STA 314: Statistical Methods for Machine Learning I

Lecture - Moving beyond linearity

Xin Bing

Department of Statistical Sciences
University of Toronto

Review on regularized linear regression

- OLS that uses p features based on n data points cannot perform well when p is large relative to n.
- Regularized approach such as Lasso and Ridge can have better performance
  - Reduce variance
  - Pay extra bias
- The benefit of regularization could be significant if the true model coefficients are either small or sparse.
  - If only $s \ll p$ features are predictive, we should only fit OLS by using these s features.

Linearity in features vs in parameters

The linearity assumption in the feature space (in X) is almost always an approximation, and sometimes a poor one.

Example

Consider $X = (X_1, X_2)$.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon.$$

What about the following one?

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 X^2 + \beta_5 (X_1 X_2) + \epsilon.$$

Also a linear model in $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_4)$ but not in $X = (X_1, X_2)$.
Implication: can deploy

- OLS
- Subset selection
- Regularized linear regression

Moving Beyond Linearity

We consider the following extensions to relax the linearity assumption (in the feature space).

- Univariate case ($p = 1$):
  - Polynomial regression
  - Step functions
  - Regression splines
- Multivariate case ($p > 1$):
  - Local regression
  - Generalized additive models

Polynomial Regression

- The **polynomial regression** assumes

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_d x_i^d + \epsilon_i,$$

where ϵ_i is the error term and $x_i \in \mathcal{X}$.

- Can be fitted by the OLS approach, the ridge and the lasso.

- Coefficients themselves are not interpretable; we are more interested in the trend of the fitted function

$$\hat{f}(x) = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 + \cdots + \hat{\beta}_d x^d, \qquad \forall x \in \mathcal{X}.$$

Polynomial Regression

- The degree d in practice is typically no greater than 4, and can be chosen via cross-validation.

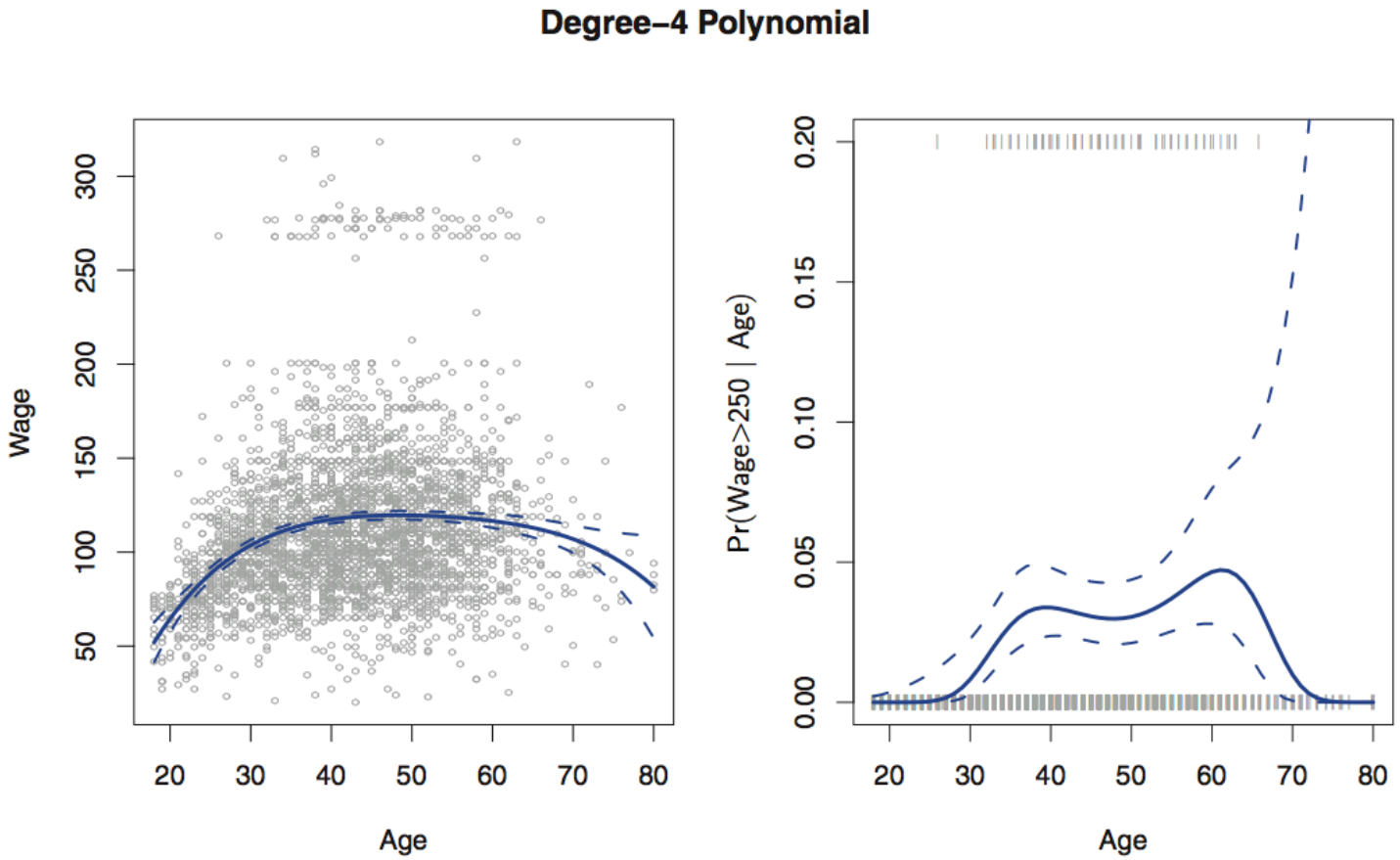
- The polynomial regression can be used for classification as well.
  - For instance, in the logistic regression,

$$\text{logit}\left(\mathbb{P}(Y_i = 1 \mid X_i = x_i)\right) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_d x_i^d.$$

  - Can be fit by maximizing the likelihood.

- However, polynomials have notorious tail behavior – very bad for extrapolation.

The Wage Data



Left: The solid blue curve is a degree-4 polynomial of wage as a function of age, fit by the OLS. The dotted curves are estimated 95 % confidence intervals.

Right: Model the binary event $1\{\text{wage} > 250\}$ by logistic regression, with a degree-4 polynomial.

Step Functions

- The polynomial regression imposes a global structure on the non-linearity of X.
- The **step function** approach avoids such a global structure by breaking the range of X into bins.
- For pre-specified K cut points $c_1 \leq c_2 \leq \cdots \leq c_{K-1} \leq c_K$, define

$$
\begin{aligned}
C_0(X) &= 1\{X < c_1\}, \\
C_1(X) &= 1\{c_1 \leq X < c_2\}, \\
&\vdots \\
C_K(X) &= 1\{c_K \leq X\}.
\end{aligned}
$$

$C_0(X), \ldots, C_K(X)$ are in fact $(K+1)$ dummy variables, and they sum up to 1.

Step Functions

- Step function approach assumes

$$y_i = \beta_0 + \beta_1 C_1(x_i) + \beta_2 C_2(x_i) + ... + \beta_K C_K(x_i) + \epsilon_i,$$

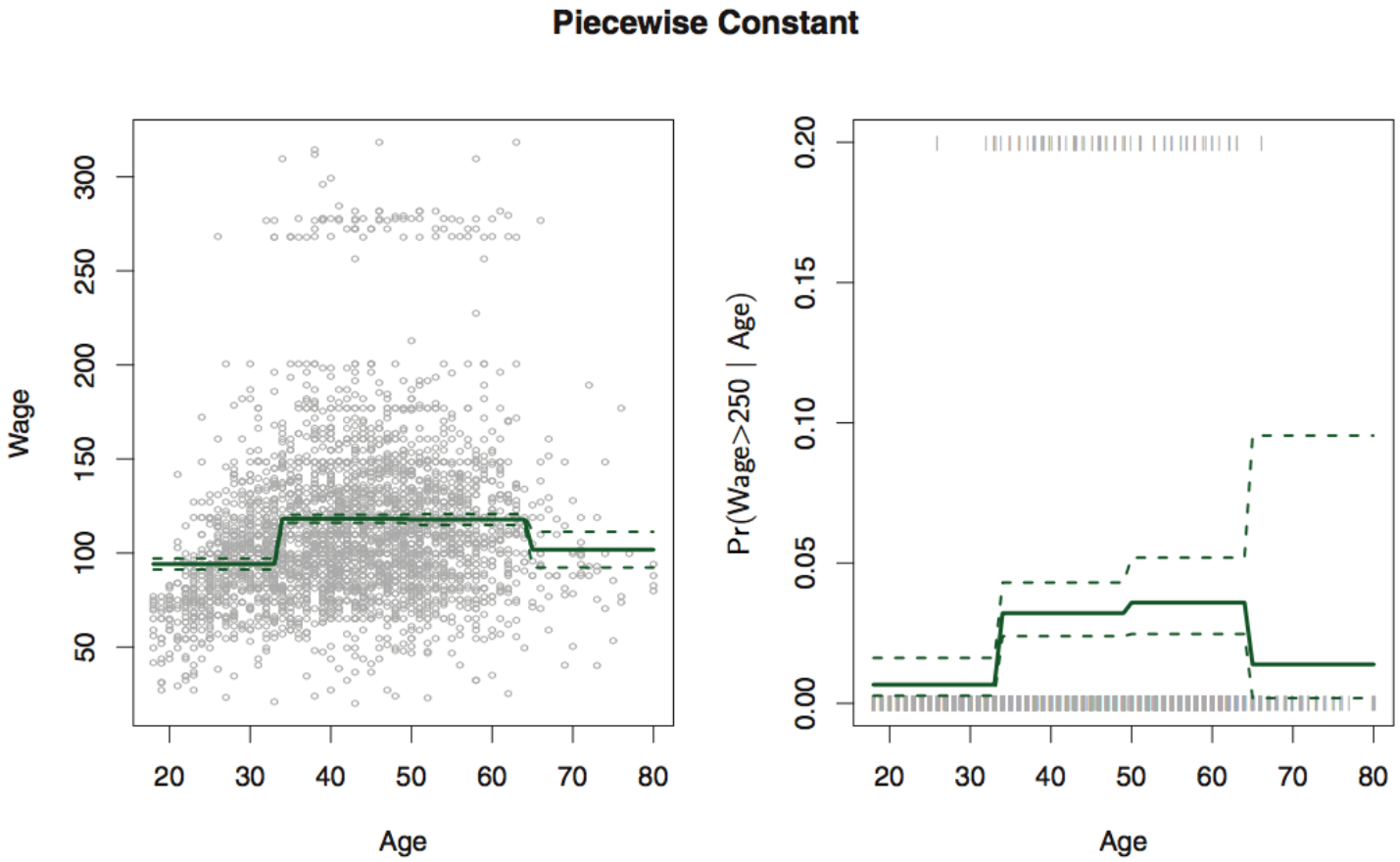
  where ϵ_i is the error term.[1]

- Can be fitted by the OLS and shrinkage regression.

