Degrees of freedom for splines

- degrees of freedom for cubic splines, 1 covariate
  - \( K \) knots, \( K + 1 \) intervals, \( 4(K + 1) \) parameters
  - \( 3K \) restrictions
  - leaves \( K + 4 \) parameters, omit constant term, leaves \( K + 3 \)

- degrees of freedom for natural splines
  - linear on \((−∞, \xi_1]\) and \([\xi_K, ∞)\): 4 pars
  - cubic in interior intervals \(4(K − 1)\)
  - \(3K\) restrictions, leaves \(K\) parameters, but
  - 2 knots added at \(x(1), x(n)\): \(K + 2\)
  - omit constant term get \(K + 1\)

- what does omit constant term mean? Example

\[
S_0(x) = a_0 + b_0 x + c_0 x^2 + d_0 x^3, \quad 0 \leq x \leq 1
\]
\[
S_1(x) = a_1 + b_1 (x − 1) + c_1 (x − 1)^2 + d_1 (x − 1)^3, \quad 1 \leq x \leq 2
\]

- e.g. \(a_0 = b_0 = c_0 = d_0 = 1\); fixes \(a_1 = 4, b_1 = 6, c_1 = 8\)
- or, force \(a_0 = 0\), add it in later
Degrees of freedom for splines

- degrees of freedom for smoothing splines

\[
\min_f \sum \{ y_i - f(x_i) \}^2 + \lambda \int \{ f''(t) \}^2 dt
\]

- solution is natural cubic splines with knots at unique \( x_i \)

\[ f(x) = \sum_{j=1}^{N} N_j(x) \theta_j, \text{ say} \]

- \( \min_\theta \sum \{ y_i - \sum_j N_j(x_i) \theta_j \}^2 + \lambda \sum_{jk} \theta_j \theta_k \Omega_{jk} \)

- \( \min_\theta (y - N\theta)^T (y - N\theta) + \lambda \theta^T \Omega_N \theta \)

- \( \Omega_{jk} = \int N''_j(t)N''_k(t) dt \)

- just like ridge regression: \( \hat{\theta} = (N^T N + \lambda \Omega_N)^{-1} N^T y \)

\[ \hat{f} = N\hat{\theta} = n(N^T N + \lambda \Omega_N)^{-1} N^T y = S_\lambda y, \text{ say} \]

- degrees of freedom defined to be trace \( S_\lambda \) by analogy

- Same formula works for regression splines (actually easier) cf (5.15)
Suppose we have $X_1$, $X_2$, and $E(y \mid X) = f(X_1, X_2)$

one solution is to combine separate spline bases for $X_1$ and $X_2$

e.g. additively: $f(X_1, X_2) = f_1(X_1) + f_2(X_2)$ (this is what was done for heart data)

doesn’t permit interactions

alternative is to use all possible cross products: called tensor products

$$f(X_1, X_2) = \sum_{j=1}^{M_1} \sum_{k=1}^{M_2} \theta_{jk} h_{1j}(X_1) h_{2k}(X_2)$$

analogous to forming quadratic functions in regression using, e.g., $x_1^2$, $x_1 x_2$, $x_2^2$
alternative to derive smoothing splines in two dimensions:

$$\min \sum_{i=1}^{N} (y_i - f(x_i))^2 + \lambda J(|f|)$$

$$J(|f|) = \int \int (\partial_1^2 f + \partial_2^2 f + 2\partial_{12} f)^2 dxdy$$

as in univariate case, solution exists in a spline basis similar to natural splines

(5.39): $$f(x) = \beta_0 + \beta^T x + \sum_{j=1}^{N} \alpha_j h_j(x)$$

$$h_j(x) = \eta(||x - x_j||), \quad \eta(z) = z^2 \log z$$

called radial basis functions: take this form because of symmetry of penalty

note uses N knots; reduced in implementation by regularization
Kernel methods for regression: univariate

- model: \( E(Y \mid x) = f(x) \) (“smooth”)
- data: \( y_i = f(x_i) + \epsilon_i \)
- running mean smoother: \( \hat{f}(x_0) = \text{ave}(y_i \mid x_i \in N_k(x_0)) \)
- \( N_k(x_0) \) set of \( k \) “nearest neighbours”: \( k \) smallest values of \( |x_i - x_0| \)
- smoother estimate using kernel weighted average

\[
\hat{f}(x_0) = \frac{\sum_{i=1}^{N} K_\lambda(x_0, y_i)y_i}{\sum_{i=1}^{N} K_\lambda(x_0, x_i)}
\]

[Figure 6.1]
- kernel

\[
K_\lambda(x_0, x) = D \left( \frac{|x - x_0|}{\lambda} \right) \text{ or } D \left( \frac{|x - x_0|}{h_\lambda(x_0)} \right)
\]
kernel methods for regression

- $\lambda$ determines the width of the neighbourhood, hence smoothness
- increasing $\lambda$ gives smoother function (higher bias, lower variance)
- metric window width ($h_\lambda(x_0) = \lambda$) - constant bias, variance $\propto 1$/local density
- nearest neighbour window width ($h_\lambda(x_0)$ depends on $x_0$) - constant variance, bias $\propto 1$/local density
- Choice of kernel:

\[
D(t) = \begin{cases}
\frac{3}{4}(1 - t^2), & |t| \leq 1 \\
0 & \text{Epanichakov}
\end{cases}
\]

\[
= \begin{cases}
(1 - |t|^3)^3, & |t| \leq 1 \\
0 & \text{tri – cube}
\end{cases}
\]

\[
= \phi(t) = \frac{1}{\sqrt{2\pi}} \exp(-t^2/2) & \text{Gaussian}
\]
R or Splus:

