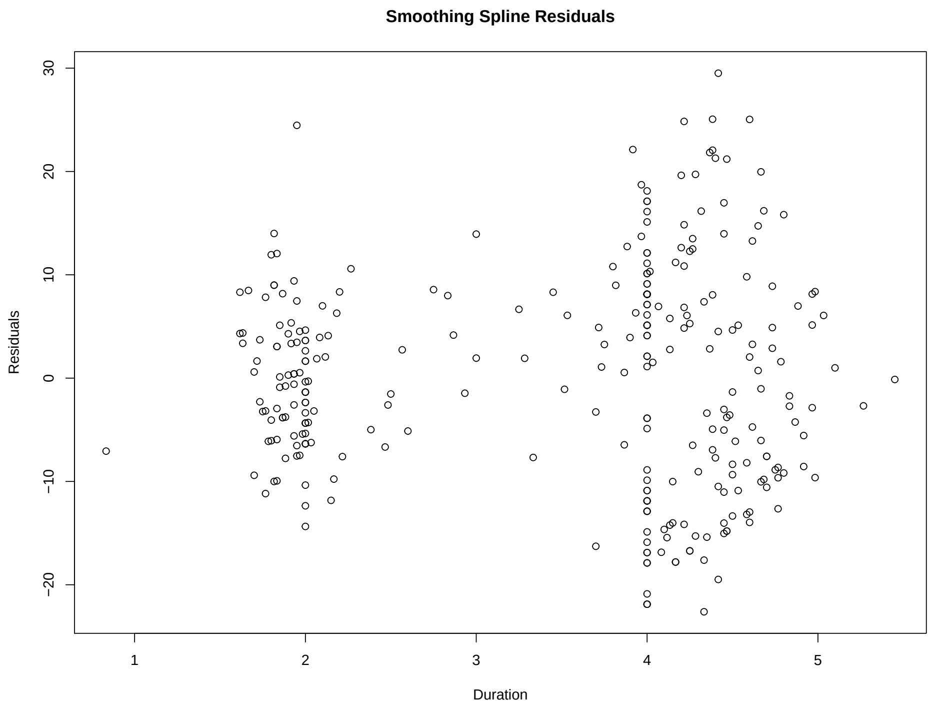

Figure 3: Residuals of smoothing spline fit to geyser data.

What, if anything, does this plot suggest?

R Demo 5.1 (Smoothing Splines and Bootstrap)

We start by fitting a smoothing spline to the Geyser data (using the default value of `lambda`) and compare to a linear fit.

```
> library(MASS)  # The data are supplied as part of this library
> data(geyser)
> y=geyser$waiting  # Set the names to match earlier discussion
> x=geyser$duration
> out=smooth.spline(y~x)  # This syntax actually works
> out$lambda
[1] 0.05174383
> plot(x,residuals(out),xlab="Duration",ylab="Residuals",
+ main="Smoothing Spline Residuals") # Figure 3
> plot(x,y,xlab="Duration",ylab="Waiting",
+ main="Geyser Data with Linear and Spline Fits") # Begin Figure 2
> lmgeyser=lm(waiting~duration,data=geyser)  # Linear fit
> abline(lmgeyser$coef) # Add linear fit to Figure 2
> x0=0.5+5*c(0:100)/100 # Equally-spaced points for spline fit
> out$pred=predict(out,x=x0)$y # Predictions at x0
> lines(x0,out$pred,lty=2) # Add spline fit to Figure 2
> legend("bottomleft",lty=c(1:2),legend=c("Least squares",
+ "Smoothing spline")) # Add legend to Figure 2
```

For the spline fit, we compute a pivotal bootstrap confidence interval for $\mathbb{E}[\widehat{r}(x)]$ by resampling cases. For this analysis, we will fix `lambda` at the value that appears in all of the above, similar to the spline-bootstrap homework problem. The uncertainty that we are quantifying is for that value alone. It might make more sense to bootstrap the entire process in which `lambda` was also chosen during the spline fit. We will do that case after.

```
> B=1000
> n=length(x)
> for(b in 1:B){
+   cases=sample(n,replace=T)
+   outb=smooth.spline(y[cases]~x[cases],lambda=lambda0)
+   thecurves=rbind(thecurves,predict(outb,x=x0)$y)
+ }
> quants=apply(thecurves,2,quantile,prob=c(0.025,0.975))
> lines(x0,2*out$pred-quants[1,],lty=3,col="blue")
> lines(x0,2*out$pred-quants[2,],lty=3,col="blue")
```

Figure 4 contains the previous plot with the pivotal bootstrap confidence intervals added. To redo the analysis with λ changing for each bootstrap sample, we merely replace the line defining `outb` with one that removes the specification of `lambda`.

```
+   outb=smooth.spline(y[cases]~x[cases])
```

One can see how much uncertainty is added by bootstrapping the choice of λ. Although not shown here, a similar increase in uncertainty occurs if one replaces the internal choice of λ with 5-fold cross-validation. Small changes in the data produce large changes in the fit. A similar result appears in Section 6.4.2 of Shalizi, where splines are replaced by kernel regressions.

