Moving Beyond Linearity

The truth is never linear!

Moving Beyond Linearity

The truth is never linear!
Or almost never!

Moving Beyond Linearity

The truth is never linear!
Or almost never!

But often the linearity assumption is good enough.

Moving Beyond Linearity

The truth is never linear!
Or almost never!

But often the linearity assumption is good enough.

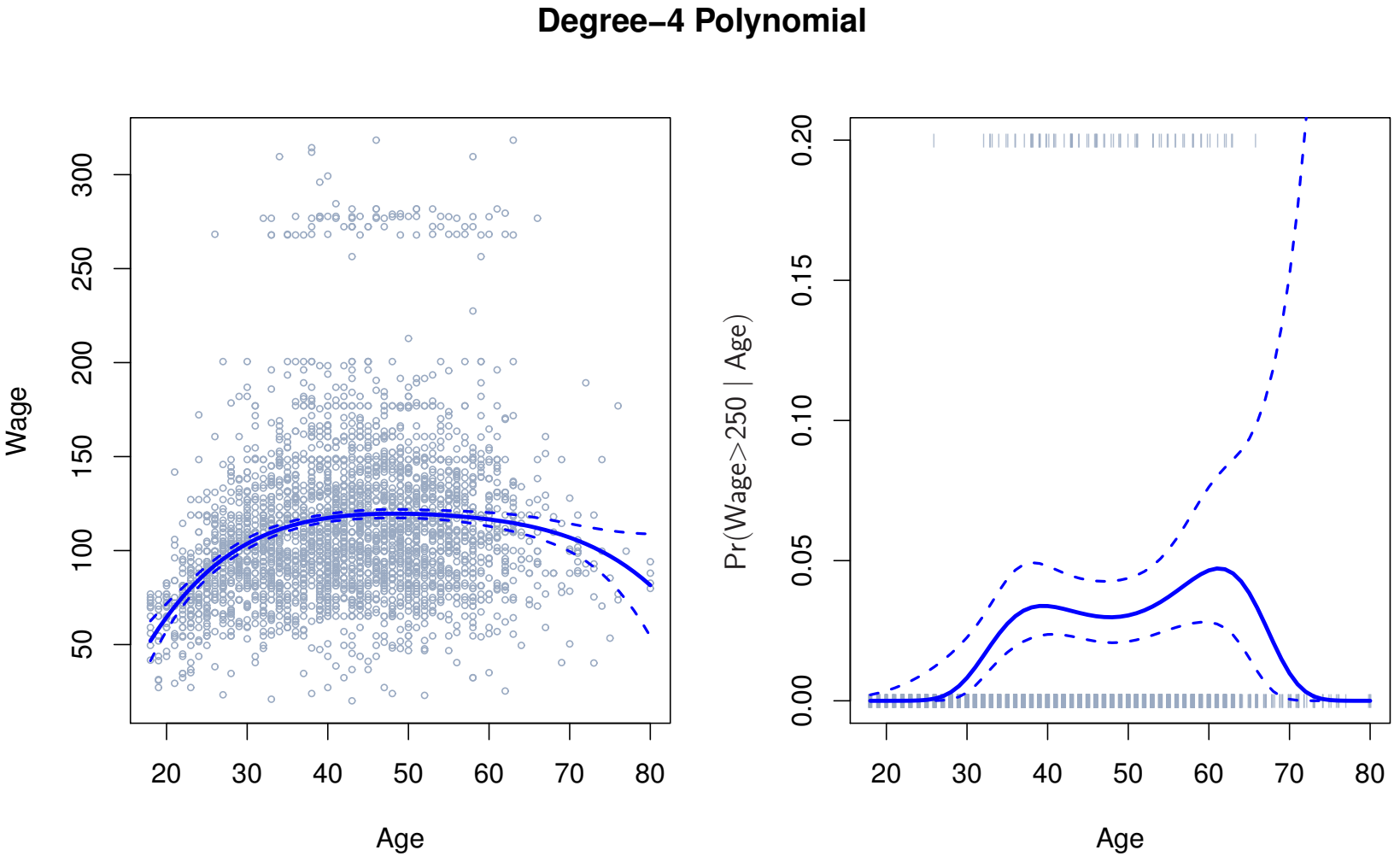
When its not ...

- polynomials,
- step functions,
- splines,
- local regression, and
- generalized additive models

offer a lot of flexibility, without losing the ease and interpretability of linear models.

Polynomial Regression

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \ldots + \beta_d x_i^d + \epsilon_i$$



Details

- Create new variables $X_1 = X$, $X_2 = X^2$, etc and then treat as multiple linear regression.

Details

- Create new variables $X_1 = X$, $X_2 = X^2$, etc and then treat as multiple linear regression.
- Not really interested in the coefficients; more interested in the fitted function values at any value x_0:

$$\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0 + \hat{\beta}_2 x_0^2 + \hat{\beta}_3 x_0^3 + \hat{\beta}_4 x_0^4.$$

Details

- Create new variables $X_1 = X$, $X_2 = X^2$, etc and then treat as multiple linear regression.
- Not really interested in the coefficients; more interested in the fitted function values at any value x_0:

$$\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0 + \hat{\beta}_2 x_0^2 + \hat{\beta}_3 x_0^3 + \hat{\beta}_4 x_0^4.$$

- Since $\hat{f}(x_0)$ is a linear function of the $\hat{\beta}_\ell$, can get a simple expression for *pointwise-variances* $\mathrm{Var}[\hat{f}(x_0)]$ at any value x_0. In the figure we have computed the fit and pointwise standard errors on a grid of values for x_0. We show $\hat{f}(x_0) \pm 2 \cdot \mathrm{se}[\hat{f}(x_0)]$.

Details

- Create new variables $X_1 = X$, $X_2 = X^2$, etc and then treat as multiple linear regression.
- Not really interested in the coefficients; more interested in the fitted function values at any value x_0:

$$\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0 + \hat{\beta}_2 x_0^2 + \hat{\beta}_3 x_0^3 + \hat{\beta}_4 x_0^4.$$

- Since $\hat{f}(x_0)$ is a linear function of the $\hat{\beta}_\ell$, can get a simple expression for *pointwise-variances* $\mathrm{Var}[\hat{f}(x_0)]$ at any value x_0. In the figure we have computed the fit and pointwise standard errors on a grid of values for x_0. We show $\hat{f}(x_0) \pm 2 \cdot \mathrm{se}[\hat{f}(x_0)]$.
- We either fix the degree d at some reasonably low value, else use cross-validation to choose d.

Details continued

- Logistic regression follows naturally. For example, in figure we model

$$\Pr(y_i > 250|x_i) = \frac{\exp(\beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_d x_i^d)}{1 + \exp(\beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_d x_i^d)}.$$

- To get confidence intervals, compute upper and lower bounds on *on the logit scale*, and then invert to get on probability scale.

Details continued

- Logistic regression follows naturally. For example, in figure we model

$$\Pr(y_i > 250|x_i) = \frac{\exp(\beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_d x_i^d)}{1 + \exp(\beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_d x_i^d)}.$$

- To get confidence intervals, compute upper and lower bounds on *on the logit scale*, and then invert to get on probability scale.
- Can do separately on several variables—just stack the variables into one matrix, and separate out the pieces afterwards (see GAMs later).

Details continued

- Logistic regression follows naturally. For example, in figure we model

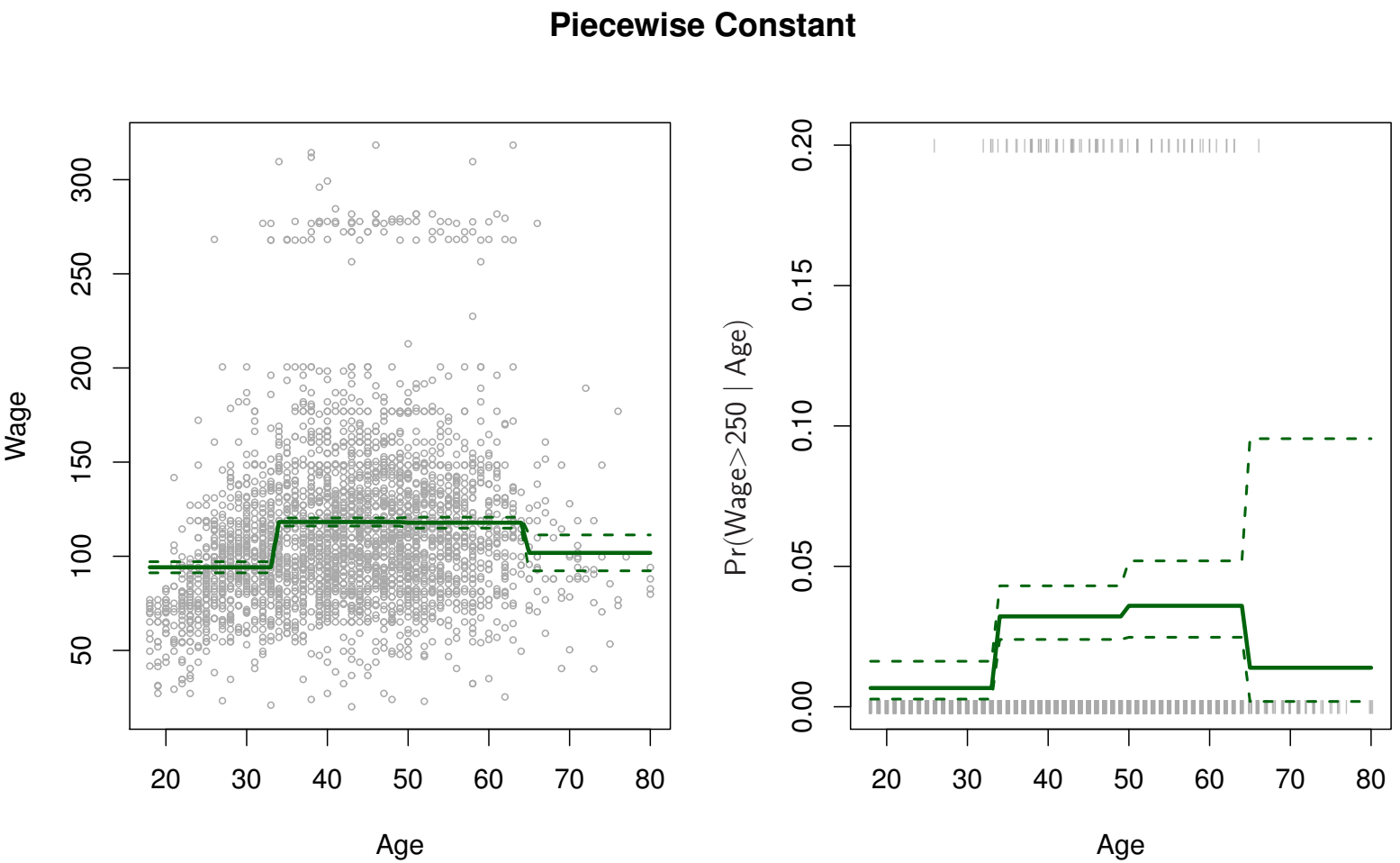
$$\Pr(y_i > 250|x_i) = \frac{\exp(\beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_d x_i^d)}{1 + \exp(\beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_d x_i^d)}.$$

- To get confidence intervals, compute upper and lower bounds on *on the logit scale*, and then invert to get on probability scale.
- Can do separately on several variables—just stack the variables into one matrix, and separate out the pieces afterwards (see GAMs later).
- Caveat: polynomials have notorious tail behavior — very bad for extrapolation.
- Can fit using $\texttt{y} \sim \texttt{poly}(\texttt{x}, \texttt{degree} = 3)$ in formula.