- **Interpretation**: β_j represents the average change in the response Y for $c_j \leq X < c_{j+1}$ relative to $X < c_1$.

[1] We don't need $C_0(x_i)$ in the model when we also have the intercept term β_0.

The Wage Data



Left: The solid blue curve is a step function of wage as a function of age, fit by least squares. The dotted curves indicate an estimated 95 % confidence interval.

Right: Model the binary event $1\{\text{wage} > 250\}$ by logistic regression, with the step function.

Pros and Cons of Step Function

- The step function approach is widely used in biostatistics and epidemiology among other areas:
  - the model is easy to fit
  - the regression coefficient has a natural interpretation

- However, piecewise-constant functions can miss the trend of the true relationship between Y and X. The choice of cut points can be difficult to specify.

- How about combining polynomial and step function?

Piecewise Polynomials

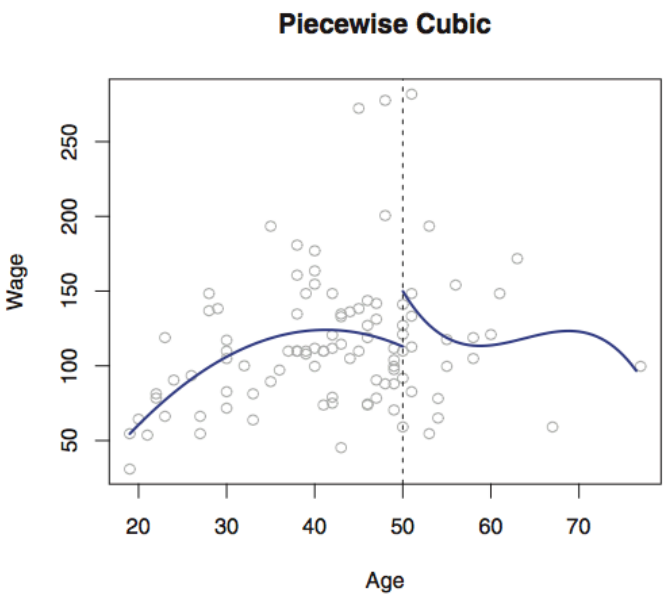
- Instead of a single polynomial in X over its whole domain, we can use different polynomials in different regions:

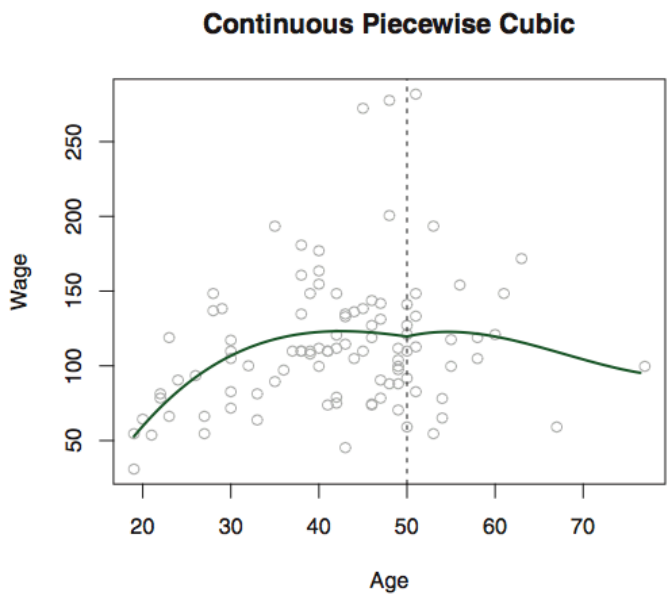
$$y_i = \begin{cases} \beta_{01} + \beta_{11} x_i + \beta_{21} x_i^2 + \beta_{31} x_i^3 + \epsilon_i & \text{if } x_i < c; \\ \beta_{02} + \beta_{12} x_i + \beta_{22} x_i^2 + \beta_{32} x_i^3 + \epsilon_i & \text{if } x_i \geq c. \end{cases}$$

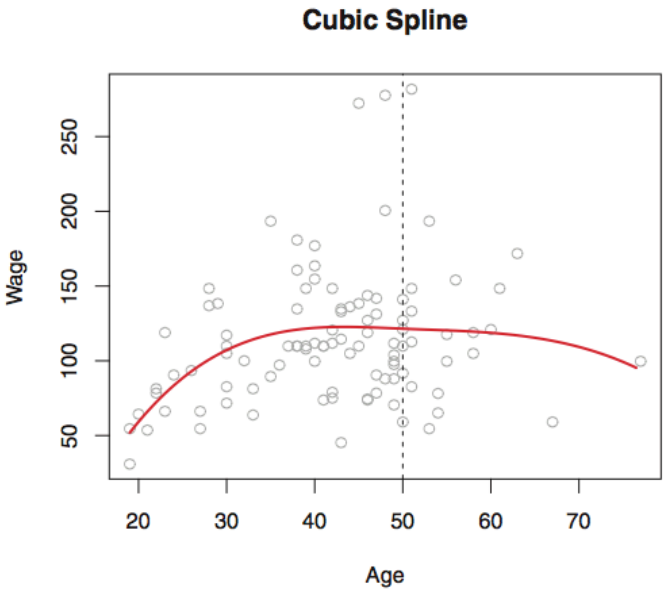
- The cut point c is called **knot**. Using more knots leads to a more flexible piecewise polynomial.

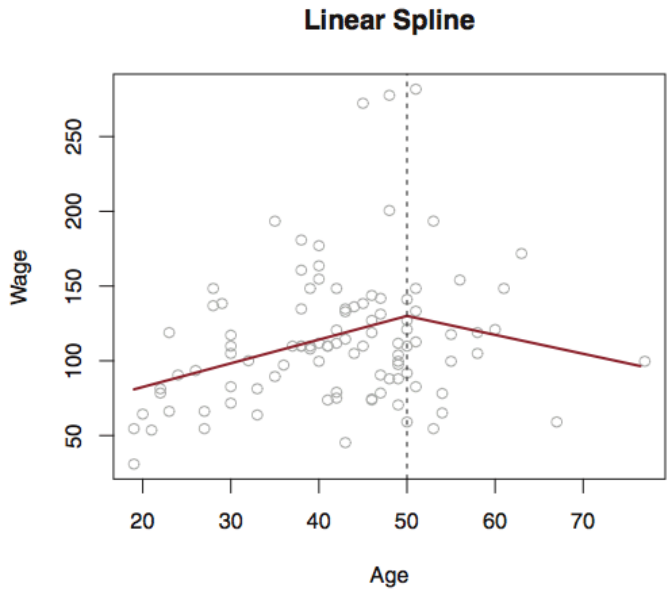
- In general, if we place K different knots throughout the range of X, then we will end up fitting $(K + 1)$ different cubic polynomials.

The Wage Data









Regression splines

- Better to add constraints to polynomials at the knots for:
  - continuity: equal function values
  - smoothness: equal first and second order derivatives
  - higher order derivatives

- The constrained polynomials are called **splines**. A degree-d spline contains piecewise degree-d polynomials, with continuity in derivatives up to degree $(d-1)$ at each knot.

- How can we construct the degree-d spline?

Linear Splines

- A **linear spline** has piecewise linear functions continuous at each knot. That is, with knots at $\xi_1 < \xi_2 < \cdots < \xi_K$,

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 (x_i - \xi_1)_+ \cdots + \beta_{K+1} (x_i - \xi_K)_+ + \epsilon_i,$$

where, for each $1 \leq k \leq K$,

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

- Interpretation of β_1: the averaged increase of Y associated with one unit of X for $X < \xi_1$.

Basis Functions

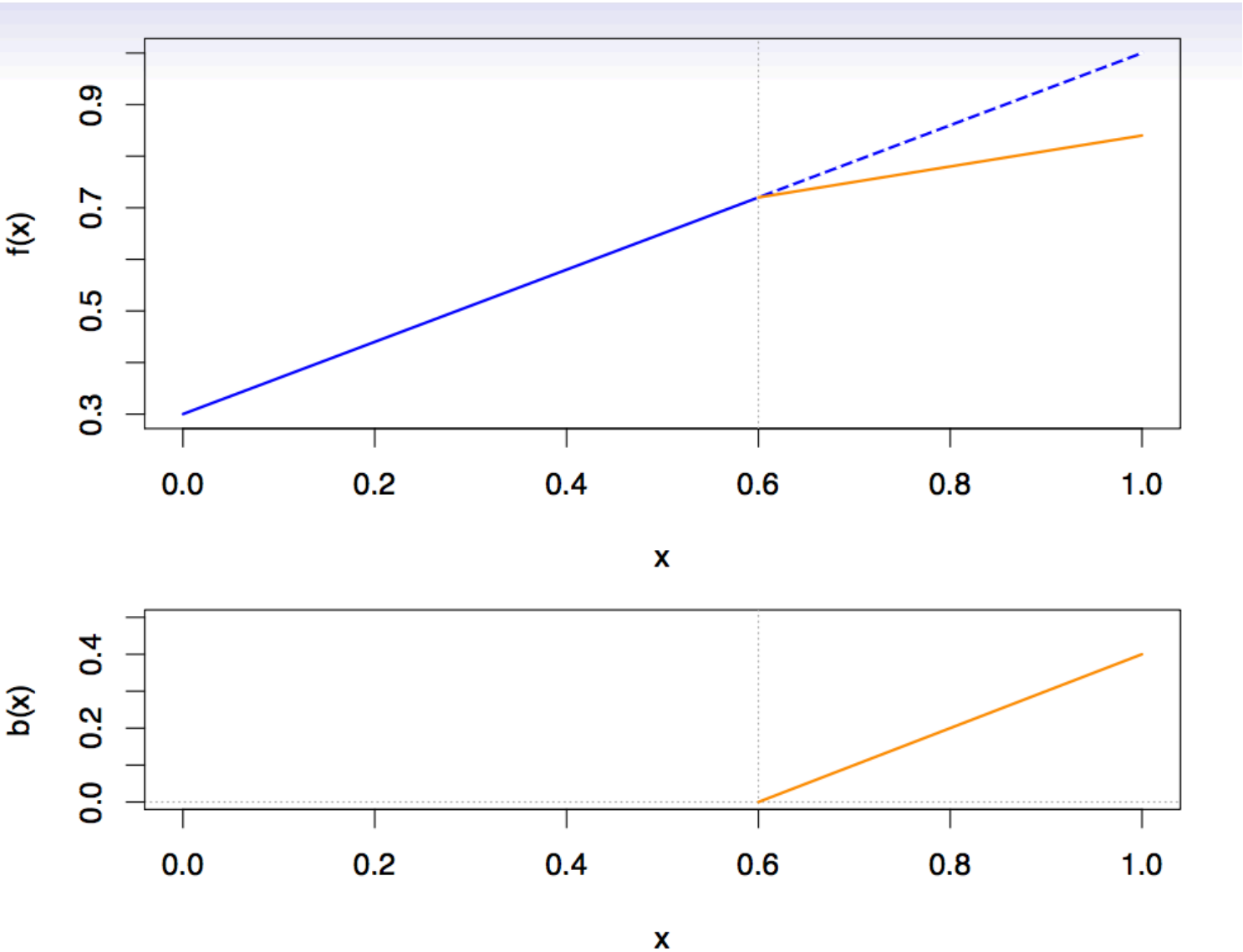
A basis representation:

