Nonlinear Regression Models

- Polynomial regression High-order polynomials are not recommended in practice: results are not stable, tail-behavior is bad, and difficult to interpret.
- Spline regression Keypoint is to find the design the matrix F: CS, NCS and etc.
- Smoothing splines
- Local regression
- Generalized additive models

Polynomial Regression

Assume $x_i \in \mathbb{R}$ [a]

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 \cdots + \beta_d x_i^d + \text{err.}$$

Create the following new variables: $X_2 = X^2, \cdots, X_d = X^d$, then treat as a multiple linear regression model:

$$\underset{n\times 1}{\begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{pmatrix}} = \underset{n\times(d+1)}{\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^d \\ 1 & x_2 & x_2^2 & \cdots & x_2^d \\ & & & & \\ 1 & x_n & x_n^2 & \cdots & x_n^d \end{pmatrix}} \underset{(d+1)\times 1}{\begin{pmatrix} \beta_0 \\ \cdots \\ \beta_d \end{pmatrix}} + \text{err.}$$

[a] From now on, assume $x \in \mathbb{R}$ is one-dimensional. Extensions to the multi-dimensional case will be discussed later.

- Fit a polynomial model in `R`: the following two are equivalent

  ```
  > lm(y ~ X + I(X^2) + I(X^3) )
  > lm(y ~ poly(X, 3))
  ```

  where `poly(X,3)` generates a design matrix with columns being the orthogonal polynomials that form a basis for polynomials of degree 3.[a]

- We are more interested in the fitted curve, as well as the prediction at some new location x^*, and less interested in the estimated coefficients because coefficients depend on the choice of the basis function. For example, the two sets of coefficients from the above R commands have totally different interpretation while the two fitted curves are the same.

[a] For the orthogonal polynomials, the j-th basis function involves all the X^k terms for $k = 0, \ldots, j$.

How to draw the fitted curve in R

1. create a grid of values for X (from small to large);

2. obtain the prediction of Y on those X points;

3. connect those points using command "`lines`".

How to choose d?

- Forward approach: keep adding terms until the coefficient for the newly added term is not significant.
- Backward approach: start with a large d, keep eliminating the highest order term if it's not significant until the highest order term is significant.
- Instead of adding/eliminating by hypothesis testing, you can also run forward/backward variable selection with AIC/BIC, or Cross-validation.
- Why not search over all possible sub-models?

- Question: Suppose we've picked d, then should we test whether the other terms, x^j's with $j = 1, \ldots, d-1$, are significant or not?

  Usually, we don't test the significance of the lower-order terms. When we decide to use a polynomial with degree d, by default, we include all the lower-order terms in our model.

- Why? For example, suppose the true curve is equal to X^2, i.e., if we fit a polynomial model with bases $1, X, X^2$, then the optimal model is of size 1.

  However we fit the data using "`poly(X, 2)`", then the optimal model would be a full model with coefficients for all basis functions including the intercept.

- Of course, if you have a particular polynomial function in mind, e.g., the data are collected to test a particular physics formula $Y \approx X^2 +$ constant, then you should test whether you can drop the intercept and the linear term.

- Or if experts believe the relationship between Y and X should be $Y \approx (X-2)^2$, then you should check the `R` output for

  ```
  lm(Y ~ X + I((X-2)^2))
  ```

  to test whether you can drop the linear term and the intercept.

- Otherwise, all we care is whether the data can be fitted by a polynomial of order d where d is the highest order.

Global vs Local

- When the data are too wiggly, we have to use a high-order polynomial. But high-order polynomials are not recommended in practice: results are not stable, tail-behavior is bad, and difficult to interpret.
- Using polynomial functions, we make a global assumption on the true mean function $\mathbb{E}(Y \mid X = x) = f(x)$. But global model is less flexible when data are wiggly. (Recall the discussion on linear regression and KNN.)
- Instead, we can try some local polynomial regression methods: estimate the function locally using piece-wise polynomials (splines) or fitting it locally (local regression).

Spline Models

- Introduction to CS and NCS

  Piece-wise polynomials: we divide the range of x into several intervals, and within each interval $f(x)$ is a low-order polynomial, e.g., cubic or quadratic, but the polynomial coefficients change from interval to interval; in addition we require overall $f(x)$ is continuous up to certain derivatives.