Step Functions

Another way of creating transformations of a variable — cut the variable into distinct regions.

$$C_1(X) = I(X < 35), \quad C_2(X) = I(35 \leq X < 50), \ldots, C_3(X) = I(X \geq 65)$$



Step functions

$$
\begin{aligned}
C_0(X) &= I(X < c_1), \\
C_1(X) &= I(c_1 \leq X < c_2), \\
C_2(X) &= I(c_2 \leq X < c_3), \\
&\vdots \\
C_{K-1}(X) &= I(c_{K-1} \leq X < c_K), \\
C_K(X) &= I(c_K \leq X),
\end{aligned}
$$

Step functions

- Notice that for any value of X, $C_0(X) + C_1(X) + \cdots + C_K(X) = 1$, since X must be in exactly one of the $K + 1$ intervals.
- We then use least squares to fit linear model using $C_1(X), C_2(X), \cdots, C_K(X)$ as predictors:

$$y_i = \beta_0 + \beta_1 C_1(X) + \beta_1 C_1(X) + \cdots + \beta_K C_K(X) + \epsilon_i$$

- For logistic regression

$$Pr(y_i > 250|x_i) = \frac{exp(\beta_0 + \beta_1 C_1(x_i)) + \cdots + \beta_K C_K(x_i)}{1 + exp(\beta_0 + \beta_1 C_1(x_i)) + \cdots + \beta_K C_K(x_i)}$$

Step functions continued

- Easy to work with. Creates a series of dummy variables representing each group.

Step functions continued

- Easy to work with. Creates a series of dummy variables representing each group.
- Useful way of creating interactions that are easy to interpret. For example, interaction effect of `Year` and `Age`:

$$I(\texttt{Year} < 2005) \cdot \texttt{Age}, \quad I(\texttt{Year} \geq 2005) \cdot \texttt{Age}$$

would allow for different linear functions in each age category.

Step functions continued

- Easy to work with. Creates a series of dummy variables representing each group.
- Useful way of creating interactions that are easy to interpret. For example, interaction effect of `Year` and `Age`:

$$I(\texttt{Year} < 2005) \cdot \texttt{Age}, \quad I(\texttt{Year} \geq 2005) \cdot \texttt{Age}$$

  would allow for different linear functions in each age category.
- In R: `I(year < 2005)` or `cut(age, c(18, 25, 40, 65, 90))`.

Step functions continued

- Easy to work with. Creates a series of dummy variables representing each group.
- Useful way of creating interactions that are easy to interpret. For example, interaction effect of `Year` and `Age`:

$$I(\texttt{Year} < 2005) \cdot \texttt{Age}, \quad I(\texttt{Year} \geq 2005) \cdot \texttt{Age}$$

  would allow for different linear functions in each age category.
- In R: `I(year < 2005)` or `cut(age, c(18, 25, 40, 65, 90))`.
- Choice of cutpoints or *knots* can be problematic. For creating nonlinearities, smoother alternatives such as *splines* are available.

Basis functions

- Polynomial and piecewise-constant regression models are in fact special cases of a basis function approach.
- $$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_K b_K(x_i) + \epsilon_i$$

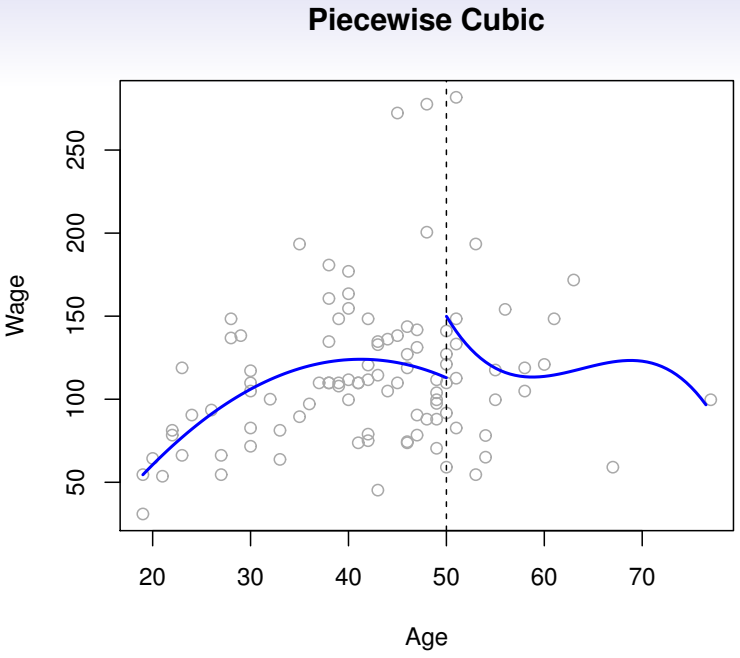
  where basis function $b_1(\cdot), b_2(\cdot), \cdots, b_K(\cdot)$ are fixed and known.
- For polynomial regression, $b_j(x_i) = x_i^j$.
- For piecewise constant functions, $b_j(x_i) = I(c_j \leq x_i < c_{j+1})$.

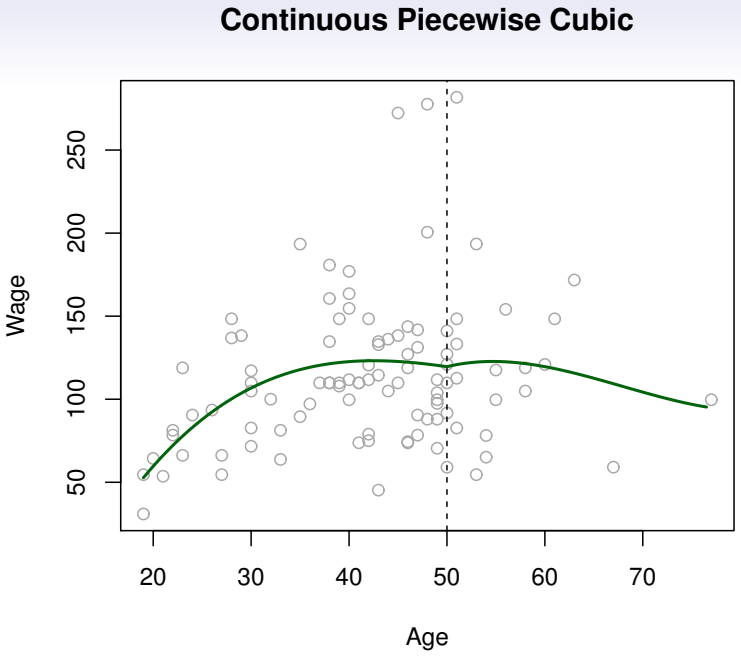
Piecewise Polynomials

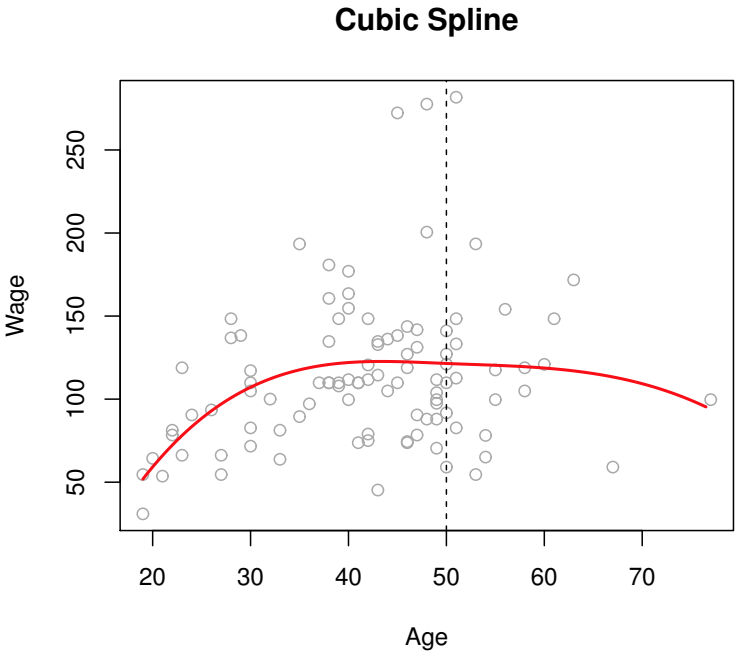
- Instead of a single polynomial in X over its whole domain, we can rather use different polynomials in regions defined by knots. E.g. (see figure)

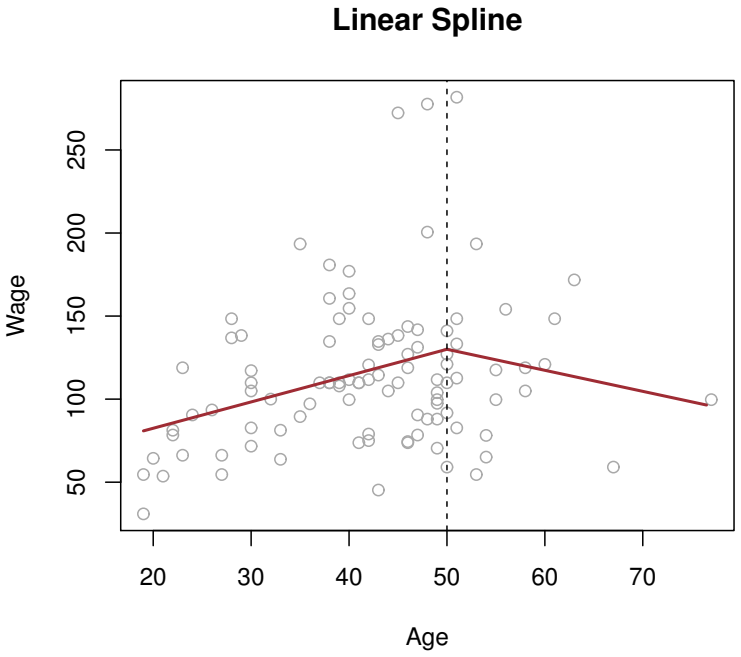
$$y_i = \begin{cases} \beta_{01} + \beta_{11} x_i + \beta_{21} x_i^2 + \beta_{31} x_i^3 + \epsilon_i & \text{if } x_i < c; \\ \beta_{02} + \beta_{12} x_i + \beta_{22} x_i^2 + \beta_{32} x_i^3 + \epsilon_i & \text{if } x_i \geq c. \end{cases}$$

- Better to add constraints to the polynomials, e.g. continuity.
- *Splines* have the "maximum" amount of continuity.