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_K b_K(x_i) + \epsilon_i,$$

where $b_k(\cdot)$ for $1 \leq k \leq K$ are **basis functions**:

- Polynomials:
$$b_k(x_i) = x_i^k.$$
- Step Functions:
$$b_k(x_i) = C_k(x_i).$$
- Linear splines:
$$b_1(x_i) = x_i, \quad b_k(x_i) = (x_i - \xi_{k-1})_+, \quad k = 1, \ldots, K,$$

Linear Splines



Cubic Splines

- A **cubic spline** has piecewise cubic polynomials with continuous derivatives up to order 2 at each knot.

- That is, with K knots at $\xi_1 < \xi_2 < \cdots < \xi_K$,

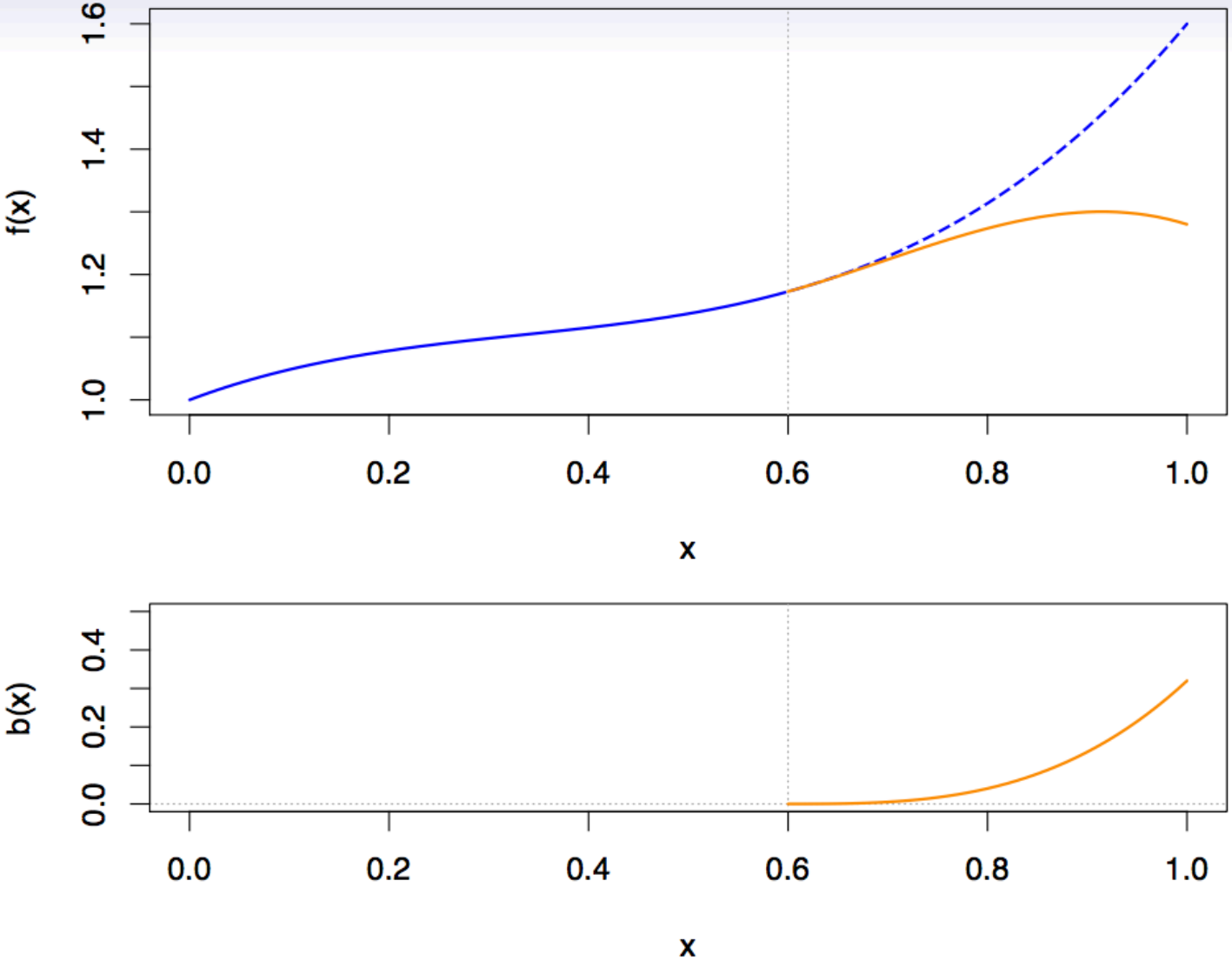
$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \ldots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i,$$

where $b_k(\cdot)$ are basis functions

$$b_1(x_i) = x_i, \quad b_2(x_i) = x_i^2, \quad b_3(x_i) = x_i^3,$$

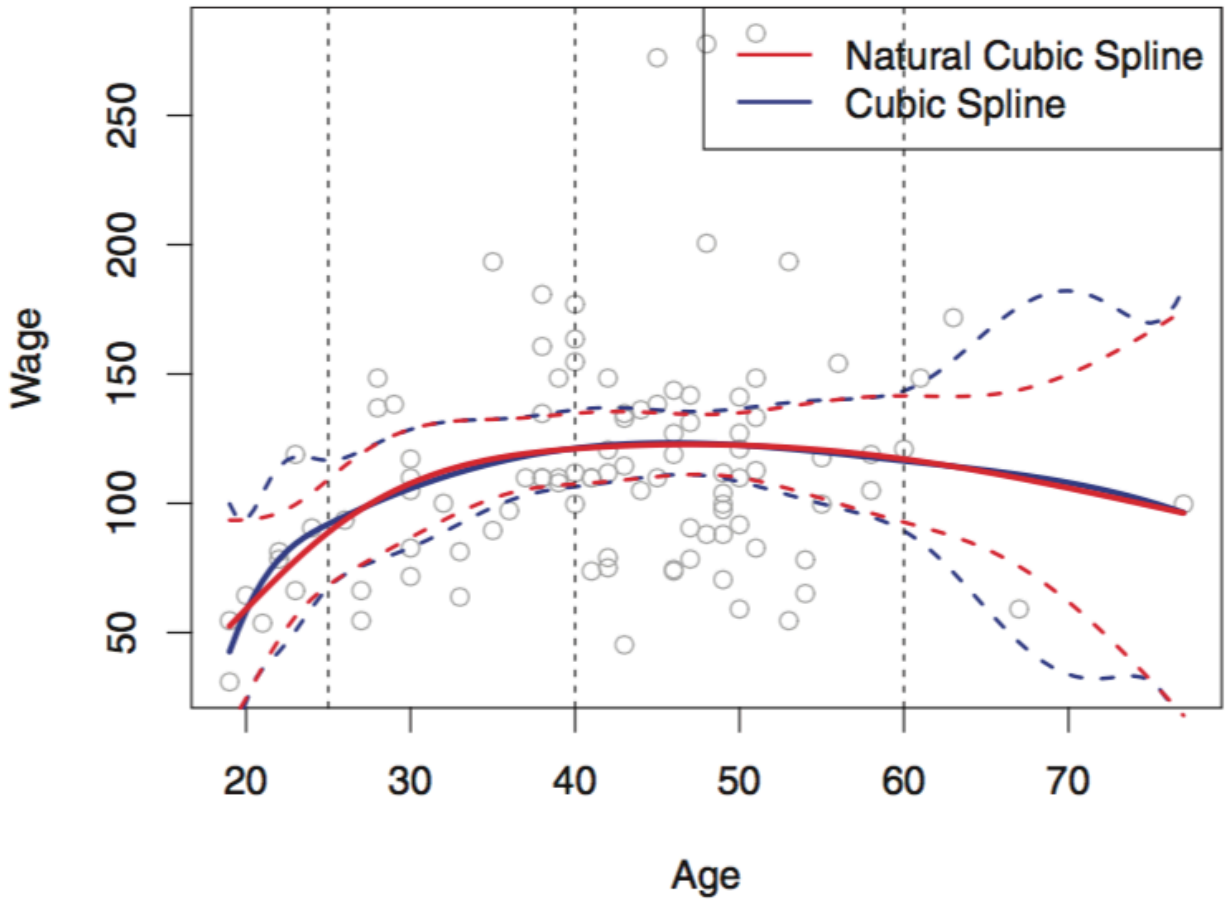
$$b_{k+3}(x_i) = (x_i - \xi_k)_+^3, \qquad k = 1, \ldots, K.$$

Cubic Splines



Natural Splines

A natural spline is a regression spline with additional boundary constraints: the function is required to be linear at the boundary.



More on splines

- Choosing the number and locations of the knots
  - Typically, we place K knots at certain quantiles of the data or place on the range of X with equal space. Oftentimes, the placement of knots is not very crucial.
  - We use cross-validation to choose K.
- Polynomial regressions and step functions are special cases of splines.
- Another variant: smoothing spline (ISLR 7.5).

Move beyond linearity

What about $p > 1$?

- Splines can be used for $p < 4$ (number of basis functions grows exponentially in p)
- Local approach for $p < 4$
  - nearest neighor approach
  - local regression
- Generalized Additive Models (GAM) for large p.