- Regression splines

- Smoothing splines

Cubic Splines

- knots: $a < \xi_1 < \xi_2 < \cdots < \xi_m < b$
- A function g defined on $[a, b]$ is a cubic spline w.r.t knots $\{\xi_i\}_{i=1}^m$ if:

  1) g is a cubic polynomial in each of the $m + 1$ intervals,

  $$g(x) = d_i x^3 + c_i x^2 + b_i x + a_i, \quad x \in [\xi_i, \xi_{i+1}]$$

  where $i = 0 : m$, $\xi_0 = a$ and $\xi_{m+1} = b$;

  2) g is continuous up to the 2nd derivative: since g is continuous up to the 2nd derivative for any point inside an interval, it suffices to check

  $$g^{(0,1,2)}(\xi_i^+) = g^{(0,1,2)}(\xi_i^-), i = 1 : m.$$

当插值曲线的二阶导数连续时，插值曲线将是连续光滑的。这意味着曲线没有不连续的拐点或锐角，而是呈现出平滑的弯曲。

- How many free parameters we need to represent g? $m+4$.

  We need 4 parameters (d_i, c_i, b_i, a_i) for each of the $(m+1)$ intervals, but we also have 3 constraints at each of the m knots, so

$$4(m+1) - 3m = m + 4.$$

Suppose the knots $\{\xi_i\}_{i=1}^m$ are given.

If $g_1(x)$ and $g_2(x)$ are two cubic splines, so is $a_1 g_1(x) + a_2 g_2(x)$, where a_1 and a_2 are two constants.

That is, for a set of given knots, the corresponding cubic splines form a linear space (of functions) with dim $(m + 4)$.

- A set of basis functions for cubic splines (wrt knots $\{\xi_i\}_{i=1}^m$) is given by

$$h_0(x) = 1; \; h_1(x) = x;$$

$$h_2(x) = x^2; \; h_3(x) = x^3;$$

$$h_{i+3}(x) = (x - \xi_i)_+^3, \quad i = 1, 2, \ldots, m.$$

- That is, any cubic spline $f(x)$ can be uniquely expressed as

$$f(x) = \beta_0 + \sum_{i=1}^{m+3} \beta_j h_j(x).$$

- Of course, there are many other choices of the basis functions. For example, `R` uses the B-splines basis functions.

Natural Cubic Splines (NCS)

- A cubic spline on $[a, b]$ is a NCS if its second and third derivatives are zero at a and b.
- That is, a NCS is linear in the two extreme intervals $[a, \xi_1]$ and $[\xi_m, b]$. Note that the linear function in two extreme intervals are totally determined by their neighboring intervals.
- The degree of freedom of NCS's with m knots is m.
- For a curve estimation problem with data $(x_i, y_i)_{i=1}^n$, if we put n knots at the n data points (assumed to be unique), then we obtain a smooth curve (using NCS) passing through all y's.

Regression Splines

- A basis expansion approach:

$$y = \beta_1 h_1(x) + \beta_2 h_2(x) + \cdots + \beta_p h_p(x) + \text{err},$$

  where $p = m + 4$ for regression with cubic splines and $p = m$ for NCS.

- Represent the model on the observed n data points using matrix notation,

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{F}\boldsymbol{\beta}\|^2,$$

where

$$\underset{n\times 1}{\begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{pmatrix}} = \underset{n\times p}{\begin{pmatrix} h_1(x_1) & h_2(x_1) & \cdots & h_p(x_1) \\ h_1(x_2) & h_2(x_2) & \cdots & h_p(x_2) \\ & & & \\ h_1(x_n) & h_2(x_n) & \cdots & h_p(x_n) \end{pmatrix}} \underset{p\times 1}{\begin{pmatrix} \beta_1 \\ \cdots \\ \beta_p \end{pmatrix}} + \text{err}$$

- We can obtain the design matrix F by commands `bs` or `ns` in `R`, and then call the regression function `lm`.

Understand how R counts the degree-of-feedom.

- To generate a cubic spline basis for a given set of x_i's, you can use the command `bs`.
- You can tell `R` the location of knots.
- Or you can tell `R` the df. Recall that a cubic spline with m knots has $m + 4$ df, so we need $m = \text{df} - 4$ knots. By default, `R` puts knots at the $1/(m+1), \ldots, m/(m+1)$ quantiles of $x_{1:n}$.

How R counts the df is a little confusing. The `df` in command `bs` actually means the number of columns of the design matrix returned by `bs`. So if the intercept is not included in the design matrix (which is the default), then the `df` in command `bs` is equal to the real df minus 1.

In summary, real DF = df-1 ot df, depending on whether the intercept is included.