Piecewise Cubic
Wage
Age
50
100
150
200
250
20
30
40
50
60
70


Continuous Piecewise Cubic
Wage
Age
50
100
150
200
250
20
30
40
50
60
70


Cubic Spline
Wage
Age
50
100
150
200
250
20
30
40
50
60
70


Linear Spline
Wage
Age
50
100
150
200
250
20
30
40
50
60
70

Linear Splines

A linear spline with knots at ξ_k, $k = 1, \ldots, K$ is a piecewise linear polynomial continuous at each knot.

We can represent this model as

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i,$$

where the b_k are *basis functions*.

Linear Splines

A linear spline with knots at ξ_k, $k = 1, \ldots, K$ is a piecewise linear polynomial continuous at each knot.

We can represent this model as

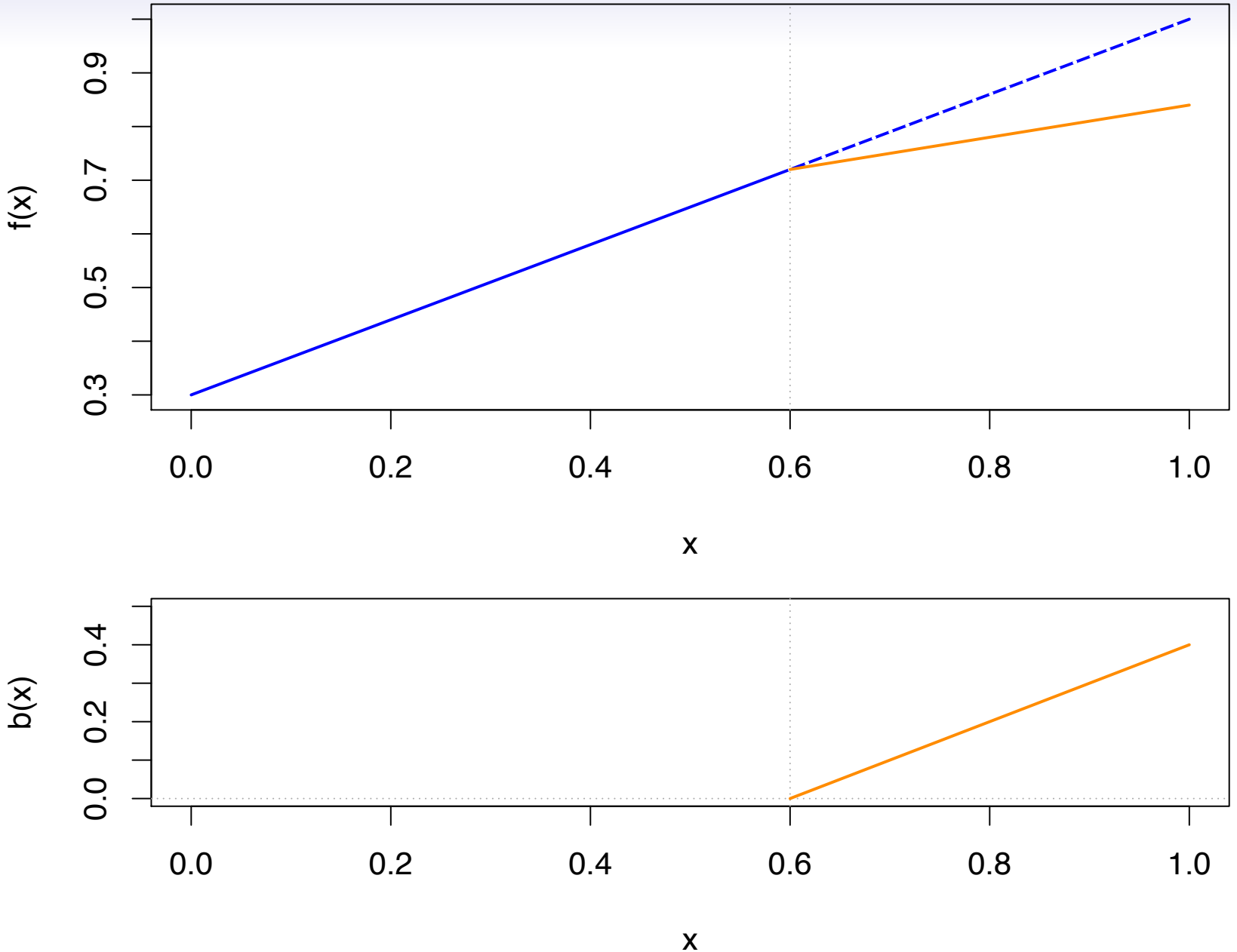
$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i,$$

where the b_k are *basis functions*.

$$\begin{aligned} b_1(x_i) &= x_i \\ b_{k+1}(x_i) &= (x_i - \xi_k)_+, \quad k = 1, \ldots, K \end{aligned}$$

Here the $()_+$ means *positive part*; i.e.

$$(x_i - \xi_k)_+ = \begin{cases} x_i - \xi_k & \text{if } x_i > \xi_k \\ 0 & \text{otherwise} \end{cases}$$


f(x)
0.3
0.5
0.7
0.9
0.0
0.2
0.4
0.6
0.8
1.0
x
b(x)
0.0
0.2
0.4
0.0
0.2
0.4
0.6
0.8
1.0
x

Cubic Splines

A cubic spline with knots at ξ_k, $k = 1, \ldots, K$ is a piecewise cubic polynomial with continuous derivatives up to order 2 at each knot.

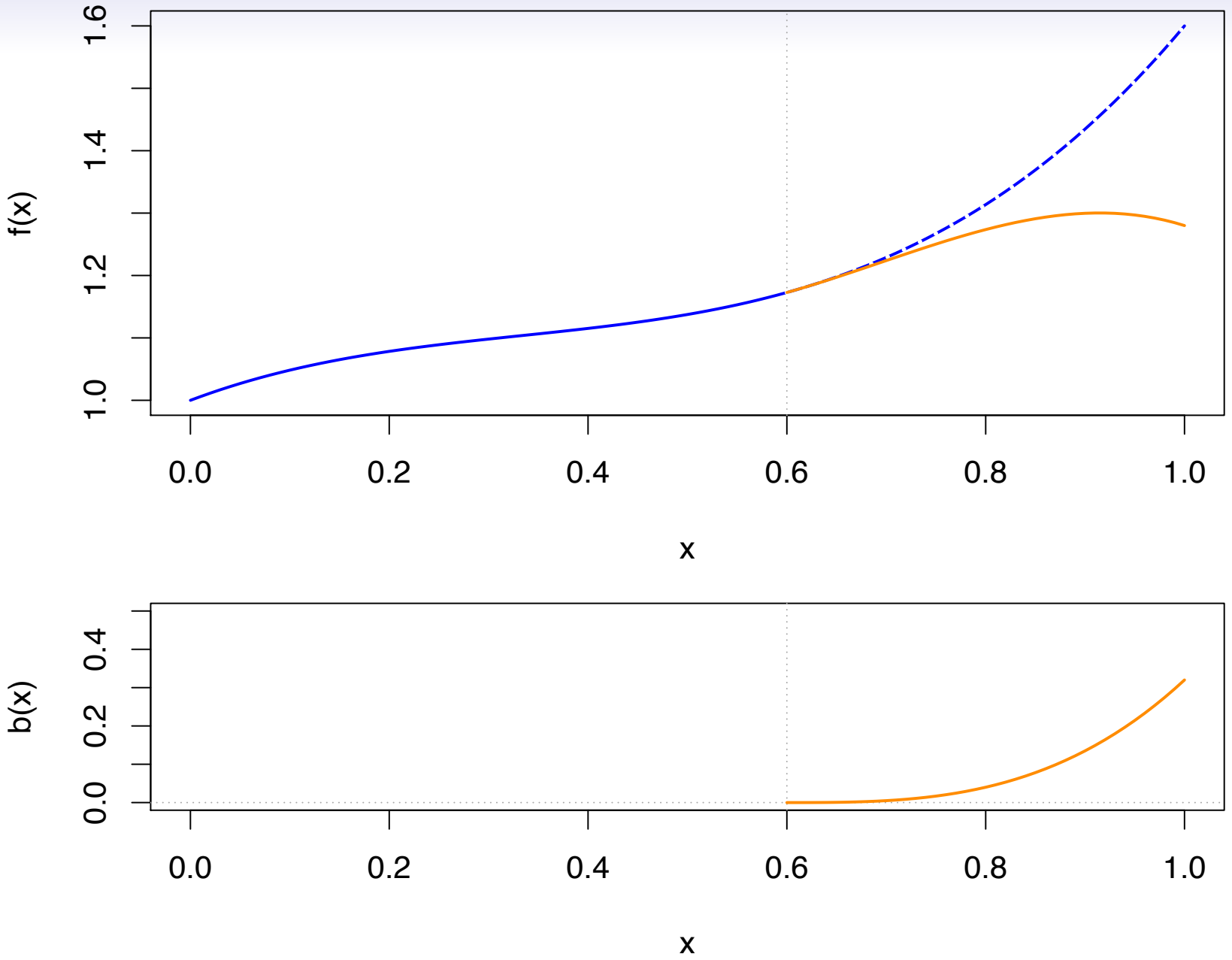
Again we can represent this model with truncated power basis functions

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i,$$

$$\begin{aligned} b_1(x_i) &= x_i \\ b_2(x_i) &= x_i^2 \\ b_3(x_i) &= x_i^3 \\ b_{k+3}(x_i) &= (x_i - \xi_k)_+^3, \quad k = 1, \ldots, K \end{aligned}$$