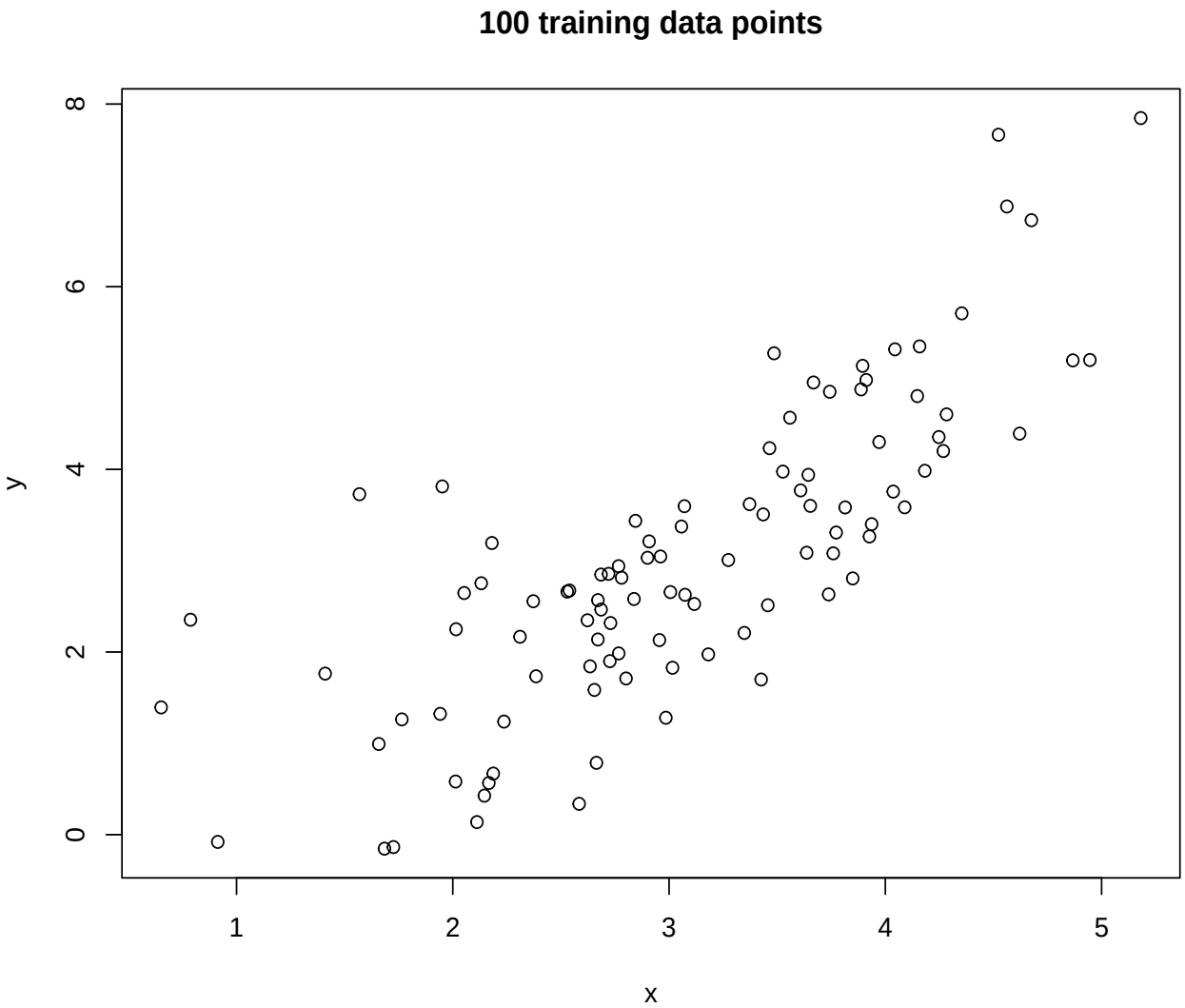
Local approach

Example (k nearest neighbours)

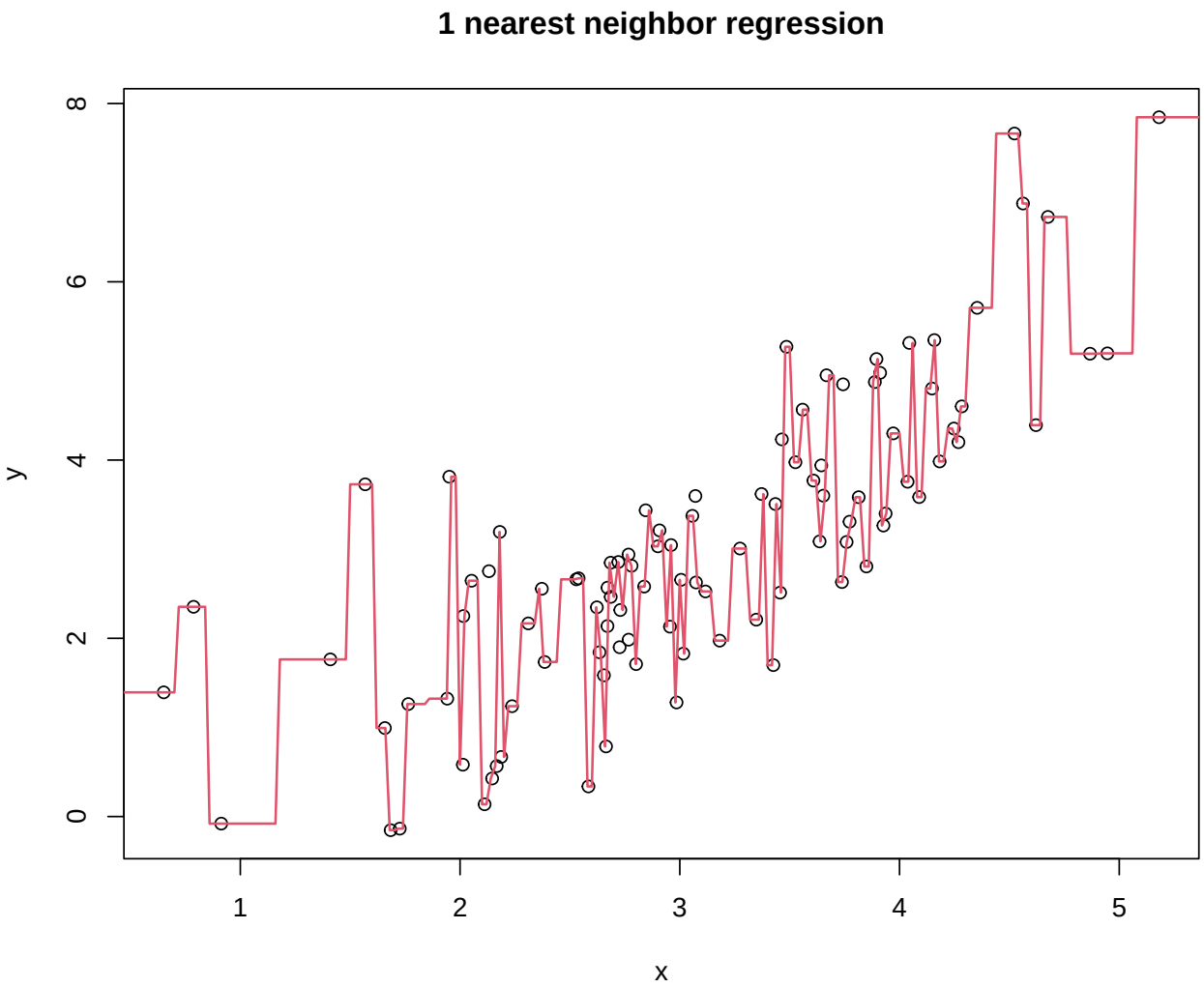
- Pick the number of neighbors $k \in \{1, \ldots, n\}$
- To predict at $X = x_0$, find the k neareast neighbors of x_0 among $\{x_1, \ldots, x_n\}$, collected in $\mathcal{N}_k(x_0)$
- Predict by using the local average

$$\hat{f}(x_0) = \frac{1}{k} \sum_{i \in \{1,\ldots,n\}: x_i \in \mathcal{N}_k(x_0)} y_i$$

