Smoothing Splines

A more principled approach is to fit spline models by minimizing residual sum of squares plus a penalty — similiar to ridge regression. If we expect the function to be fairly smooth, a suitable penalty will relate to how big its derivatives are.

Here's one such penalized criterion for how good a function f is:

$$\sum_{i=1}^{N} \Big[y_i - f(x_i)\Big]^2 \;+\; \lambda \int \Big[f''(x)\Big]^2 dx$$

The first term is just the usual RSS. The second term is a penalty that is large if the second derivative of f is large.

The constant λ controls how big the penalty is. If $\lambda = 0$, the first term will force the function to pass through the data points. As $\lambda \to \infty$, the second term will force the function to be a straight line.

Since f here can be *any* function with second derivatives, it seems very hard to find the f that minimizes this criterion.

Remarkably, it turns out that the solution is always a natural cubic spline, with knots at the distinct data points.

Fitting Smoothing Splines (Part I)

If we have N training cases, with (univariate) inputs $x_1, \ldots, x_N$, we first sort these points, and eliminate duplicates, to give the knot locations $\xi_1, \ldots, \xi_K$, where $K \leq N$ (less if there are duplicates).

We then find a set of basis functions for the natural cubic splines with these K knot locations. Call them $N_m(x)$ for $m = 1, \ldots, K$. These basis functions are piecewise cubic polynomials.

We need to find estimates $\hat{\theta}_m$ for $m = 1, \ldots, K$ of the coefficients for these basis functions. We do this by minimizing RSS plus the penalty.

Fitting Smoothing Splines (Part II)

Given coefficients $\theta_1, \ldots, \theta_K$, the function is defined by

$$f_\theta(x) \;=\; \sum_{m=1}^{K} \theta_m N_m(x)$$

and its second derivative is therefore

$$f_\theta''(x) \;=\; \sum_{m=1}^{K} \theta_m N_m''(x)$$

$N_m(x)$ is piecewise cubic, so $N_m''(x)$ is piecewise linear (and zero outside (ξ_1, ξ_K)).

The penalty is

$$\lambda \int \Big[f''(x) \Big]^2 dx \;=\; \lambda \sum_{j=1}^{K} \sum_{k=1}^{K} \theta_j \theta_k \int N_j''(x) N_k''(x)\, dx$$

Given the knot locations, we can compute the integrals above, since the N_j'' are piecewise polynomials.

Fitting Smoothing Splines (Part III)

We can compute a penalty matrix, $\mathbf{P}$, with $\mathbf{P}_{jk} = \int N_j''(x) N_k''(x)\, dx$. The penalty can then be written as $\theta^T \mathbf{P} \theta$.

We can also compute a matrix, $\mathbf{N}$, with $\mathbf{N}_{ij} = N_j(x_i)$ — the values of the basis functions at the training inputs. The penalized RSS can then be written as

$$(\mathbf{y} - \mathbf{N}\theta)^T (\mathbf{y} - \mathbf{N}\theta) \; + \; \lambda \theta^T \mathbf{P} \theta$$

which is a quadratic function of θ.

The $\hat{\theta}$ that minimizes this is found analogously to ridge regression, with the solution being

$$\hat{\theta} \;\; = \;\; (\mathbf{N}^T \mathbf{N} + \lambda \mathbf{P})^{-1} \mathbf{N}^T \mathbf{y}$$

The prediction for a test case with inputs x is then

$$\hat{y} \;\; = \;\; \sum_{m=1}^{K} \hat{\theta}_m N_m(x)$$

Choosing the Smoothing Parameter

With smoothing splines, we use as many knots as we have data points, so we don't have to choose K (or the knot locations).

But we do have to choose λ — a small value will result in the function passing almost through the data points (probably bad), while a large a value will result in a nearly straight line fit.

One approach is to choose λ by cross validation.

There is a computational trick for doing leave-one-out cross validation without recomputing everthing with each training case left out (see p. 137 in the text).

Unlike for choosing K by cross validation, there's no reason why λ would systematically be chosen differently for large datasets than for small datasets.

Additive Spline Models

When we have more than one input variable, we can fit an additive model in which the contribution of each input is modeled by a spline:

$$f(x) \;=\; \beta_0 + \sum_{j=1}^{p} f_j(x_j)$$

$$f_j(x) \;=\; \sum_{m=1}^{M_j} \theta_{m,j} h_{m,j}(x_j)$$

Here $h_{m,j}$ for $m = 1, \ldots, M_j$ are basis functions for the spline used for input j.

These additive models can handle non-linear relationships of y with any x_j, but they **don't** handle interactions between inputs.

A technical point: In the full set of basis functions for a univariate spline, there will be one (or a combination of several) that has the value 1 everywhere. This allows for the equivalent of an intercept term. In an additive model, we want only one intercept. So we use one fewer basis function for each spline, and add a single intercept, β_0, explicitly. For example, if we used natural cubic splines, we would have only $N - 1$ basis functions for each.

Avoiding Overfitting in Additive Spline Models

As for univariate splines. . .

Either we need to somehow select K_j for $j = 1, \ldots, p$, perhaps by cross validation (and also knot locations), and then fit by least squares,

Or we use natural cubic splines for each input, selecting a suitable λ_j for each input variable (again perhaps by cross validation), and then minimize the penalized RSS.

If we adjust the scales of the inputs to be comparable, we might use the same λ for all splines.

Fitting Additive Spline Models

We can fit an additive spline model by just combining the basis functions for all the splines, and estimating all their coefficients simultaneously.

From our original input variables, we create a matrix $\mathbf{N}$ that looks like

$$\mathbf{N} \;=\; \left[\begin{array}{c|c|c} 1 & & \\ 1 & & \\ 1 & \quad\mathbf{N}_1\quad & \quad\mathbf{N}_2\quad \\ 1 & & \\ 1 & & \end{array}\right]$$

The first column contains 1s for the intercept, the next set of columns contains values of basis functions in training cases for the spline for the first input, etc.

We can also write the penalty matrices for each spline as one big block-diagonal matrix, $\mathbf{P}$, so that the total penalty is $\beta^T\mathbf{P}\beta$, where β contains β_0 and the coefficients for all the splines. We can then compute $\hat{\beta} = (N^T N + \lambda\mathbf{P})^{-1}\, N^T\mathbf{y}$. (However, for cubic splines, one can compute this more efficiently in other ways.)