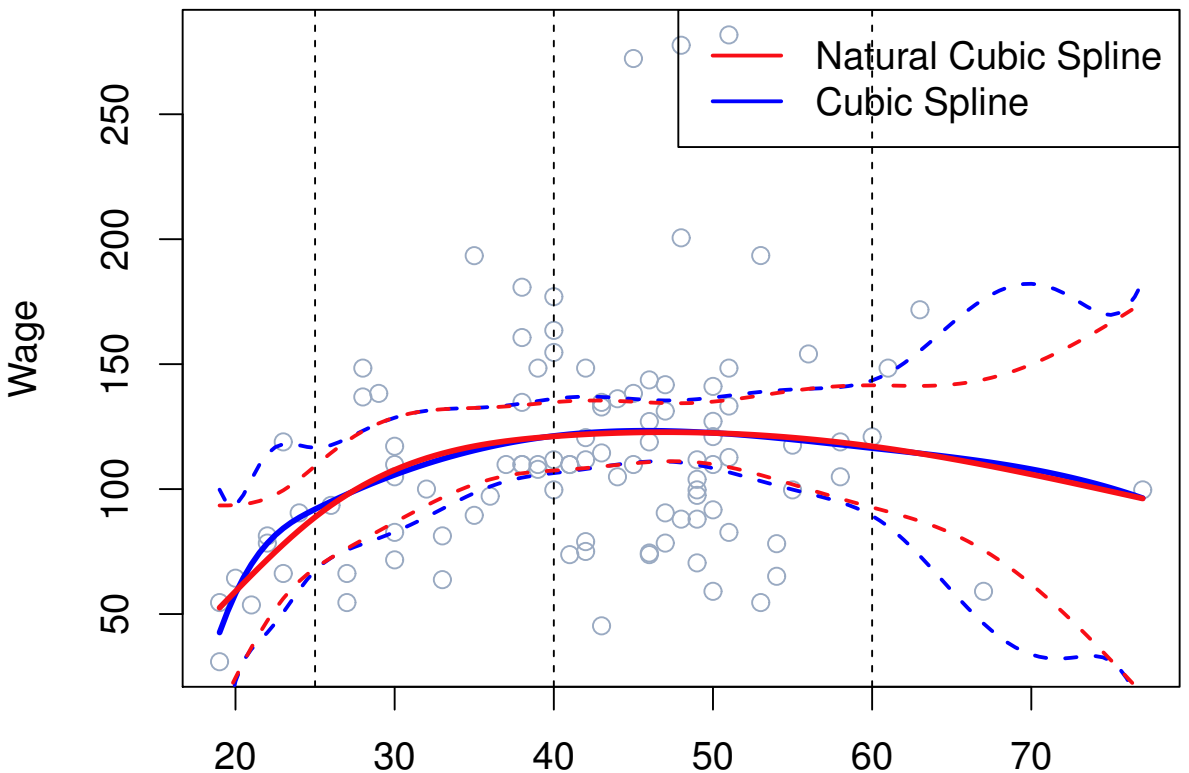
where

$$(x_i - \xi_k)_+^3 = \begin{cases} (x_i - \xi_k)^3 & \text{if } x_i > \xi_k \\ 0 & \text{otherwise} \end{cases}$$

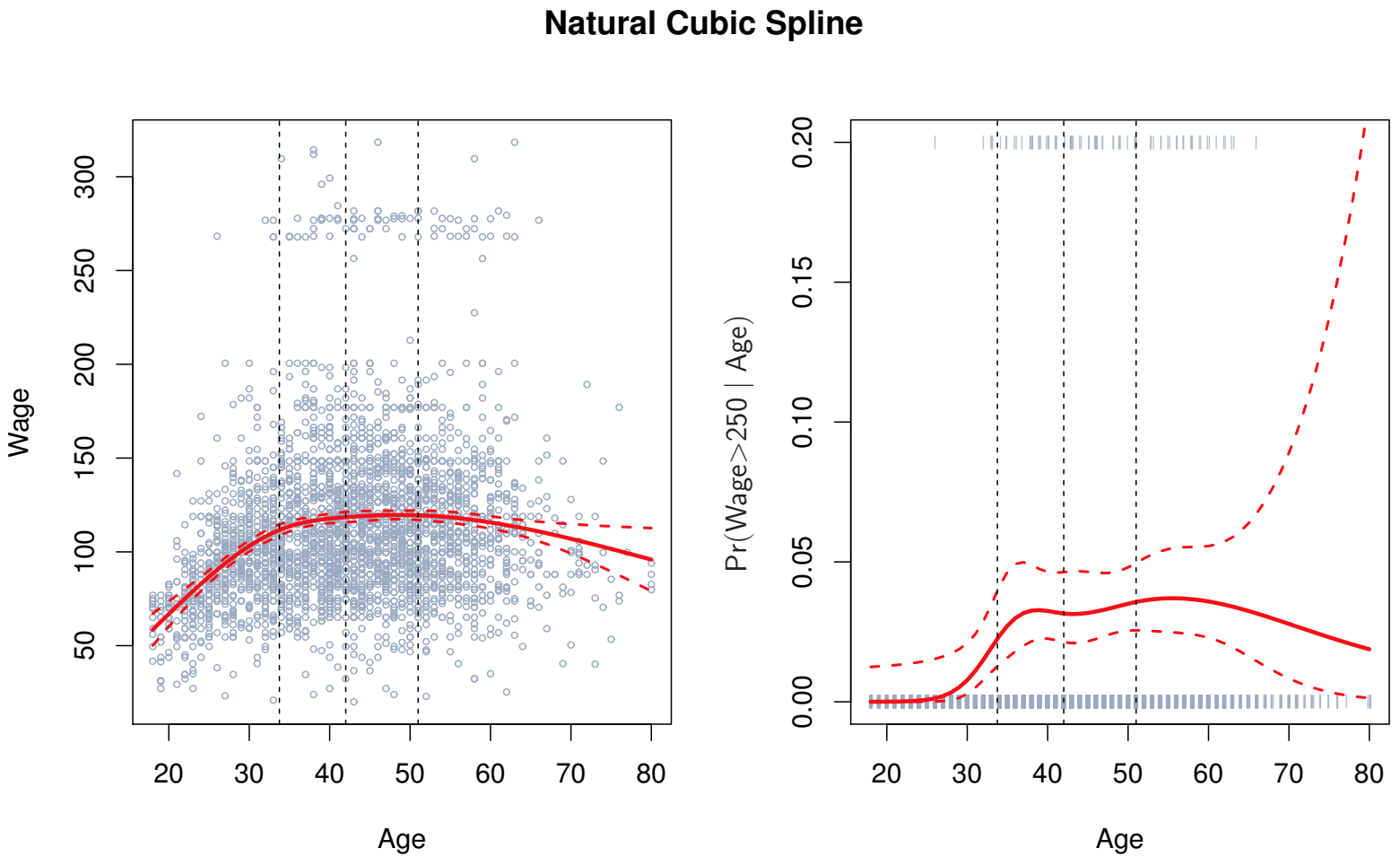

f(x)
1.0
1.2
1.4
1.6
0.0
0.2
0.4
0.6
0.8
1.0
x
b(x)
0.0
0.2
0.4
0.0
0.2
0.4
0.6
0.8
1.0
x

Natural Cubic Splines

A natural cubic spline extrapolates linearly beyond the boundary knots. This adds $4 = 2 \times 2$ extra constraints, and allows us to put more internal knots for the same degrees of freedom as a regular cubic spline.

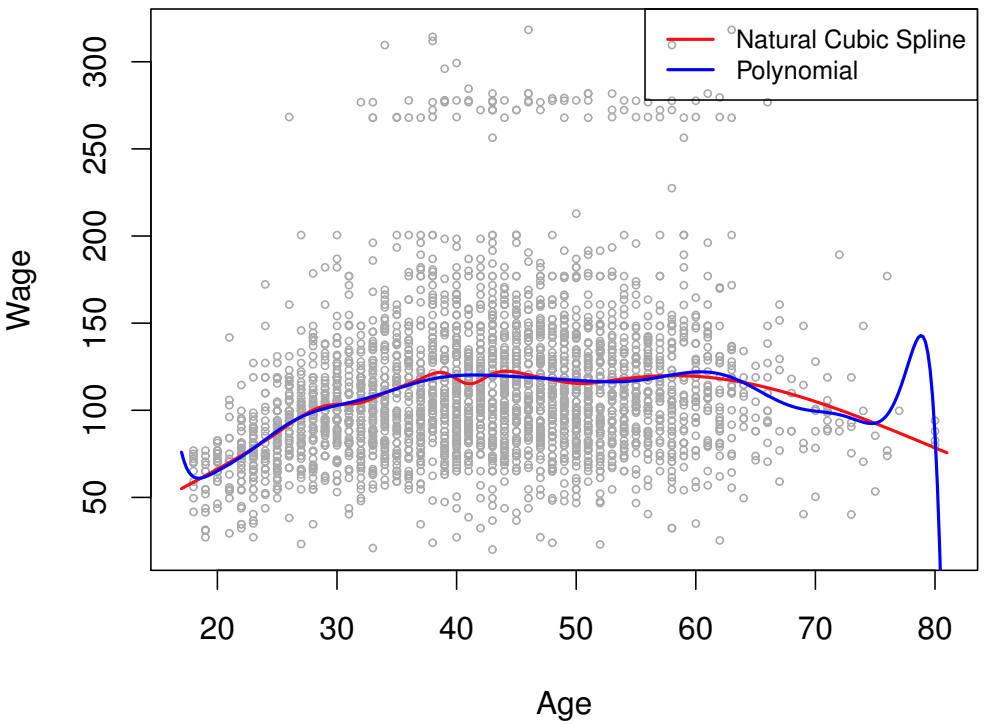


Fitting splines in R is easy: `bs(x, ...)` for any degree splines, and `ns(x, ...)` for natural cubic splines, in package `splines`.