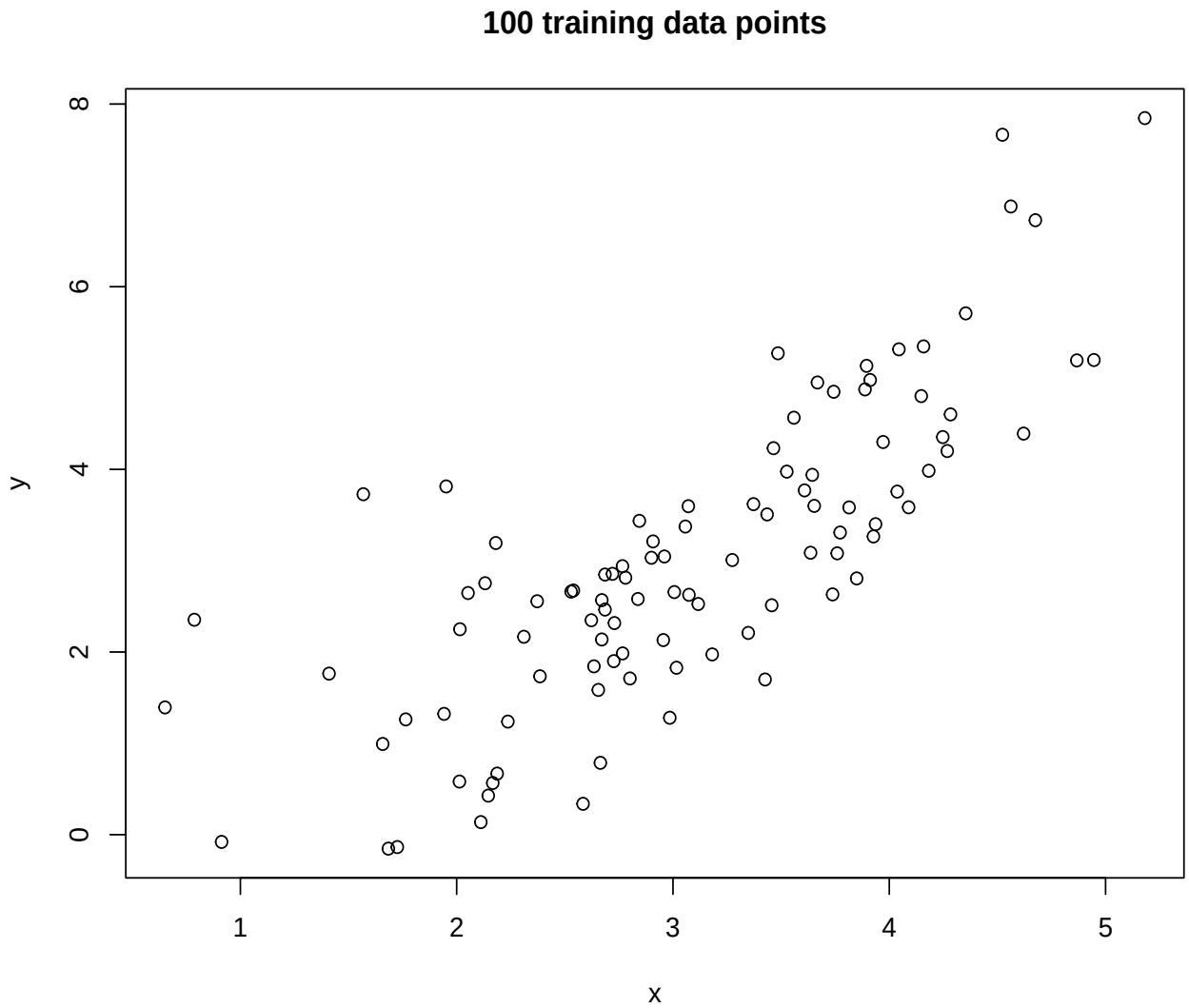
k nearest neighbors: the role of k



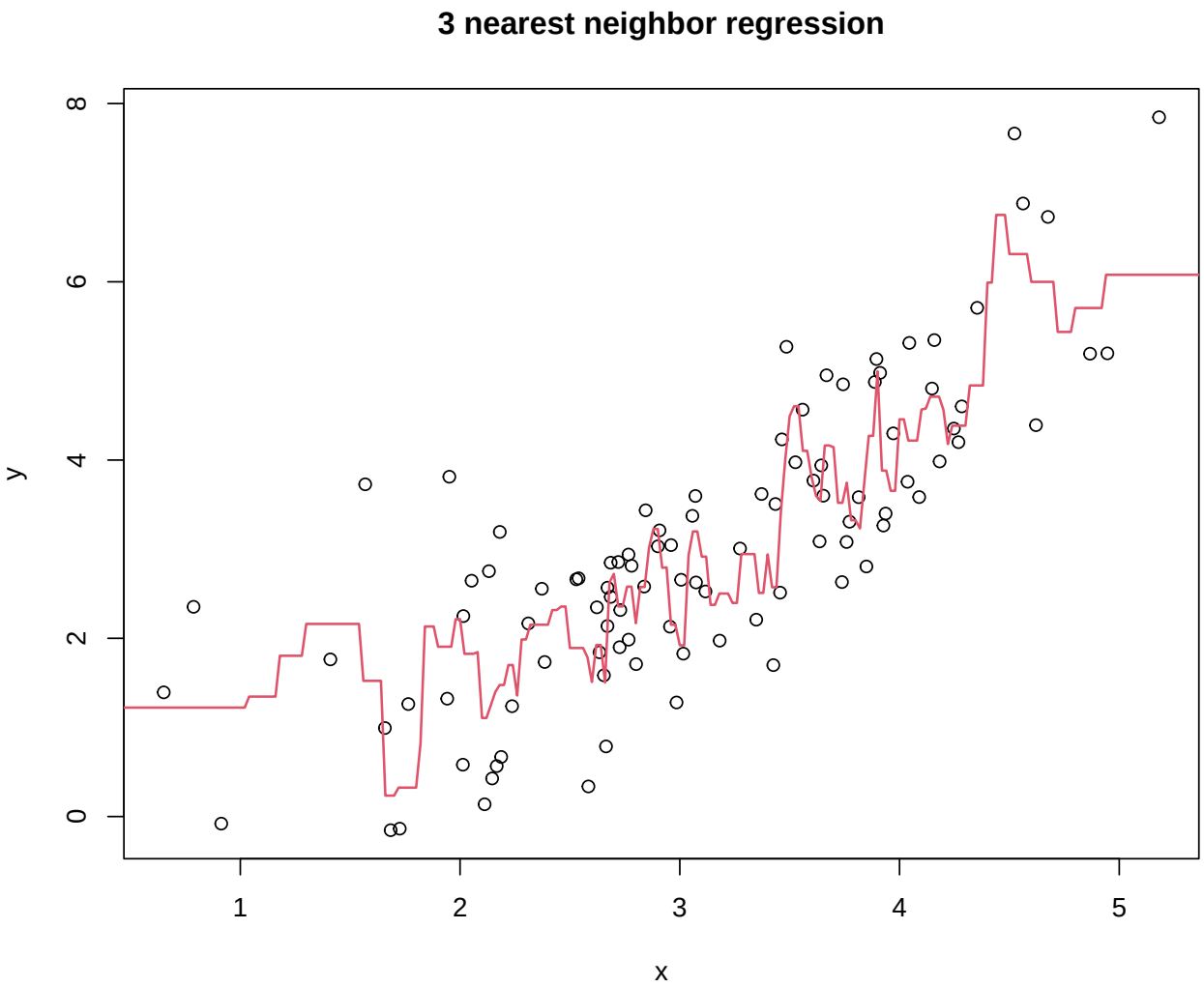
$k = 1$ nearest neighbor



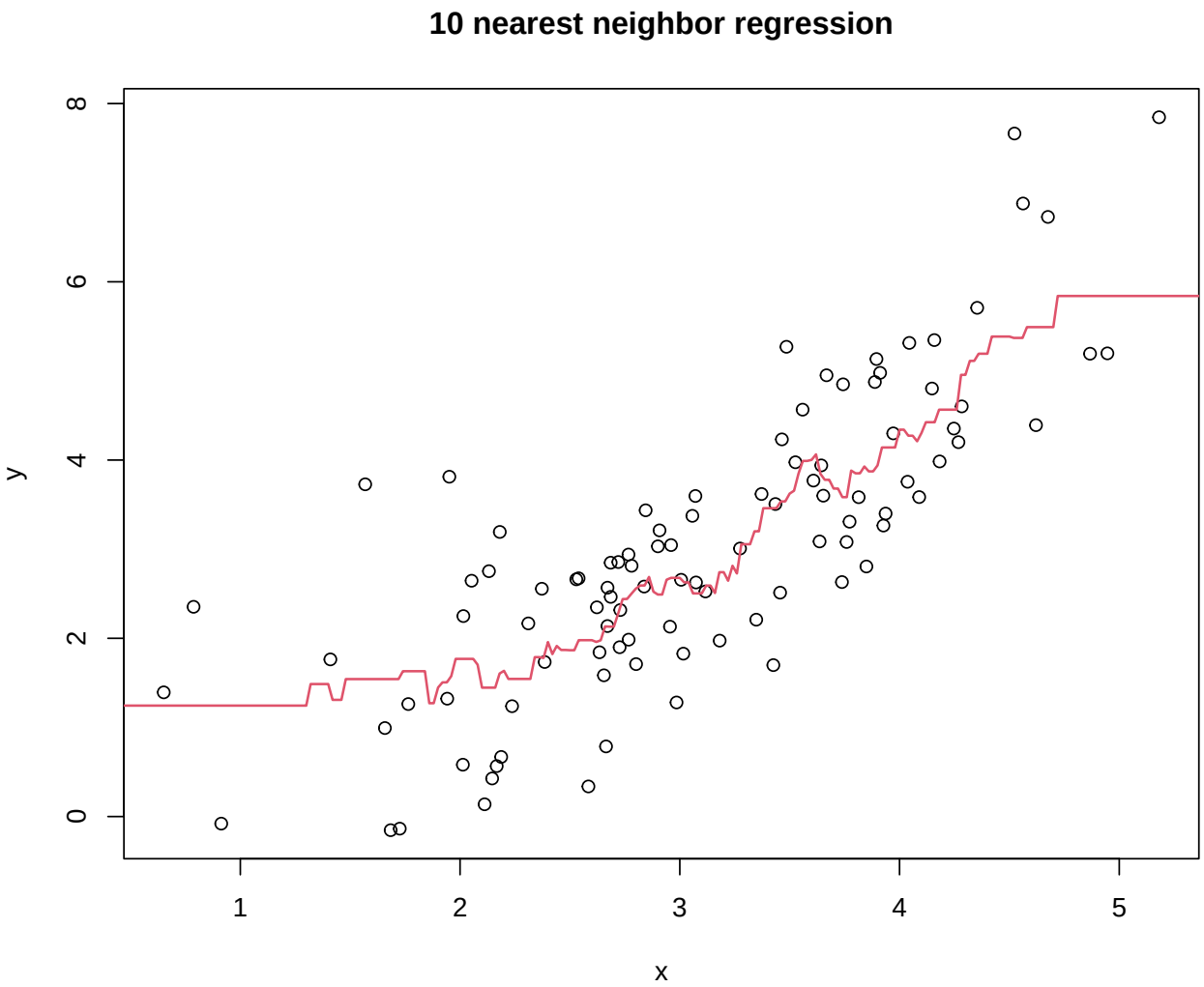
$k = 3$ nearest neighbors



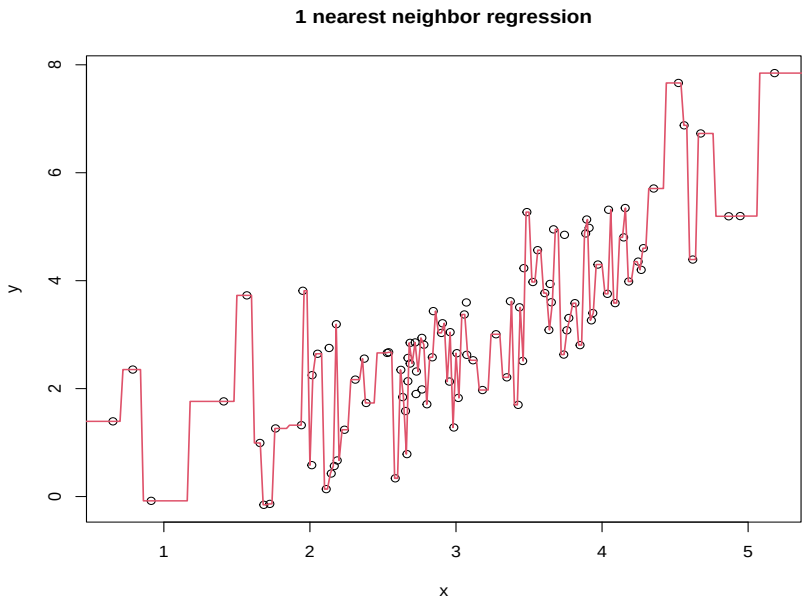
$k = 3$ nearest neighbors

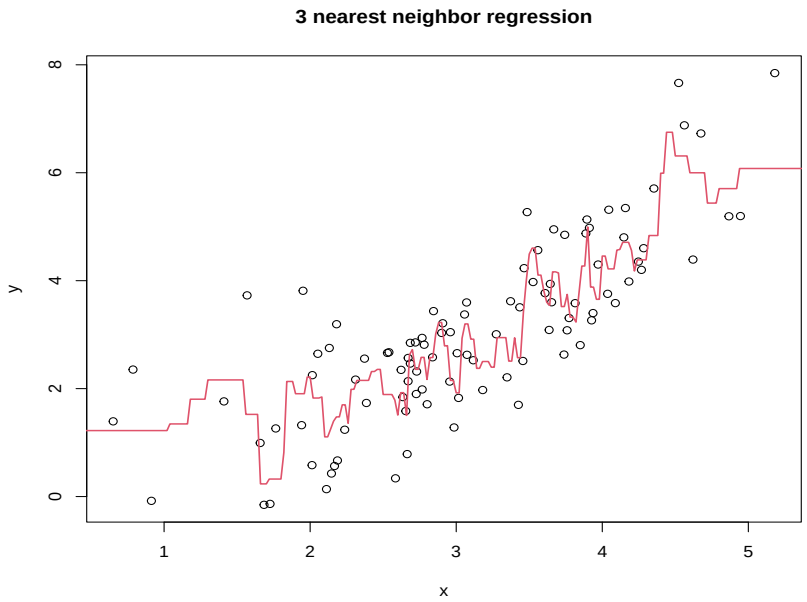


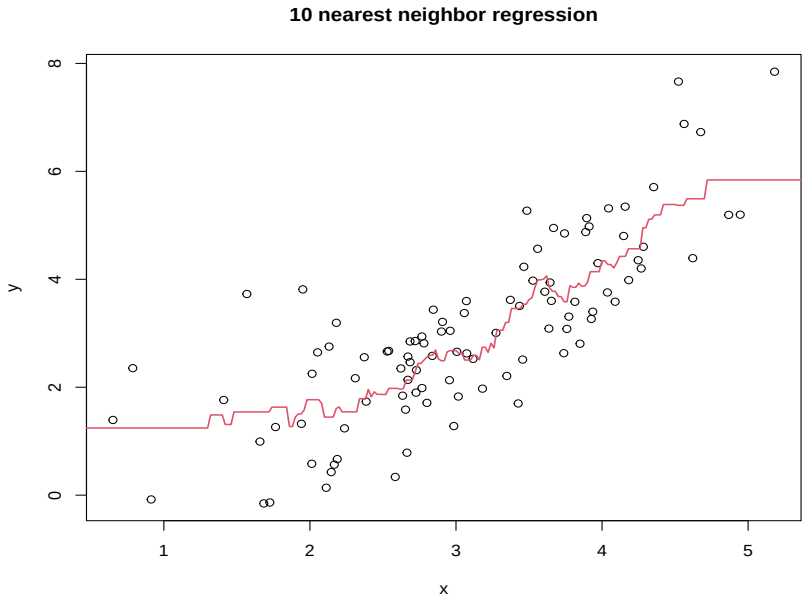
$k = 10$ nearest neighbors

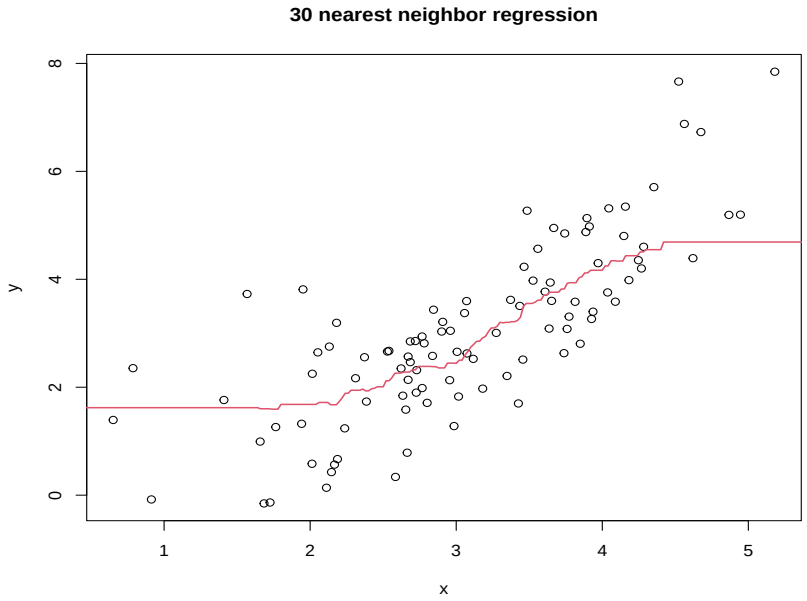


k nearest neighbours: role of k









Role of k

Controls the bias and variance tradeoff!

- A smaller k means more flexible predictor
  - Larger variance
  - Smaller bias
- How to select k?
  - CV!

Generalization of k-nn: weighted k-nn

Recall that k-nn predicts by using the local **average**

$$\hat{f}(x_0) = \sum_{i \in \{1,\ldots,n\}: x_i \in \mathcal{N}_k(x_0)} \frac{1}{k} \, y_i.$$