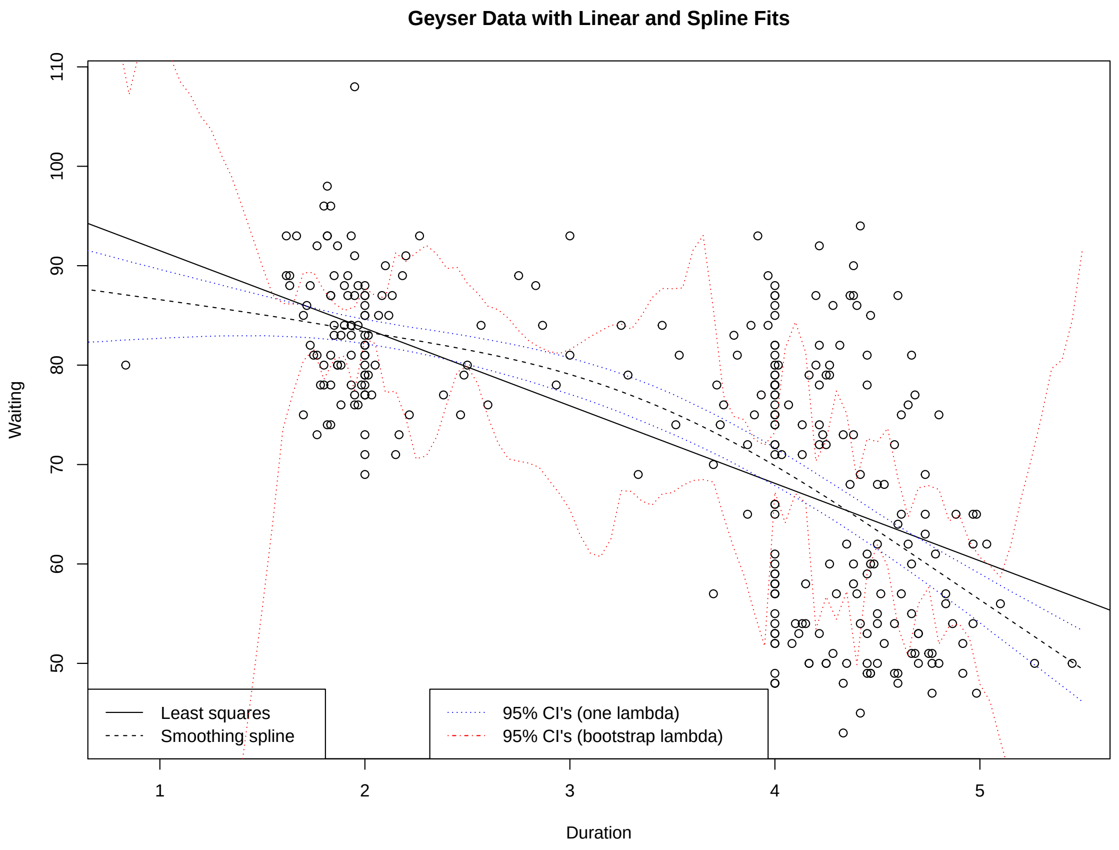

Figure 4: Bootstrap confidence intervals added to Figure 2. Both fixed $\lambda = 0.0517$ and bootstrapping λ are included.

Can you think of a reason why the spread between the intervals based on bootstrapping the choice of λ are so much wider in particular places than those based on a fixed λ with this data set?

If the one point on the far left is not included in the bootstrap sample, there is virtually nothing constraining the curve for `Duration`< 1.6, where a straight line will be fit because of the fit being a natural spline. On the other hand if the point on the far left appears more than once in the bootstrap sample, there will be more leverage driving the curve close to that point. Similar things happen at the far right.
In the middle. where points are more sparse, small changes to how many and which of these points are included will have an impact on how wiggly the curve gets in the middle region.