Knot placement

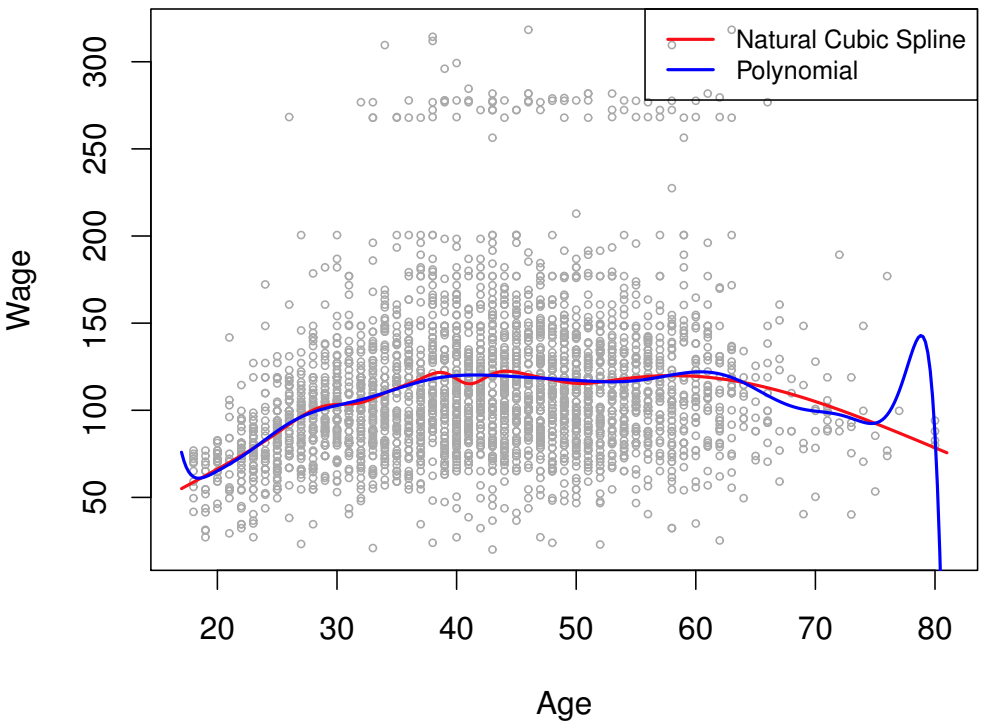
- One strategy is to decide K, the number of knots, and then place them at appropriate quantiles of the observed X.
- A cubic spline with K knots has $K + 4$ parameters or degrees of freedom.
- A natural spline with K knots has K degrees of freedom.



Comparison of a degree-14 polynomial and a natural cubic spline, each with 15df.

Knot placement

- One strategy is to decide K, the number of knots, and then place them at appropriate quantiles of the observed X.
- A cubic spline with K knots has $K + 4$ parameters or degrees of freedom.
- A natural spline with K knots has K degrees of freedom.



Comparison of a degree-14 polynomial and a natural cubic spline, each with 15df.

```
ns(age, df=14)
poly(age, deg=14)
```

More for spline

- Simple linear model:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

X matrix:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}.$$

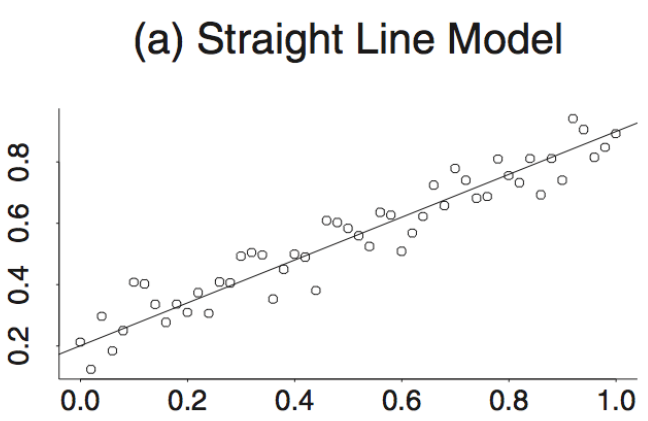
- quadratic model (polynomial) :

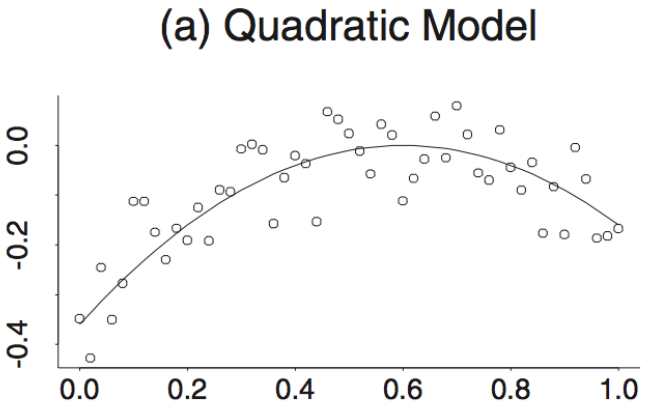
$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \epsilon_i$$

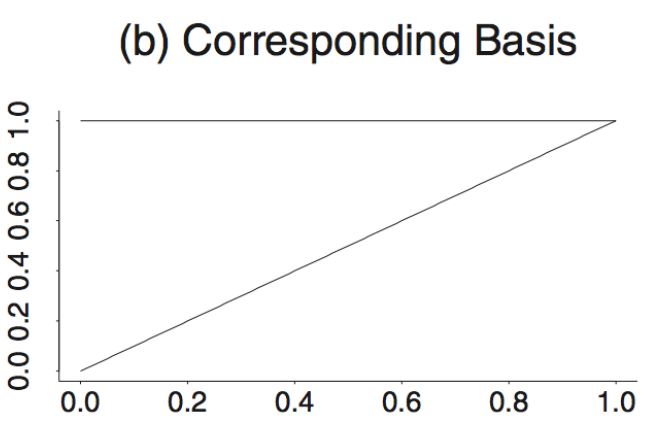
X matrix:

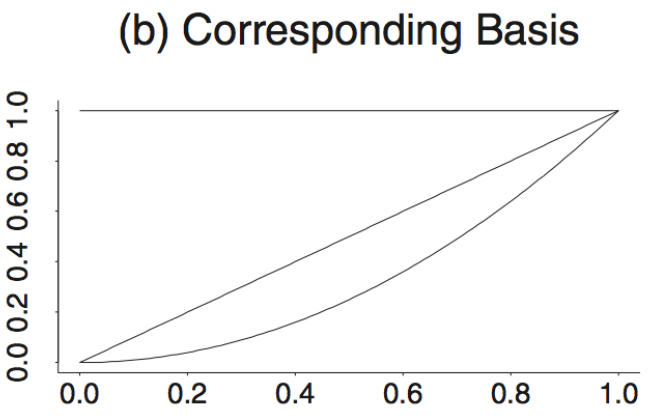
$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix},$$

More for spline









More for spline

- broken stick model:

$$y_i = \beta_0 + \beta_1 x_i + \beta_{11}(x_i - 0.6)_+ + \epsilon_i$$

- whip model (polynomial) :

$$y_i = \beta_0 + \beta_1 x_i + \beta_{11}(x_i - 0.5)_+ + \beta_{12}(x_i - 0.55)_+ \cdots + \beta_{1k}(x_i - 0.95)_+ + \epsilon_i$$

X matrix:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & (x_1 - 0.6)_+ \\ \vdots & \vdots & \vdots \\ 1 & x_n & (x_n - 0.6)_+ \end{bmatrix}. \quad \mathbf{X} = \begin{bmatrix} 1 & x_1 & (x_1 - 0.5)_+ & (x_1 - 0.55)_+ & \cdots & (x_1 - 0.95)_+ \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & (x_n - 0.5)_+ & (x_n - 0.55)_+ & \cdots & (x_n - 0.95)_+ \end{bmatrix}.$$

- Spline model for f

$$f(x) = \beta_0 + \beta_1 x + \sum_{k=1}^{K} b_k (x - k_k)_+$$

Smoothing Splines

This section is a little bit mathematical 

Consider this criterion for fitting a smooth function $g(x)$ to some data:

$$\underset{g \in \mathcal{S}}{\text{minimize}} \sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int g''(t)^2 dt$$

Smoothing Splines

This section is a little bit mathematical 

Consider this criterion for fitting a smooth function $g(x)$ to some data:

$$\underset{g \in \mathcal{S}}{\text{minimize}} \sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int g''(t)^2 dt$$

- The first term is RSS, and tries to make $g(x)$ match the data at each x_i.

Smoothing Splines

This section is a little bit mathematical 

Consider this criterion for fitting a smooth function $g(x)$ to some data:

$$\underset{g \in \mathcal{S}}{\text{minimize}} \sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int g''(t)^2 dt$$

- The first term is RSS, and tries to make $g(x)$ match the data at each x_i.
- The second term is a *roughness penalty* and controls how wiggly $g(x)$ is. It is modulated by the *tuning parameter* $\lambda \geq 0$.

Smoothing Splines

This section is a little bit mathematical 

Consider this criterion for fitting a smooth function $g(x)$ to some data:

$$\underset{g \in \mathcal{S}}{\text{minimize}} \sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int g''(t)^2 dt$$

- The first term is RSS, and tries to make $g(x)$ match the data at each x_i.
- The second term is a *roughness penalty* and controls how wiggly $g(x)$ is. It is modulated by the *tuning parameter* $\lambda \geq 0$.
  - The smaller λ, the more wiggly the function, eventually interpolating y_i when $\lambda = 0$.

Smoothing Splines

This section is a little bit mathematical 

Consider this criterion for fitting a smooth function $g(x)$ to some data:

$$\underset{g \in \mathcal{S}}{\text{minimize}} \sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int g''(t)^2 dt$$

- The first term is RSS, and tries to make $g(x)$ match the data at each x_i.
- The second term is a *roughness penalty* and controls how wiggly $g(x)$ is. It is modulated by the *tuning parameter* $\lambda \geq 0$.
  - The smaller λ, the more wiggly the function, eventually interpolating y_i when $\lambda = 0$.
  - As $\lambda \to \infty$, the function $g(x)$ becomes linear.

Smoothing Splines continued

The solution is a natural cubic spline, with a knot at every unique value of x_i. The roughness penalty still controls the roughness via λ.

Smoothing Splines continued

The solution is a natural cubic spline, with a knot at every unique value of x_i. The roughness penalty still controls the roughness via λ.

Some details

- Smoothing splines avoid the knot-selection issue, leaving a single λ to be chosen.

Smoothing Splines continued

The solution is a natural cubic spline, with a knot at every unique value of x_i. The roughness penalty still controls the roughness via λ.

Some details

- Smoothing splines avoid the knot-selection issue, leaving a single λ to be chosen.
- The algorithmic details are too complex to describe here. In R, the function `smooth.spline()` will fit a smoothing spline.

Smoothing Splines continued

The solution is a natural cubic spline, with a knot at every unique value of x_i. The roughness penalty still controls the roughness via λ.

Some details

- Smoothing splines avoid the knot-selection issue, leaving a single λ to be chosen.
- The algorithmic details are too complex to describe here. In R, the function `smooth.spline()` will fit a smoothing spline.
- The vector of n fitted values can be written as $\hat{\mathbf{g}}_\lambda = \mathbf{S}_\lambda \mathbf{y}$, where $\mathbf{S}_\lambda$ is a $n \times n$ matrix (determined by the x_i and λ).
- The *effective degrees of freedom* are given by

$$df_\lambda = \sum_{i=1}^{n} \{\mathbf{S}_\lambda\}_{ii}.$$

Smoothing Splines continued — choosing λ

- We can specify df rather than λ!