Can we choose different weights for each neighbour?

$$\hat{f}(x_0) = \sum_{i \in \{1,\ldots,n\}: x_i \in \mathcal{N}_k(x_0)} K(x_i, x_0) \, y_i$$

with

$$0 \leq K(x_i, x_0) \leq 1, \qquad \sum_{i \in \{1,\ldots,n\}: x_i \in \mathcal{N}_k(x_0)} K(x_i, x_0) = 1.$$

Choices of the weight

One popular choice is the so-called **inverse distance weighting** (IDW). Of course there are other more sophisticated weighting scheme.....

- IDW: Compute the inverse distances

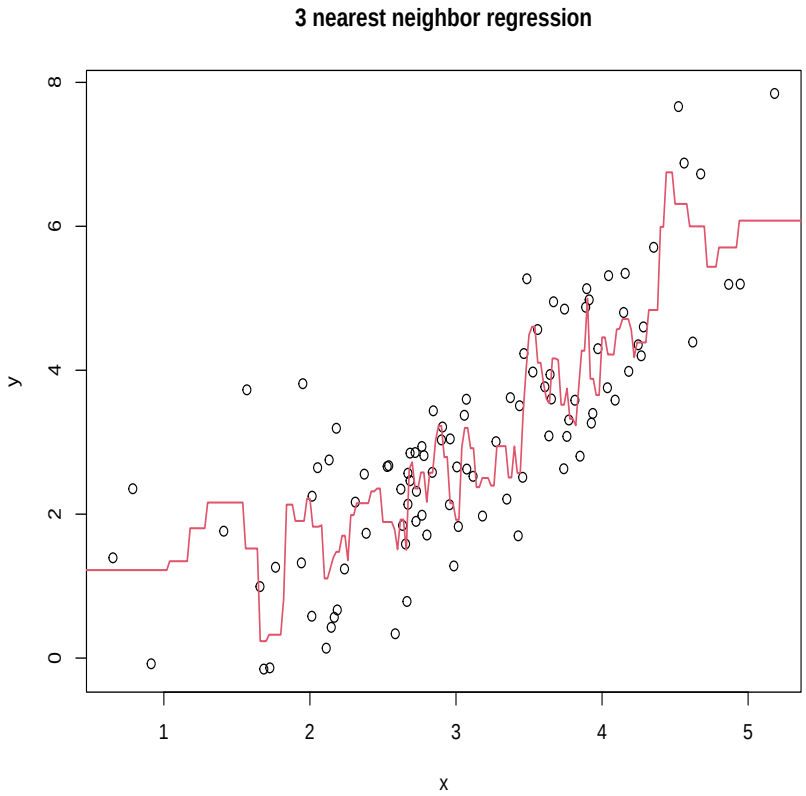
$$ID_i = \frac{1}{\|x_i - x_0\|_2}, \qquad \forall x_i \in \mathcal{N}_k(x_0).$$

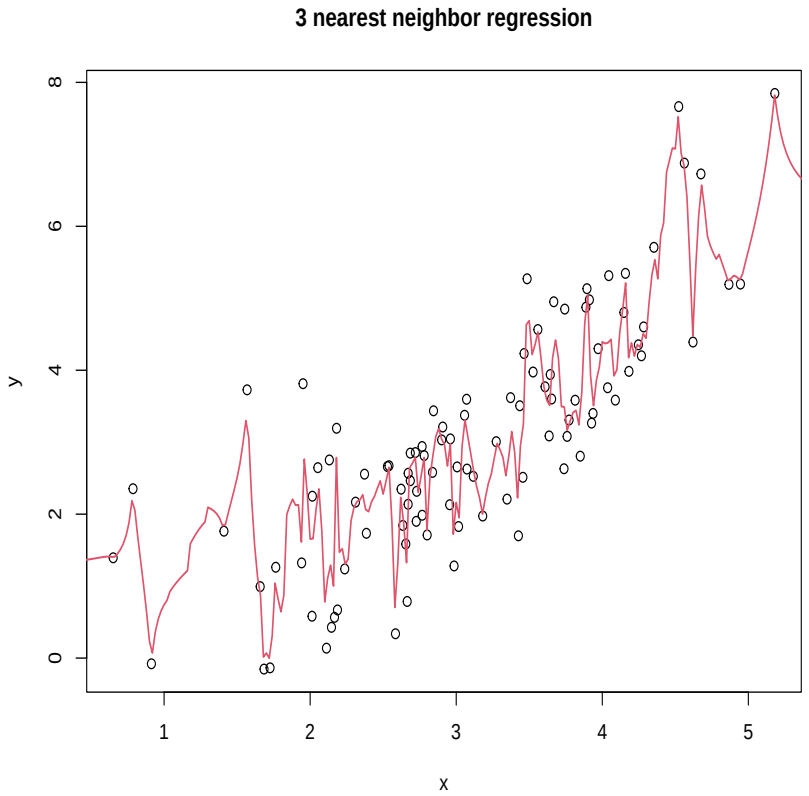
  The weights are

$$K(x_i, x_0) = \frac{ID_i}{\sum_{i:x_i \in \mathcal{N}_k(x_0)} ID_i}, \qquad \forall x_i \in \mathcal{N}_k(x_0).$$

- Tricubic weights (see, practical problem set 3)

Weighted k-nn vs k-nn ($k = 3$)