So the following three design matrices (the first two are of $n \times 5$ and the last one is of $n \times 6$) correspond to the same regression model with cubic splines of df 6.

```
> bs(x, knots=quantile(x, c(1/3, 2/3)));
```
m (number of knots) = 2, DF = m+4 = 6

```
> bs(x, df=5);
```
(By default, the intercept is not included)

```
> bs(x, df=6, intercept=TRUE);
```

- To generate a NCS basis for a given set of x_i's, use the command `ns`.
- Recall that the linear functions in the two extreme intervals are totally determined by the other cubic splines, even if no data points are in the two extreme intervals (i.e., data points are inside the two boundary knots). By default, `R` puts the two boundary knots as the min and max of x_i's.
- You can tell `R` the location of knots, which are the interior knots. Recall that a NCS with m knots has m df. So the df is equal to the number of (interior) knots plus 2, where 2 means the two boundary knots.

- Or you can tell `R` the df. If `intercept = TRUE`, then we need $m = \text{df} - 2$ knots, otherwise we need $m = \text{df} - 1$ knots. Again, by default, `R` puts knots at the $1/(m+1), \ldots, m/(m+1)$ quantiles of $x_{1:n}$.

- The following three design matrices (the first two are of $n \times 3$ and the last one is of $n \times 4$) correspond to the same regression model with NCS of df 4.

```
> ns(x, knots=quantile(x, c(1/3, 2/3)));
> ns(x, df=3);
> ns(x, df=4, intercept=TRUE);
```

Choice of Knots

- Approach I: ignore the selection of locations – by default, the knots are located at the quantiles of x_i's, and focus on the selection of number of knots.

  It can be formulated as a variable selection problem (an easier version, since there are just p models, not 2^p)[a] using AIC/BIC/ m-fold CV

- Approach II: put a large number of knots, and then apply Lasso or Ridge.

[a] But note that when number of knots goes from K to $K+1$, the underlying function spaces are not nested since the K knots and the $(K+1)$ knots could be totally different.

Summary: Regression Splines

- Use LS to fit a spline model: Specify the DF[a] p, and then fit a regression model with a design matrix of p columns (including the intercept).
- How to do it in `R`?
- How to select the number/location of knots?

[a] Not the polynomial degree, but the DF of the spline, related to the number of knots.

Smoothing Splines

- In Regression Splines (let's use NCS), we need to choose the number and the location of knots.
- What's a Smoothing Spline? Start with a "naive" solution: put knots at all the observed data points $(x_1, \ldots, x_n)$:

$$\mathbf{y}_{n\times 1} = \mathbf{F}_{n\times n}\boldsymbol{\beta}_{n\times 1}.$$

Instead of selecting knots, let's carry out the following ridge regression (Ω will be defined later):

$$\min_{\boldsymbol{\beta}} \Big[\|\mathbf{y} - \mathbf{F}\boldsymbol{\beta}\|^2 + \lambda\boldsymbol{\beta}^t\Omega\boldsymbol{\beta}\Big],$$

In the linear regression task, Omega = I.

where the tuning parameter λ is often chosen by CV.

- Next we'll see how smoothing splines are derived from a different aspect.

Roughness Penalty Approach

- Let $S[a, b]$ be the space of all "smooth" functions defined on $[a, b]$.
- Among all the functions in $S[a, b]$, look for the minimizer of the following penalized residual sum of squares

$$\text{RSS}_\lambda(g) = \sum_{i=1}^{n} [y_i - g(x_i)]^2 + \lambda \int_a^b [g''(x)]^2 dx,$$

  where λ is a smoothing parameter.

- Theorem. $\min_g \text{RSS}_\lambda(g) = \min_{\tilde{g}} \text{RSS}_\lambda(\tilde{g})$ where $\tilde{g}$ is a NCS with knots at the n data points $x_1, \ldots, x_n$ (WLOG, $x_i \neq x_j$ and $x_1 < x_2 < \cdots < x_n$)

(WLOG, assume $n \geq 2$.) Let g be a smooth function on $[a, b]$ and $\tilde{g}$ be a NCS with knots at $\{x_i\}_{i=1}^n$ satisfying

$$g(x_i) = \tilde{g}(x_i), \quad i = 1 : n. \tag{1}$$

在x处，两个函数有相同的值

First, such $\tilde{g}$ exists since NCS with n knots has n dfs, so we can pick the n coefficients properly such that (1) is satisfied.

Next we want to show that

$$\int_a^b [g''(x)]^2 dx \geq \int_a^b [\tilde{g}''(x)]^2 dx \quad (*)$$

with equality holds if and only if $\tilde{g} \equiv g$. Recall that

$$\text{RSS}_\lambda(g) = \sum_{i=1}^{n} [y_i - g(x_i)]^2 + \lambda \int_a^b [g''(x)]^2 dx.$$

So it is easy to conclude that if $(*)$ holds,

$$\text{RSS}_\lambda(\tilde{g}) \leq \text{RSS}_\lambda(g).$$

That is, for any smooth function g, we can find a NCS $\tilde{g}$ which matches $g(x_i)$ on the n samples and whose penalized residual sum of squares is not worse than the one of g. So Theorem follows.