  In R: `smooth.spline(age, wage, df = 10)`

Smoothing Splines continued — choosing λ

- We can specify *df* rather than λ!

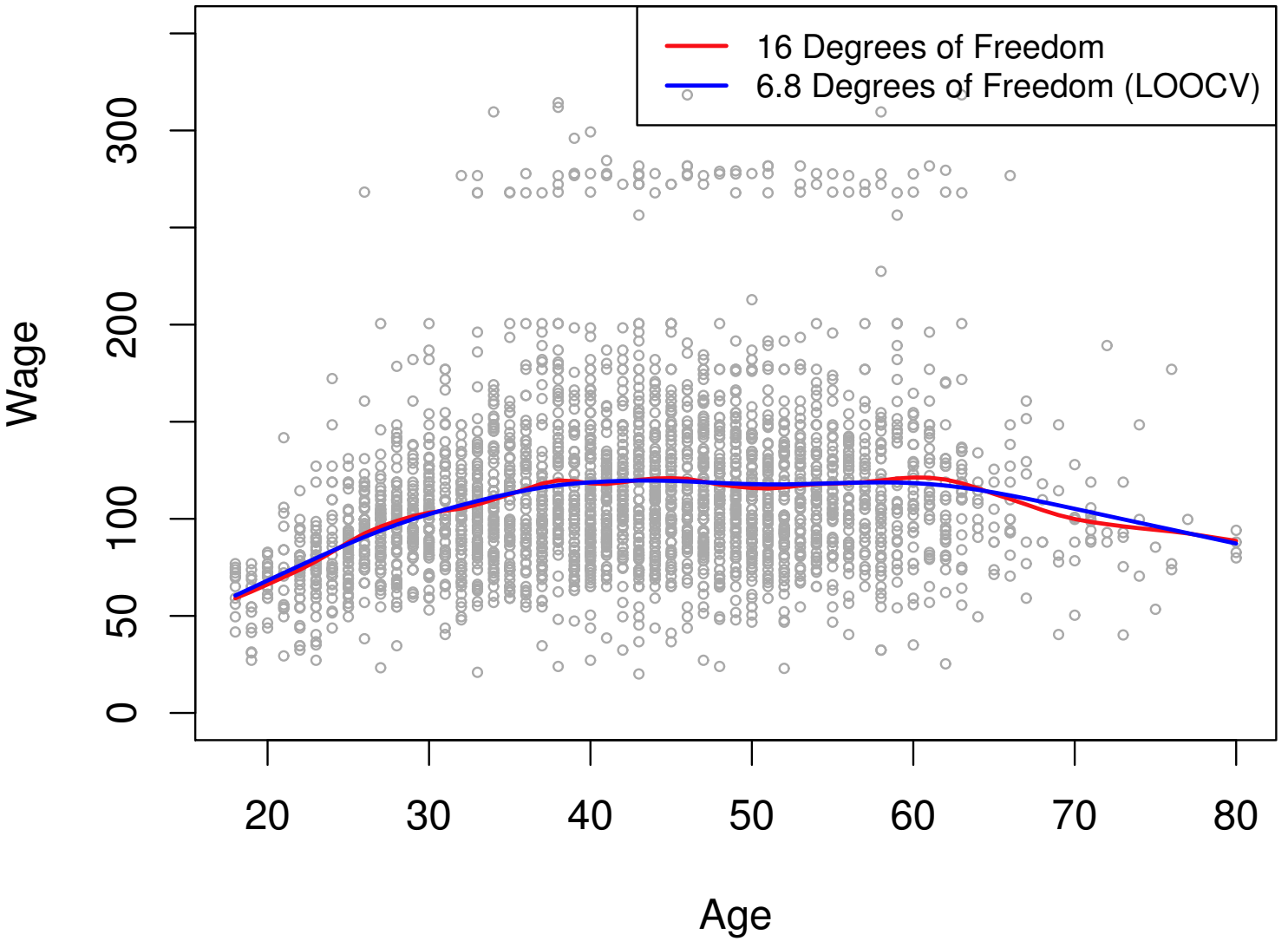
  In R: `smooth.spline(age, wage, df = 10)`

- The leave-one-out (LOO) cross-validated error is given by

$$\text{RSS}_{cv}(\lambda) = \sum_{i=1}^{n}(y_i - \hat{g}_\lambda^{(-i)}(x_i))^2 = \sum_{i=1}^{n}\left[\frac{y_i - \hat{g}_\lambda(x_i)}{1 - \{\mathbf{S}_\lambda\}_{ii}}\right]^2.$$

  In R: `smooth.spline(age, wage)`

Smoothing Spline



Penalized spline

- The ordinary least squares fit can be written as

$$\hat{y} = X\hat{\beta}, where \hat{\beta} min||y - X\beta||^2$$

  where $\beta = [\beta_0, \beta_1, \beta_{11}, \cdots, \beta_{1K}]^T$, with β_{1k} the coefficient of the kth knot.
- unconstrained estimation of the β_{1k} leads to a wiggly fit, constraints on the β_{1k}:
  (1) $max|\beta_{1k}| < C$, (2) $\sum |\beta_{1k}| < C$, (3) $\sum \beta_{1k}^2 < C$
- If we define the $(K+2) \times (K+2)$ matrix

$$\mathbf{D} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{2\times 2} & \mathbf{0}_{2\times K} \\ \mathbf{0}_{K\times 2} & \mathbf{I}_{K\times K} \end{bmatrix}$$

Penalized spline

- then our minimization problem can be written as

$$min||y - X\beta||^2 subject to \beta^T D\beta \leq C$$

- It can be shown, using a Lagrange multiplier argument, that this is equivalent to minimize

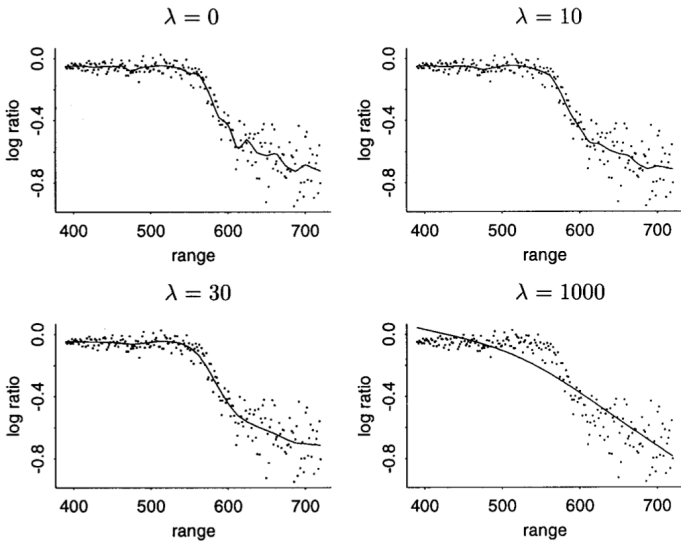
$$||y - X\beta||^2 + \lambda^2 \beta^T D\beta$$

- the solution

$$\hat{\beta}_\lambda = (X^T X + \lambda^2 D)^{-1} X^T y$$

where the term $\lambda^2 \beta^T D\beta$ is called a roughness penalty.

Penalized spline



Kernel smoothers

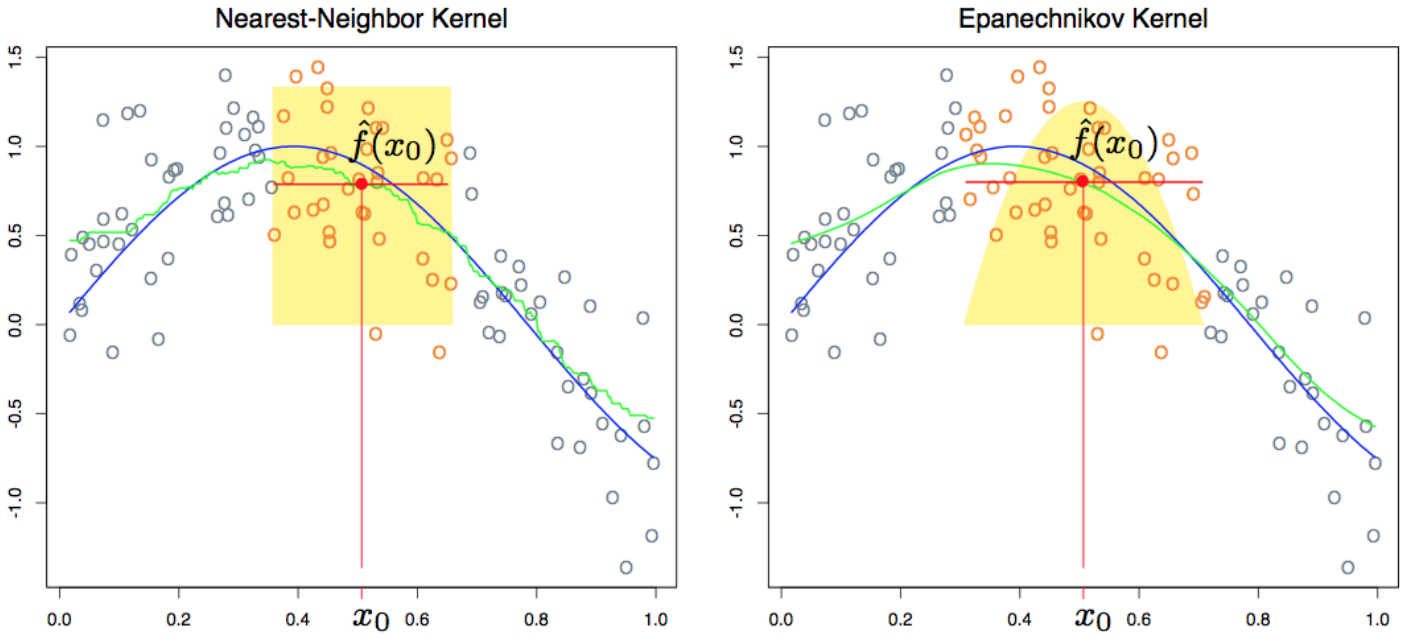


FIGURE 6.1. *In each panel* 100 *pairs* x_i, y_i *are generated at random from the blue curve with Gaussian errors:* $Y = \sin(4X) + \varepsilon$, $X \sim U[0,1]$, $\varepsilon \sim N(0, 1/3)$. *In the left panel the green curve is the result of a* 30*-nearest-neighbor running-mean smoother. The red point is the fitted constant* $\hat{f}(x_0)$, *and the red circles indicate those observations contributing to the fit at* x_0. *The solid yellow region indicates the weights assigned to observations. In the right panel, the green curve is the kernel-weighted average, using an Epanechnikov kernel with (half) window width* $\lambda = 0.2$.