Generalization of k-nn: local regression

Recall that k-nn predicts by using the local average of the **responses**

$$\hat{f}(x_0) = \frac{1}{k} \sum_{i: x_i \in \mathcal{N}_k(x_0)} y_i. \tag{1}$$

Local (linear) regression:

$$\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0 \tag{2}$$

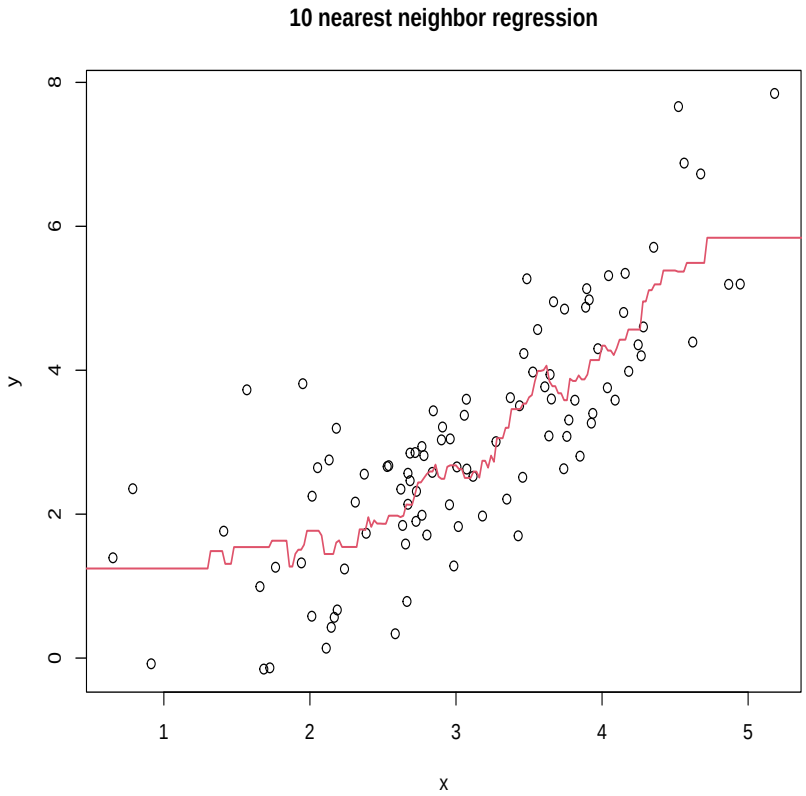
where

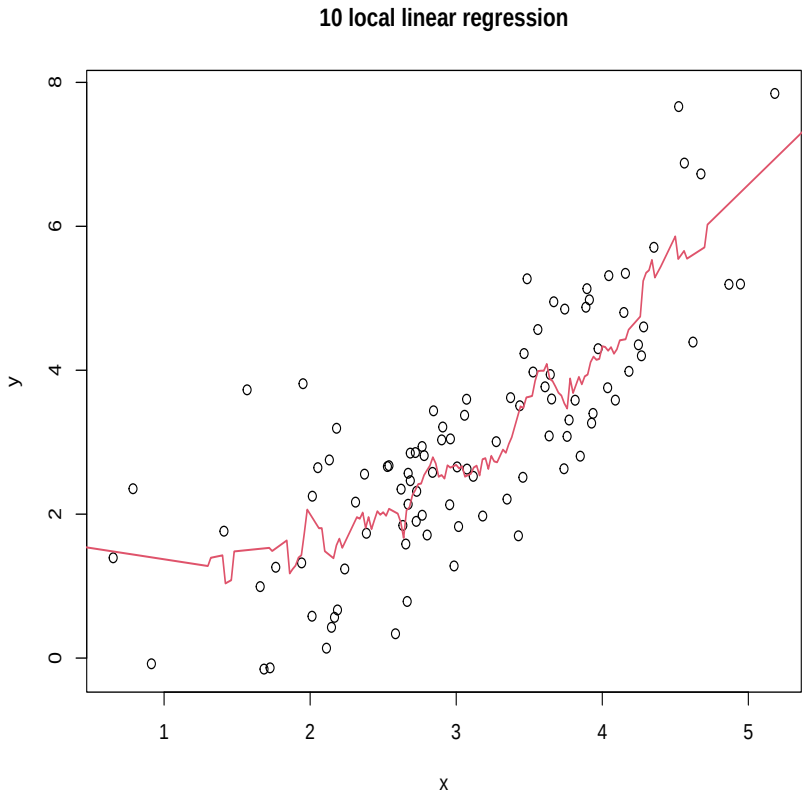
$$(\hat{\beta}_0, \hat{\beta}_1) = \underset{\beta_0, \beta_1}{\operatorname{argmin}} \sum_{i: x_i \in \mathcal{N}_k(x_0)} \frac{1}{k} (y_i - \beta_0 - \beta_1 x_i)^2.$$

Discussion:

- connection between (1) and (2)?
- can also use local polynomial regression

k-nn vs local linear regression ($k = 10$)





Local (linear) regression

Local regression predicts at a target point x_0 using only the nearby training observations in a weighted scheme.

Predict at $x = x_0$ by

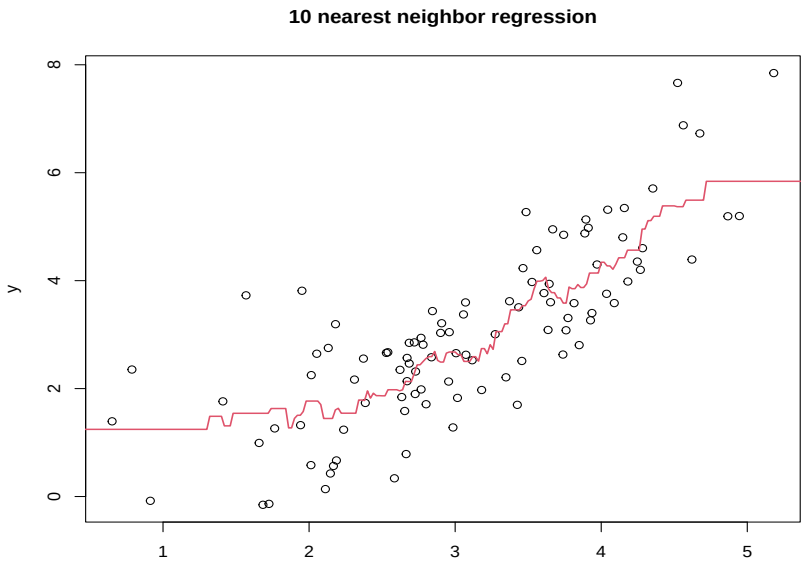
$$\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0$$

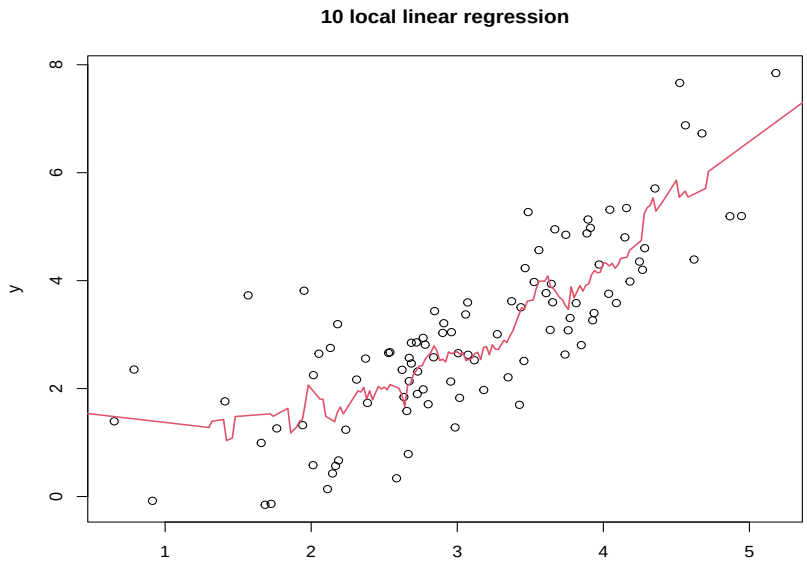
where

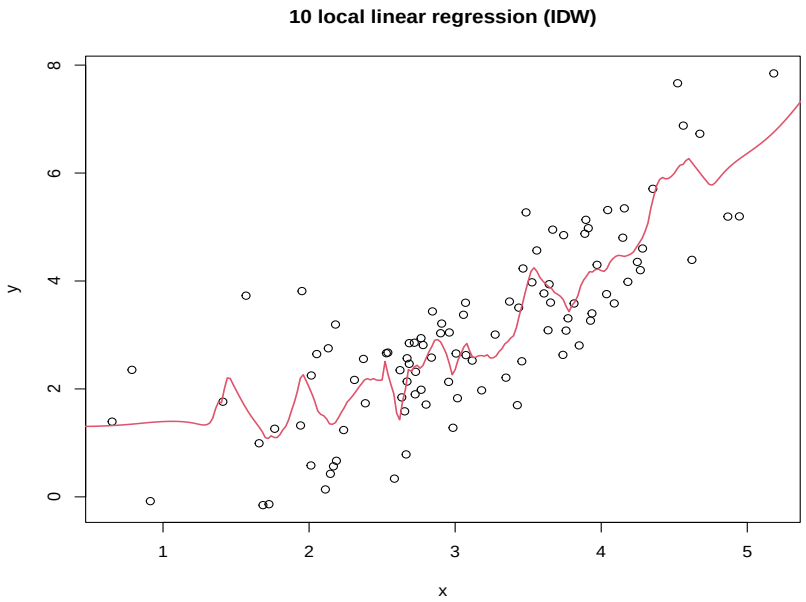
$$(\hat{\beta}_0, \hat{\beta}_1) = \underset{\beta_0, \beta_1}{\text{argmin}} \sum_{i: x_i \in \mathcal{N}_k(x_0)} K(x_i, x_0) \, (y_i - \beta_0 - \beta_1 x_i)^2,$$

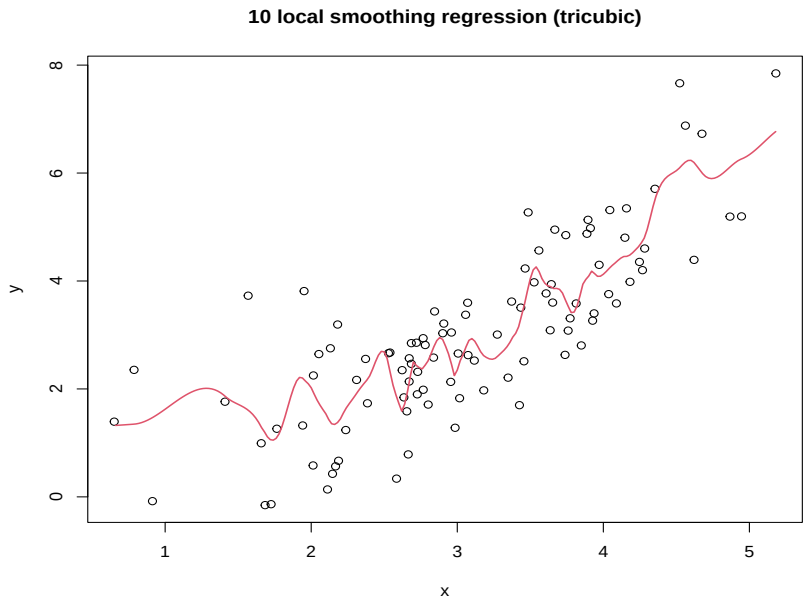
using the weighted least squares.

k-nn vs local linear regression ($k = 10$)









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.