PROOF : We will use integration by parts and the fact that $\tilde{g}$ is a NCS.

$$h(x) = g(x) - \tilde{g}(x). \quad \text{Note } h(x_i) = 0 \text{ for } i = 1, \ldots, n.$$

$$\begin{aligned}
\int_a^b [g''(x)]^2 dx &= \int_a^b \left[\tilde{g}''(x) + h''(x)\right]^2 dx \\
&= \int_a^b [\tilde{g}''(x)]^2 dx + \int_a^b [h''(x)]^2 dx + 2 \underbrace{\int_a^b \tilde{g}''(x) h''(x) dx}_{=0}
\end{aligned}$$

$$\begin{aligned}
\int_a^b \tilde{g}''(x) h''(x) dx &= \int_a^b \tilde{g}''(x) dh'(x) = \underbrace{h'(x)\tilde{g}''(x)\Big|_a^b}_{=0 \text{ 因为NCS}} - \int_a^b h'(x)\tilde{g}^{(3)}(x) dx \\
&= -\sum_{i=1}^{n-1} \tilde{g}^{(3)}\left(\frac{x_i + x_{i+1}}{2}\right) \int_{x_i}^{x_{i+1}} h'(x) dx \\
&= -\sum_{i=1}^{n-1} \tilde{g}^{(3)}\left(\frac{x_i + x_{i+1}}{2}\right) h(x)\Big|_{x_i}^{x_{i+1}} = 0
\end{aligned}$$

因为g_tilt是NCS，最高次只到3次，所以三次偏导是常数

= 0

Smoothing Splines

Write $g(x) = \sum_{i=1}^{n} \beta_i h_i(x)$ where h_i's are basis functions for NCS with knots at $x_1, \ldots, x_n$.

$$\sum_{i=1}^{n} [y_i - g(x_i)]^2 = (\mathbf{y} - \mathbf{F}\boldsymbol{\beta})^t (\mathbf{y} - \mathbf{F}\boldsymbol{\beta}),$$

where $\mathbf{F}_{n\times n}$ with $\mathbf{F}_{ij} = h_j(x_i)$.

$$\begin{aligned} \int_a^b \left[g''(x)\right]^2 dx &= \int \Big[\sum_i \beta_i h_i''(x)\Big]^2 dx \\ &= \sum_{i,j} \beta_i \beta_j \int h_i''(x) h_j''(x) dx = \boldsymbol{\beta}^t \Omega \boldsymbol{\beta}, \end{aligned}$$

where $\Omega_{n\times n}$ with $\Omega_{ij} = \int_a^b h_i''(x) h_j''(x) dx$.

So

$$\mathsf{RSS}_\lambda(\boldsymbol{\beta}) = (\mathbf{y} - \mathbf{F}\boldsymbol{\beta})^t(\mathbf{y} - \mathbf{F}\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}^t\Omega\boldsymbol{\beta},$$

and the solution is

$$\begin{aligned}
\hat{\boldsymbol{\beta}} &= \arg\min_{\boldsymbol{\beta}} \mathsf{RSS}_\lambda(\boldsymbol{\beta}) \\
&= (\mathbf{F}^t\mathbf{F} + \lambda\Omega)^{-1}\mathbf{F}^t\mathbf{y} \\
\hat{\mathbf{y}} &= \mathbf{F}(\mathbf{F}^t\mathbf{F} + \lambda\Omega)^{-1}\mathbf{F}^t\mathbf{y} \\
&= S_\lambda\mathbf{y}
\end{aligned}$$

What if we use a different set of basis functions $\tilde{h}_1(x), \ldots, \tilde{h}_n(x)$?

beta-matrix不同但是S相同

- Demmler & Reinsch (1975): a basis with double orthogonality property, i.e.

$$\mathbf{F}^t\mathbf{F} = \mathbf{I}, \quad \Omega = \mathsf{diag}(d_i),$$

where d_i's are arranged in an increasing order and in addition $d_2 = d_1 = 0$ (Why?).

- Using this basis, we have

$$\begin{aligned} \hat{\boldsymbol{\beta}} &= (\mathbf{F}^t\mathbf{F} + \lambda\Omega)^{-1}\mathbf{F}^t\mathbf{y} \\ &= (\mathbf{I} + \lambda\mathsf{diag}(d_i))^{-1}\mathbf{F}^t\mathbf{y}, \end{aligned}$$

i.e.,

$$\hat{\beta}_i = \frac{1}{1+\lambda d_i}\hat{\beta}_i^{(\mathsf{LS})}.$$

- Smoother matrix S_λ

$$\hat{\mathbf{y}} = \mathbf{F}\hat{\boldsymbol{\beta}} = \mathbf{F}(\mathbf{F}^t\mathbf{F} + \lambda\Omega)^{-1}\mathbf{F}^t\mathbf{y} = S_\lambda\mathbf{y}.$$

- Using D&R basis,

$$S_\lambda = \mathbf{F}\text{diag}\Big(\frac{1}{1+\lambda d_i}\Big)\mathbf{F}^t.$$

  So columns of $\mathbf{F}$ are the eigen-vectors of S_λ, which does not depend on λ.