Kernel smoothers

- k-nearest neighbor average

$$\hat{f}(x) = Ave(y_i | x_i \in N_k(x))$$

- Nadaraya-Watson kernel-weighted average

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

with the *Epanechnikov* quadratic kernel

$$K_\lambda(x_0, x) = D\left(\frac{|x - x_0|}{\lambda}\right),$$

with

$$D(t) = \begin{cases} \frac{3}{4}(1 - t^2) & \text{if } |t| \leq 1; \\ 0 & \text{otherwise.} \end{cases}$$

Kernel smoothers

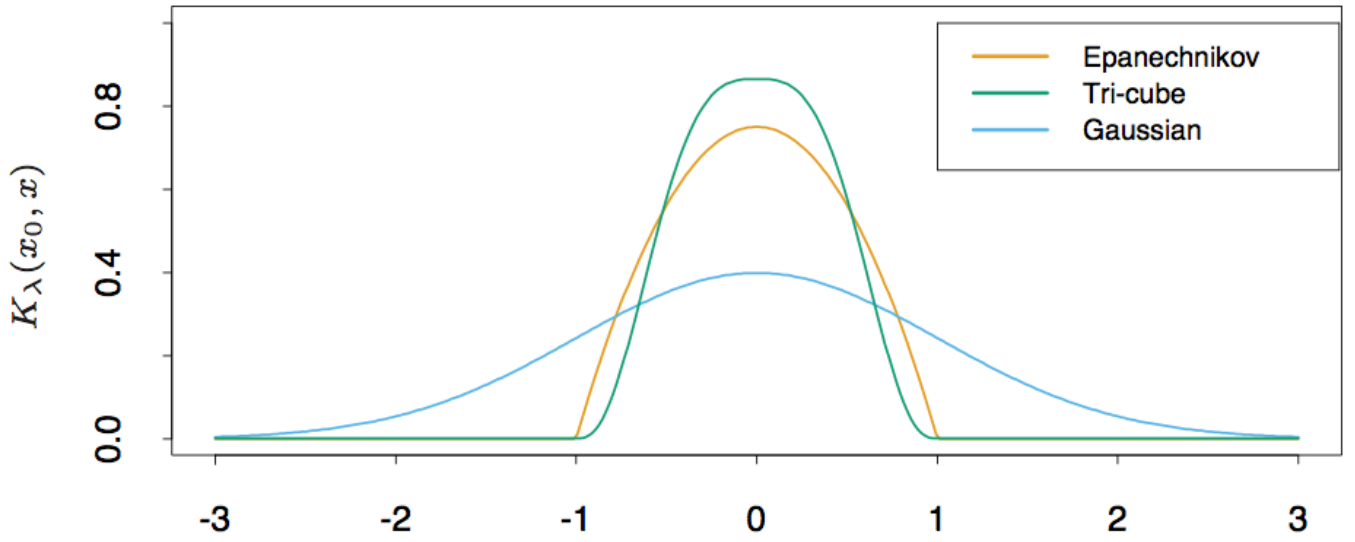
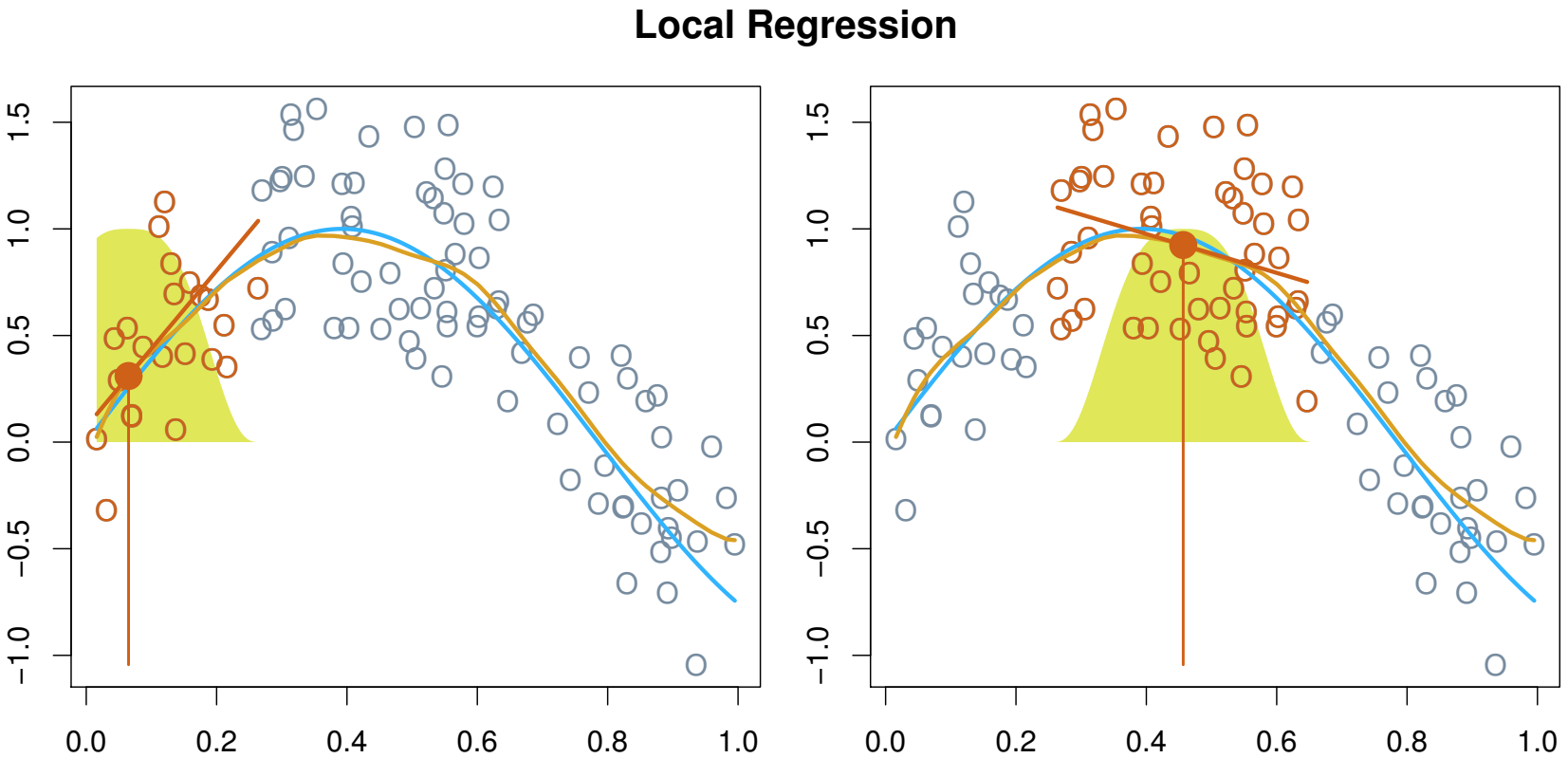


FIGURE 6.2. *A comparison of three popular kernels for local smoothing. Each has been calibrated to integrate to* 1. *The tri-cube kernel is compact and has two continuous derivatives at the boundary of its support, while the Epanechnikov kernel has none. The Gaussian kernel is continuously differentiable, but has infinite support.*

Local Regression



With a sliding weight function, we fit separate linear fits over the range of X by weighted least squares.
See text for more details, and `loess()` function in R.

local regression

Algorithm 7.1 *Local Regression At* $X = x_0$

1. Gather the fraction $s = k/n$ of training points whose x_i are closest to x_0.

2. Assign a weight $K_{i0} = K(x_i, x_0)$ to each point in this neighborhood, so that the point furthest from x_0 has weight zero, and the closest has the highest weight. All but these k nearest neighbors get weight zero.

3. Fit a *weighted least squares regression* of the y_i on the x_i using the aforementioned weights, by finding $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize

$$\sum_{i=1}^{n} K_{i0}(y_i - \beta_0 - \beta_1 x_i)^2. \tag{7.14}$$

4. The fitted value at x_0 is given by $\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0$.

local regression

- locally weighted regression solves a separate weighted least squares prob- lem at each target point x_0:

$$\min_{\alpha(x_0),\beta(x_0)} \sum_{i=1}^{N} K_\lambda(x_0, x_i) \left[y_i - \alpha(x_0) - \beta(x_0)x_i\right]^2 .$$

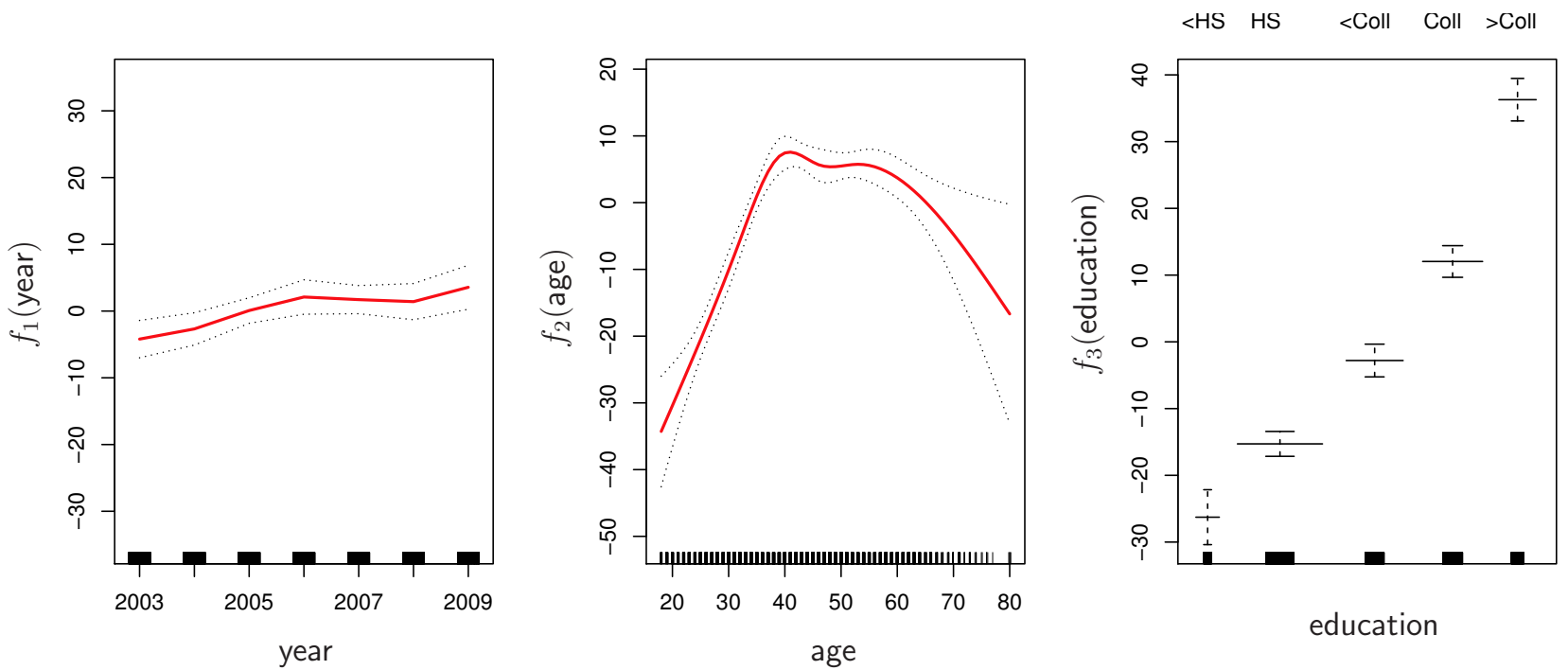
- the estimate is then $\hat{f}(x_0) = \alpha(\hat{x}_0) + \beta(\hat{x}_0)x_0$
- define the vector function $b(x)^T = (1, x)$, let B be the $N \times 2$ rgression matrix with ith row $b(x_i)^T$, and $W(x_0)$ the $N \times N$ diagonal matrix with ith diagonal element $K_\lambda(x_0, x_i)$, then