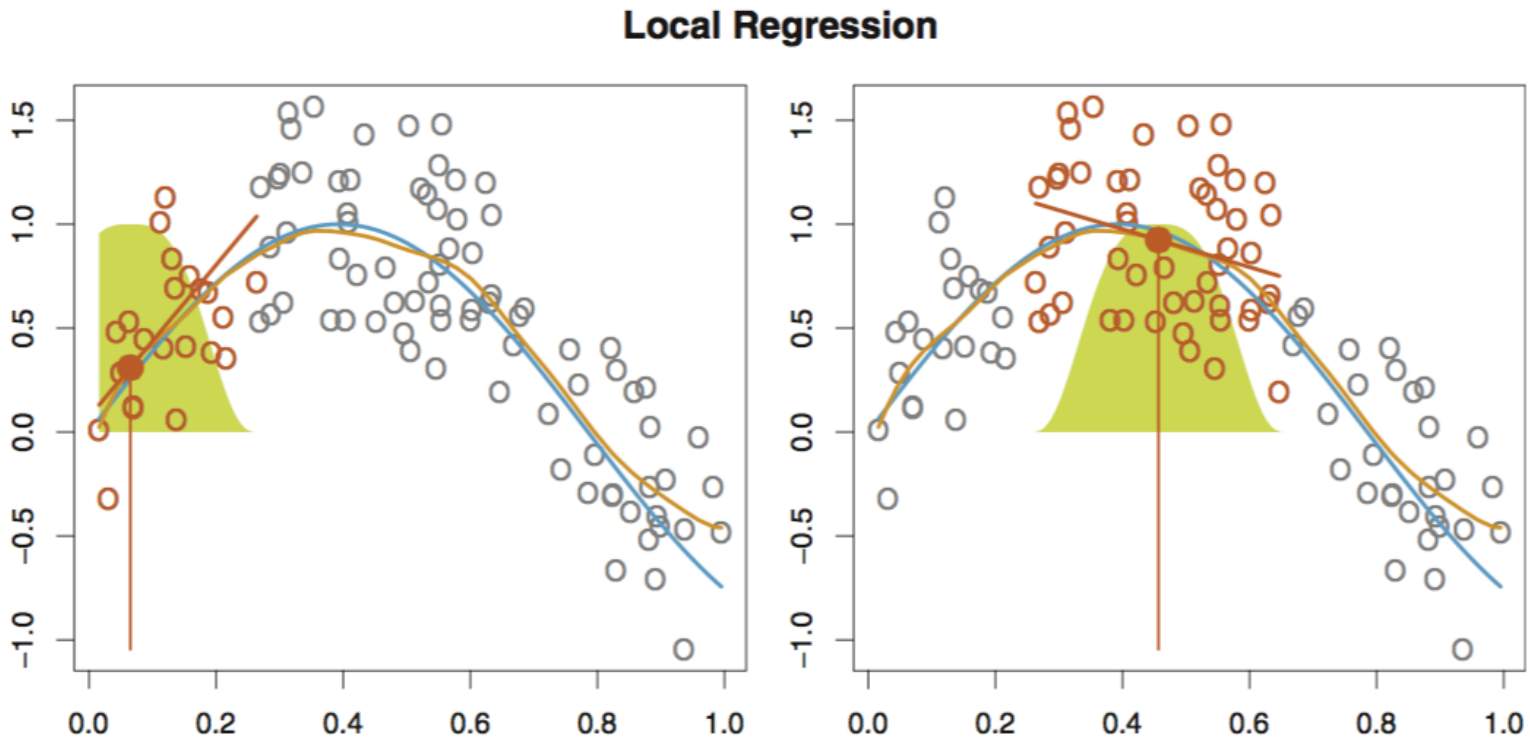
2. Assign a weight $K_{i0} = K(x_i, x_0)$ to each point in this neighborhood, so that the point furthest from x_0 has weight zero, and the closest has the highest weight. All but these k nearest neighbors get weight zero.

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

Simulated Example



The blue curve is true $f(x)$, and the light orange curve is the local regression $\hat{f}(x)$. The orange points are local to the target point x_0, represented by the orange vertical line. The yellow bell-shape indicates weights assigned to each point. The fit $\hat{f}(x_0)$ at x_0 is obtained by fitting a weighted linear regression (orange line segment), and using the fitted value at x_0 (orange solid dot) as the estimate $\hat{f}(x_0)$.

Local Regression

- The size of the neighborhood (fraction s of training data) is a tuning parameter, which can be chosen by cross-validation.
- The weight of each point in the neighborhood needs to be specified.
- When we have two dimensional predictors X_1 and X_2, we can simply use 2-dimensional neighborhoods, and fit bivariate linear regression models using the observations that are near each target point in 2-dimensional space.
- However, local regression can perform poorly if $p \geq 4$ (the curse of dimensionality).

Generalized Additive Models

- **Generalized additive models** (GAMs) provide a general framework for extending a standard linear model by allowing non-linear functions of each of the variables, while maintaining additivity,

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

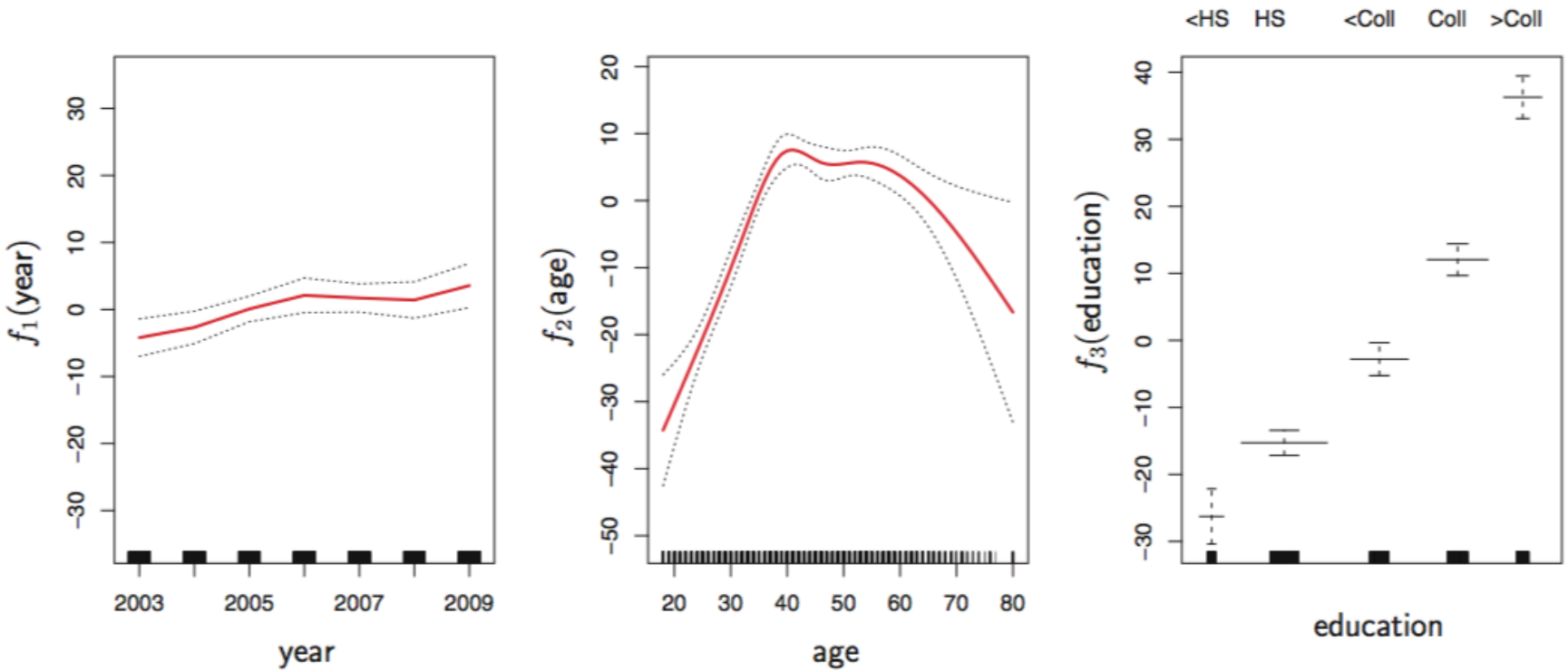
- Each f_j for $1 \leq j \leq p$ can be linear functions, polynomials, step functions, splines and local regression.
- Can be applied to classification problems.
  - Logistic regression:

$$\text{logit}\left(\mathbb{P}(Y_i = 1 \mid X_i = x_i)\right) = \beta_0 + f_1(x_{i1}) + f_2(x_{i2}) + \cdots + f_p(x_{ip}).$$

Wage Data

Consider the wage data

$$\text{wage} = \beta_0 + f_1(\text{year}) + f_2(\text{age}) + f_3(\text{education}) + \epsilon.$$



The first two functions are natural splines in year and age. The third function is a step function, fit to the qualitative variable education.

Pros and Cons of GAMs

- GAMs allow us to fit a non-linear function f_j to each X_j: model complicated relationship between the respone and the original feature space.
- The non-linear fit can potentially improve prediction accuracy.
- Because the model is additive, we can still examine the effect of each X_j on Y individually while holding all of the other variables fixed.
- It avoids the curse of dimensionality by assuming additivity.
- However, GAMs fail to incorporate the interaction of variables.

So far on regression problems

- Linear regression already covers a wide range of models!
  - Polynomials
  - Step functions
  - Splines
  - GAMs
- Local approaches
  - k-nn
  - local regressions
- Later we will learn tree-based approaches and neural nets!