- Effective df of a smoothing spline:

$$df(\lambda) = \text{tr}S_\lambda = \sum_{i=1}^{n}\frac{1}{1+\lambda d_i}.$$

- Check the R page to see what the DR basis functions look like.

Weighted Spline Models

We can always assume all x_i's are different, otherwise, we just need to fit a weighted regression model due to the following argument. Suppose the first two obs have the same x value, i.e.,

$$(x_1, y_1), \quad (x_2, y_2), \qquad \text{where } x_1 = x_2.$$

Then

$$\begin{aligned}
\big[y_1 - g(x_1)\big]^2 + \big[y_2 - g(x_1)\big]^2 &= \sum_{i=1}^{2} \big[y_i - \frac{y_1+y_2}{2} + \frac{y_1+y_2}{2} - g(x_1)\big]^2 \\
&= \big(y_1 - \frac{y_1+y_2}{2}\big)^2 + \big(y_2 - \frac{y_1+y_2}{2}\big)^2 \\
&\quad +2\big[\frac{y_1+y_2}{2} - g(x_1)\big]^2
\end{aligned}$$

So we can replace the first two obs by one, $(x_1, \frac{y_1+y_2}{2})$, and its weight is 2 while the weights for other obs are 1.

Choice of λ

- Leave-one-out (LOO) CV (n-fold CV)

$$\begin{aligned} \text{LOO-CV}(\lambda) &= \frac{1}{n}\sum_{i=1}^{n}[\, y_i - \hat{g}^{[-i]}(x_i)]^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{g}(x_i)}{1 - S_\lambda(i,i)}\right)^2 \end{aligned}$$

  where $\hat{g}^{[-i]}$ denotes the model learned based on $n-1$ samples (i.e., leave the i-th sample out).

- Generalized CV

$$\text{GCV}(\lambda) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{g}(x_i)}{1 - \frac{1}{n}\text{tr}S_\lambda}\right)^2$$

- In R, you tune λ through $df(\lambda) = \text{tr}S_\lambda = \sum_{i=1}^{n}\frac{1}{1+\lambda d_i}$.

A Simple Formula for LOO-CV

Initially, it seems that we need to repeatedly fit n models when leaving out each sample. But it turns out that LOO prediction error can be computed based on the original model (the one that uses all the n samples):

$$y_i - \hat{y}_i^{[-i]} = \frac{y_i - \hat{y}_i}{1 - S_{ii}}, \tag{2}$$

where

- S_{ii} is the (i, i)-th entry of the smoothing matrix S associated with the original SS (smoothing spline) model (for simplicity we suppress the subscript λ in S_λ), and
- $\hat{y}_i$ denotes the prediction at x_i from the original SS model (i.e., the ith entry of vector $S\mathbf{y}$).

So LOO-CV is actually much faster to compute than the general m-fold CV.

- In addition to smoothing splines, this simple formula (2) is true for many models, such as multiple linear regression model, polynomial regression model and ridge regression, where the prediction at the n sample points can be written as

$$\hat{\mathbf{y}}_{n\times 1} = S_{n\times n}\mathbf{y}_{n\times 1}.$$

  Note that S is a matrix computed based on x_i's, not y_i's.

- Sometimes the following GCV is used to approximate LOO-CV (to further reduce the computation):

$$\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i^{[-i]}\right)^2 = \frac{1}{n}\sum_{i}\left(\frac{y_i - \hat{y}_i}{1 - S_{ii}}\right)^2 \approx \frac{1}{n}\sum_{i}\left(\frac{y_i - \hat{y}_i}{1 - m}\right)^2$$

  where $m = \frac{1}{n}\sum_i S_{ii}$ is the average of the trace of S. Note that $\sum_i S_{ii} = \text{tr}S$ is basically the df of a linear model (including the intercept) or the effective df of a ridge regression model.

Summary: Smoothing Splines

- Start with a model with the maximum complexity: NCS with knots at n (unique) x points.
- Fit a Ridge Regression model on the data. If we parameterize the NCS function space by the DR basis, then the design matrix is orthogonal and the corresponding coefficient is penalized differently: no penalty for the two linear basis functions, higher penalties for wigglier basis functions.
- How to do it in R?
- How to select the tuning parameter λ or equivalently the df?
- What if we have collected two obs at the same location x?