$$\begin{aligned} \hat{f}(x_0) &= b(x_0)^T(\mathbf{B}^T\mathbf{W}(x_0)\mathbf{B})^{-1}\mathbf{B}^T\mathbf{W}(x_0)\mathbf{y} \\ &= \sum_{i=1}^{N} l_i(x_0)y_i. \end{aligned}$$

Generalized Additive Models

Allows for flexible nonlinearities in several variables, but retains the additive structure of linear models.

$$y_i = \beta_0 + f_1(x_{i1}) + f_2(x_{i2}) + \cdots + f_p(x_{ip}) + \epsilon_i.$$



GAM details

- Can fit a GAM simply using, e.g. natural splines:

$$\texttt{lm}(\texttt{wage} \sim \texttt{ns}(\texttt{year}, \texttt{df} = 5) + \texttt{ns}(\texttt{age}, \texttt{df} = 5) + \texttt{education})$$

GAM details

- Can fit a GAM simply using, e.g. natural splines:

  $$\texttt{lm}(\texttt{wage} \sim \texttt{ns}(\texttt{year}, \texttt{df} = 5) + \texttt{ns}(\texttt{age}, \texttt{df} = 5) + \texttt{education})$$

- Coefficients not that interesting; fitted functions are. The previous plot was produced using `plot.gam`.

GAM details

- Can fit a GAM simply using, e.g. natural splines:

$$\texttt{lm}(\texttt{wage} \sim \texttt{ns}(\texttt{year}, \texttt{df} = 5) + \texttt{ns}(\texttt{age}, \texttt{df} = 5) + \texttt{education})$$

- Coefficients not that interesting; fitted functions are. The previous plot was produced using `plot.gam`.
- Can mix terms — some linear, some nonlinear — and use `anova()` to compare models.

GAM details

- Can fit a GAM simply using, e.g. natural splines:

$$\texttt{lm}(\texttt{wage} \sim \texttt{ns}(\texttt{year}, \texttt{df} = 5) + \texttt{ns}(\texttt{age}, \texttt{df} = 5) + \texttt{education})$$

- Coefficients not that interesting; fitted functions are. The previous plot was produced using `plot.gam`.
- Can mix terms — some linear, some nonlinear — and use `anova()` to compare models.
- Can use smoothing splines or local regression as well:

$$\texttt{gam}(\texttt{wage} \sim \texttt{s}(\texttt{year}, \texttt{df} = 5) + \texttt{lo}(\texttt{age}, \texttt{span} = .5) + \texttt{education})$$

GAM details

- Can fit a GAM simply using, e.g. natural splines:

  $$\texttt{lm}(\texttt{wage} \sim \texttt{ns}(\texttt{year}, \texttt{df} = 5) + \texttt{ns}(\texttt{age}, \texttt{df} = 5) + \texttt{education})$$

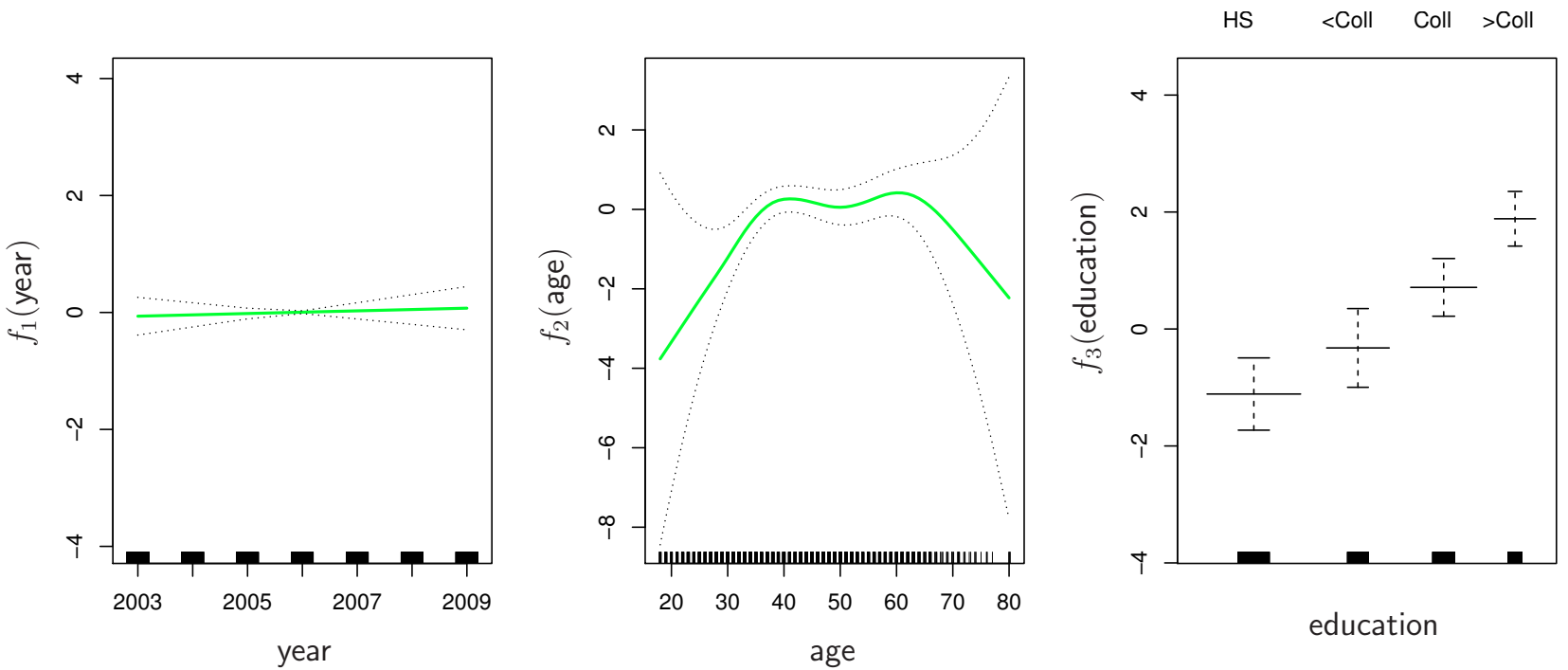
- Coefficients not that interesting; fitted functions are. The previous plot was produced using `plot.gam`.
- Can mix terms — some linear, some nonlinear — and use `anova()` to compare models.
- Can use smoothing splines or local regression as well:

  $$\texttt{gam}(\texttt{wage} \sim \texttt{s}(\texttt{year}, \texttt{df} = 5) + \texttt{lo}(\texttt{age}, \texttt{span} = .5) + \texttt{education})$$

- GAMs are additive, although low-order interactions can be included in a natural way using, e.g. bivariate smoothers or interactions of the form `ns(age,df=5):ns(year,df=5)`.

GAMs for classification

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + f_1(X_1) + f_2(X_2) + \cdots + f_p(X_p).$$



`gam(I(wage > 250) ~ year + s(age, df = 5) + education, family = binomial)`

GAM

Algorithm 9.1 *The Backfitting Algorithm for Additive Models.*

1. Initialize: $\hat{\alpha} = \frac{1}{N}\sum_1^N y_i$, $\hat{f}_j \equiv 0, \forall i, j$.

2. Cycle: $j = 1, 2, \ldots, p, \ldots, 1, 2, \ldots, p, \ldots,$

$$\hat{f}_j \leftarrow \mathcal{S}_j\left[\{y_i - \hat{\alpha} - \sum_{k \neq j} \hat{f}_k(x_{ik})\}_1^N\right],$$

$$\hat{f}_j \leftarrow \hat{f}_j - \frac{1}{N}\sum_{i=1}^N \hat{f}_j(x_{ij}).$$

until the functions $\hat{f}_j$ change less than a prespecified threshold.

GAM

Algorithm 9.2 *Local Scoring Algorithm for the Additive Logistic Regression Model.*

1. Compute starting values: $\hat{\alpha} = \log[\bar{y}/(1-\bar{y})]$, where $\bar{y} = \text{ave}(y_i)$, the sample proportion of ones, and set $\hat{f}_j \equiv 0 \; \forall j$.

2. Define $\hat{\eta}_i = \hat{\alpha} + \sum_j \hat{f}_j(x_{ij})$ and $\hat{p}_i = 1/[1+\exp(-\hat{\eta}_i)]$.

   Iterate:

   (a) Construct the working target variable

   $$z_i = \hat{\eta}_i + \frac{(y_i - \hat{p}_i)}{\hat{p}_i(1-\hat{p}_i)}.$$

   (b) Construct weights $w_i = \hat{p}_i(1-\hat{p}_i)$

   (c) Fit an additive model to the targets z_i with weights w_i, using a weighted backfitting algorithm. This gives new estimates $\hat{\alpha}, \hat{f}_j, \; \forall j$

3. Continue step 2. until the change in the functions falls below a prespecified threshold.