ksmooth(x, y, kernel=c("box", "normal"), bandwidth=0.5, range.x=range(x), n.points=max(100, length(x)), x.points)

loess(formula)

more later

> eps<-rnorm(100, 0, 1/3)
> x<-runif(100)
> sin4x <- function(x){sin(4*x)}
> y<-sin4(x)+eps
> plot(sin4,0,1,type="l",ylim=c(-1.0,1.5),xlim=c(0,1))
> points(x,y)
> lines(ksmooth(x,y,"box",bandwidth=.2),col="blue")
> lines(ksmooth(x,y,"normal",bandwidth=.2),col="green")
> plot(sin4,0,1,type="l",ylim=c(-1.0,1.5),xlim=c(0,1))
> lines(ksmooth(x,y,"normal",bandwidth=0.4),col="blue")
> lines(ksmooth(x,y,"normal",bandwidth=0.6),col="red")

(Figure 6.1)
Local linear regression

- replace weighted average of $x_i$'s with weighted linear (or polynomial) regression: better endpoint behaviour

$$
\min_{\alpha(x_0), \beta(x_0)} \sum K_\lambda(x_0, x_i) \{ y_i - \alpha(x_0) - \beta(x_0) x_i \}^2
$$

$$
\hat{f}(x_0) = (1, x_0)(X^T W(x_0) X)^{-1} X^T W(x_0) y
$$

$$
X = \begin{pmatrix}
1 & x_1 \\
1 & x_2 \\
\vdots & \vdots \\
1 & x_n
\end{pmatrix} = B
$$

- $W(x_0) = \text{diag} \, K_\lambda(x_0, x_i)$
Recall weighted least squares:

$$\min_{\beta} \sum w_i(y_i - \beta_0 - \beta_1 x_i)^2 \text{ or } \min_{\beta} (y - X\beta)^T W(y - X\beta)$$

$$\hat{\beta} = (X^T WX)^{-1} X^T Wy$$

can combine the least squares weights with the kernel weights; see Figure 6.4 and pp. 169, 170.

can also do local quadratic regression (and higher) but increases bias at endpoints

for extrapolation book recommends local linear fits; for good fits in middle local quadratic

In R there are several smoothers: ksmooth and loess are built in

The first uses kernel smoothing, the second uses local linear regression (robustified)
scatter.smooth fits a loess curve to a scatter plot

loess takes a family argument: family = gaussian gives weighted least squares using $K_\lambda$ as weights and family=symmetric gives a robust version using Tukey’s biweight

supsmu implements “Friedman’s super smoother”: a running lines smoother with elaborate adaptive choice of bandwidth

Library KernSmooth has locpoly for local polynomial fits, and by setting degree = 0 gives a kernel smooth
kernel methods for regression

> lol <- loess(y˜x, degree=1, span=0.75)
> attributes(lol)
$n$
$n$ names
[names]
[1] "fitted" "residuals" "enp" "s" "one.delta"
[7] "two.delta" "trace.hat" "divisor" "pars" "kd"
[13] "terms" "xnames" "x" "y" "weights"
$class$
 class
[1] "loess"
$plot$(sin4,0,1,type="l",ylim=c(-1.0,1.5),xlim=c(0,1))
> points(x,lol$fitted,pch=".",col="red")
> plot(x,lol$fitted,pch=".",col="red",ylim=c(-1.0,1.5),xlim=c(0,1))
> lines(ksmooth(x,y,"normal",bandwidth=0.4),col="blue")
> plot(x,lol$fitted,pch=".",col="red",ylim=c(-1.0,1.5),xlim=c(0,1))
> lo2<-loess(y˜x, degree=1, span=0.4)
> points(x,lo2$fitted,pch=".",col="green")
> points(x,loess(y˜x,degree=2,span=0.4)$fitted,pch=".",col="purple")
Notes

- $\hat{f} = S_\lambda y$ and $\text{df} = \text{trace}(S_\lambda)$, as in smoothing splines
- $X$ can have up to 4 numerical predictors
- while possible to fit these models in $R^p$, (see §6.3, 6.4), doesn’t seem so useful
- §6.4 describes ways to impose some structure to get a more interpretable model
- can use the same idea for likelihood functions and maximum likelihood estimates:

$$\max_\beta \sum \ell(\beta; y_i)$$

replaced by

$$\max_\beta \sum K_\lambda(x_0, x_i) \ell(\beta; y_i)$$

called local likelihood and described in §6.5
Kernel methods for classification

- model: \( X \sim f(\cdot) \)
- training data \((x_1, \ldots, x_N)\)
- \( \hat{f}(x_0) = \frac{\#\{x_i \in n_\lambda(x_0)\}}{N_\lambda} \) (a histogram)
- \( \hat{f}(x_0) = \frac{1}{N_\lambda} \sum K_\lambda(x_0, x_i) \): smooth density estimate
- implemented in \( R \) as \( \text{density}(x, \ldots) \) with a large choice of kernels; default is Gaussian, see (6.23)
- for classification: compute \( \hat{f}_j(X) \) for each class

\[
\hat{\Pr}(Y = j \mid X = x_0) = \frac{\pi_j \hat{f}_j(x_0)}{\sum \pi_k \hat{f}_k(x_0)}
\]

- with \( p \) inputs (§6.6.3); treat the inputs as independent
- \( \hat{f}_j(X) = \prod_{k=1}^{p} \hat{f}_{jk}(X_k) \)
- the Naive Bayes classifier:

\[
\hat{\Pr}(Y = j \mid X = x_0) = \frac{\pi_j \hat{f}_j((x_0))/\Sigma \pi_j \hat{f}_j(x_0)}{\Sigma \pi \hat{f}_j(x_0)}
\]