Local Regression

Recall k-NN for regression

$$\hat{f}(x) = \frac{1}{k} \sum_{i \in \mathcal{N}_k(x)} y_i,$$

which falls into a class of curve estimates known as linear smoothers:

$$\hat{f}(x) = \sum_{i=1}^{n} \ell_i(x) y_i,$$

where our estimate at each x is a weighted average of y_i's with $\ell_i(x)$ being the weight assigned to the i-th sample and $\sum_i \ell_i(x) = 1$.

In particular,

$$\hat{\mathbf{y}}_{n\times 1} = \begin{pmatrix} \hat{f}(x_1) \\ \hat{f}(x_2) \\ \vdots \\ \hat{f}(x_n) \end{pmatrix} = \begin{pmatrix} \ell_1(x_1) & \ell_2(x_1) & \cdots & \ell_n(x_1) \\ \ell_1(x_2) & \ell_2(x_2) & \cdots & \ell_n(x_2) \\ \vdots & \vdots & & \vdots \\ \ell_1(x_n) & \ell_2(x_n) & \cdots & \ell_n(x_n) \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

That is, the fitted value is a linear[a] transformation of $\mathbf{y}$ (note that S does not depend on $\mathbf{y}$):

$$\hat{\mathbf{y}} = S\mathbf{y}.$$

[a]It doesn't mean that $\hat{f}(x)$ is a linear function of x.

The class of linear smoothers contains many curve estimators we have learned so far. We will define the effective df of a linear smoother $\hat{\mathbf{y}} = S\mathbf{y}$ to be $\text{tr}(S)$.

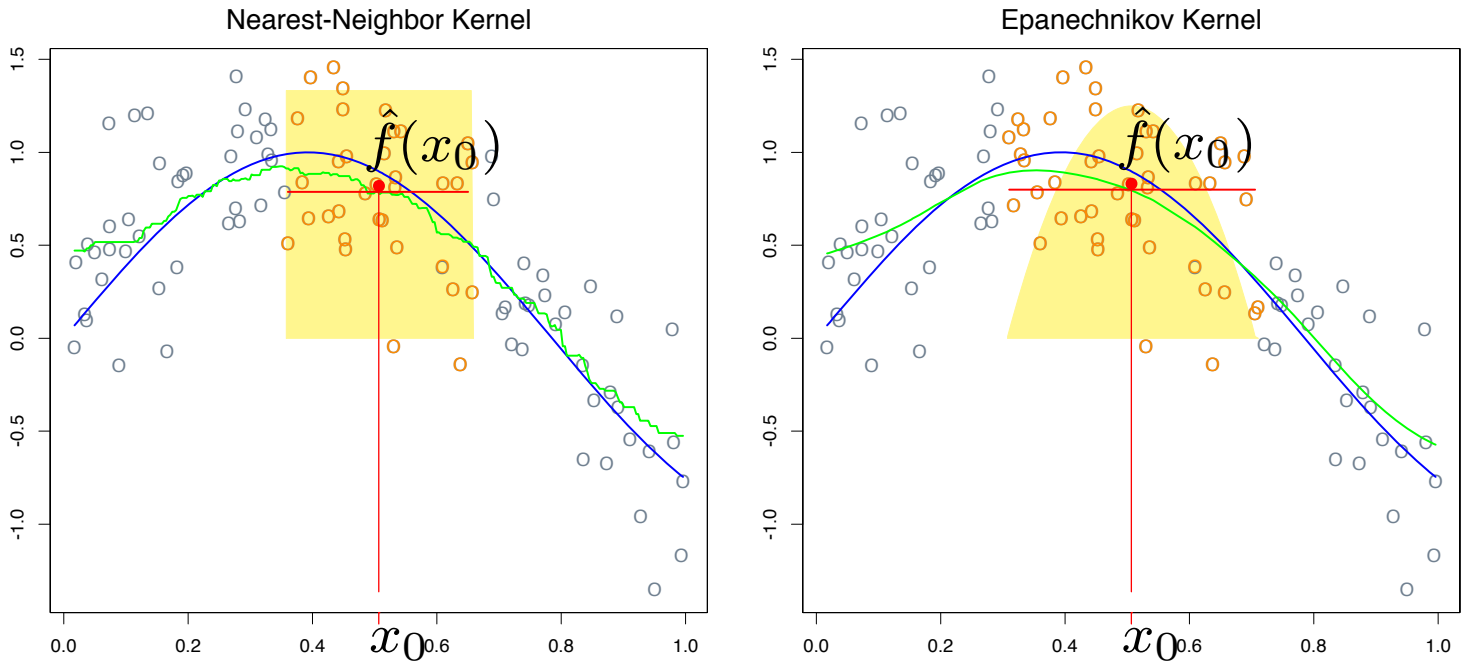
The simple formula for LOO-CV holds for $\hat{\mathbf{y}} = S\mathbf{y}$, that is,

$$y_i - \hat{y}_i^{[-i]} = \frac{y_i - \hat{y}_i}{1 - S_{ii}}.$$

Each linear smoother usually has a smoothing parameter h.

Kernel Smoothing

Issues with kNN: fitted curve looks jagged. This is because $\ell_i(x)$ in $\hat{f}(x) = \sum_{i=1}^{n} \ell_i(x) y_i$ is not continuous.



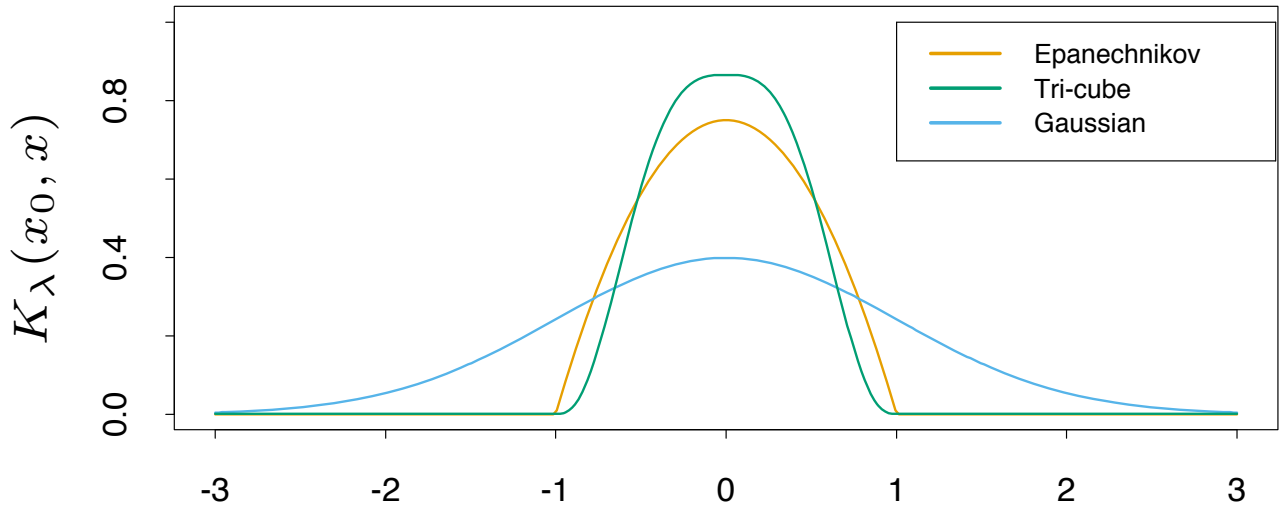
$K : \mathbb{R} \to \mathbb{R}$ is a kernel function satisfying

$$\int K(x)dx = 1, \quad \int xK(x)dx = 0, \quad \int x^2 K(x)dx < \infty.$$

For example:

$$\text{Gaussian kernel} \quad : \quad K(x) = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2)$$

$$\text{Epanechnikov kernel} \quad : \quad K(x) = (3/4)(1 - x^2), \text{ if } |x| \leq 1; \quad 0, \text{ otherwise.}$$



The local average based on a kernel function K can be written as

$$\hat{f}(x) = \frac{\sum_{i=1}^{n} K(\frac{x-x_i}{h}) y_i}{\sum_{j=1}^{n} K(\frac{x-x_j}{h})} = \sum_{i=1}^{n} \ell_i(x) y_i$$

Choice of kernel: not important

Choice of bandwidth h: crucial

Local Regression

Consider a Taylor expansion at x_0:

$$\begin{aligned} f(x_i) &\approx f(x_0) + f'(x_0)(x_i - x_0) + f''(x_0)(x_i - x_0)^2 + \cdots \\ y_i &= f(x_i) + \text{err} \\ &\approx f(x_0) + f'(x_0)(x_i - x_0) + f''(x_0)(x_i - x_0)^2 + \text{err} \end{aligned}$$

At each location x_0, fit a weighted linear or polynomial regression model on $(y_i, z_i)_{i=1}^n$ where $z_i = x_i - x_0$, then use the intercept to predict $f(x_0)$. This prediction can still be expressed as a weighted average of y_i's, i.e., $\hat{f}(x_0) = \sum_i \ell_i(x_0) y_i$.

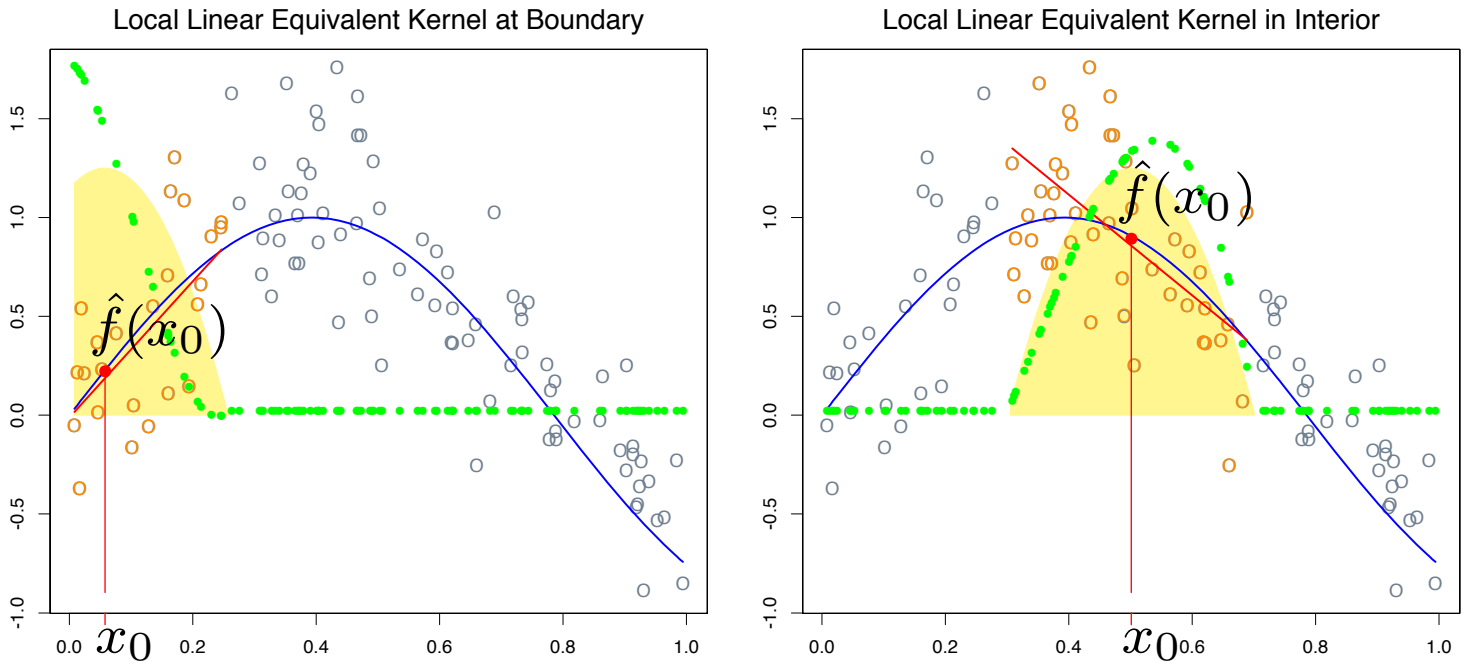


FIGURE 6.4. *The green points show the equivalent kernel $l_i(x_0)$ for local regression. These are the weights in $\hat{f}(x_0) = \sum_{i=1}^{N} l_i(x_0)y_i$, plotted against their cor-*

LOESS

The value of $\hat{f}(x_0)$ is obtained as follows:

1. The αn points, those with $|x_i - x|$ smallest, are called the neighborhood of x_0: $\mathcal{N}(x_0)$.

2. A weighted least-squares linear (or quadratic) regression

$$\hat{f}(x) = \hat{\beta}_0 + \hat{\beta}_1(x - x_0)$$

is fit in $\mathcal{N}(x_0)$. That is, choose $\hat{\beta}_0$ and $\hat{\beta}_1$ to minimize

$$\sum_{x_i \in \mathcal{N}(x_0)} w_i[y_i - \beta_0 - \beta_1(x_i - x_0)]^2$$

where the weights $w_i = (1 - u_i^3)^3$ with

$$u_i = \frac{|x_i - x_0|}{\max_{\mathcal{N}(x_0)} |x_j - x_0|}.$$

Generalized Additive Models (GAMs)

- Curve estimation for one-dimension x:

$$y = f(x) + \text{err}$$

- Additive model for multi-dimensional $\mathbf{x} = (x_1, \ldots, x_p)^t$

$$y = \alpha_1 + f_1(x_1) + f_2(x_2) + \cdots + f_p(x_p) + \text{err}.$$

- Fitting a GAM: iteratively fit each f_i using Smoothing Splines or Loess.

- GAMs in R: `mgcv`, `gam`, ...