Today

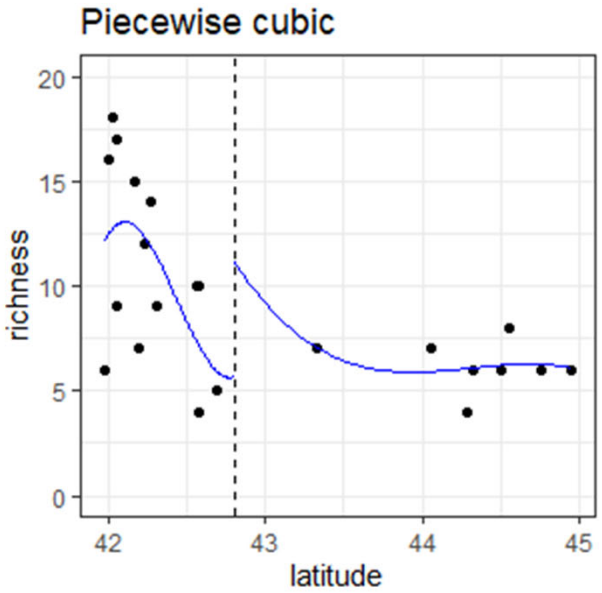
- Model algorithms
  - Smoothing spline algorithm
  - K nearest neighbors
- Training algorithm
  - optimize penalized SSQ
  - regularization
- Theory of bias-variance tradeoff

Smoothing spline model algo

- James et al. Ch 7.4 - 7.5
- Cubic smoothing spline
  - piecewise cubic polynomial
- Knots (joins) at data
- Constraints: continuous first and second derivates at knots

Building a smoothing spline

Ants data

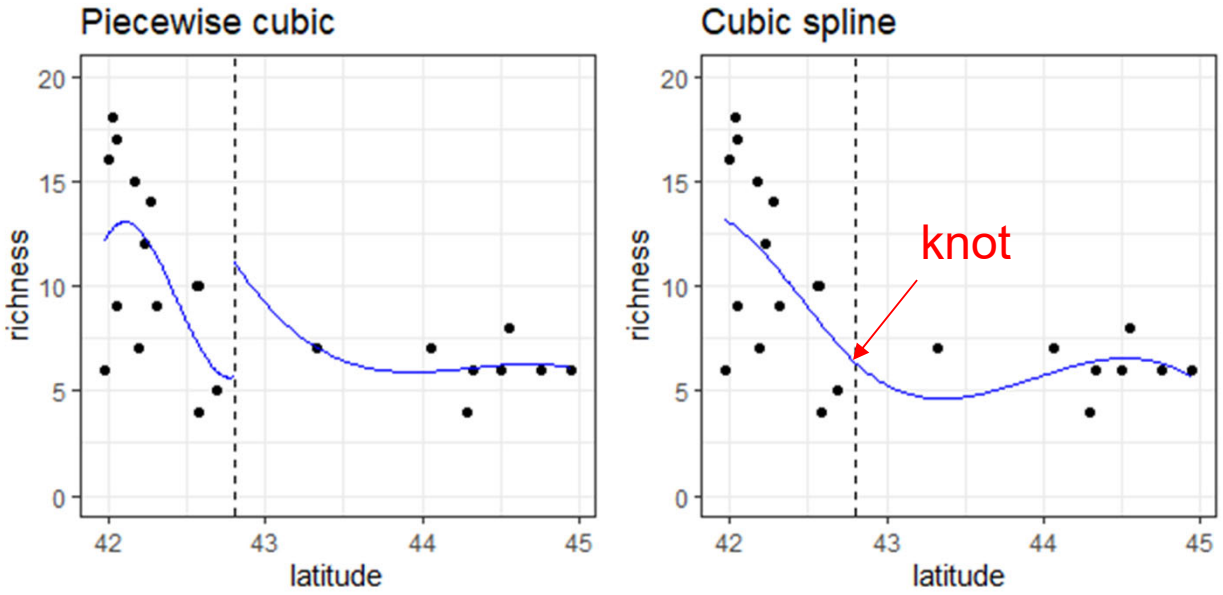


Cubic polynomials:

$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$

Building a smoothing spline

Ants data

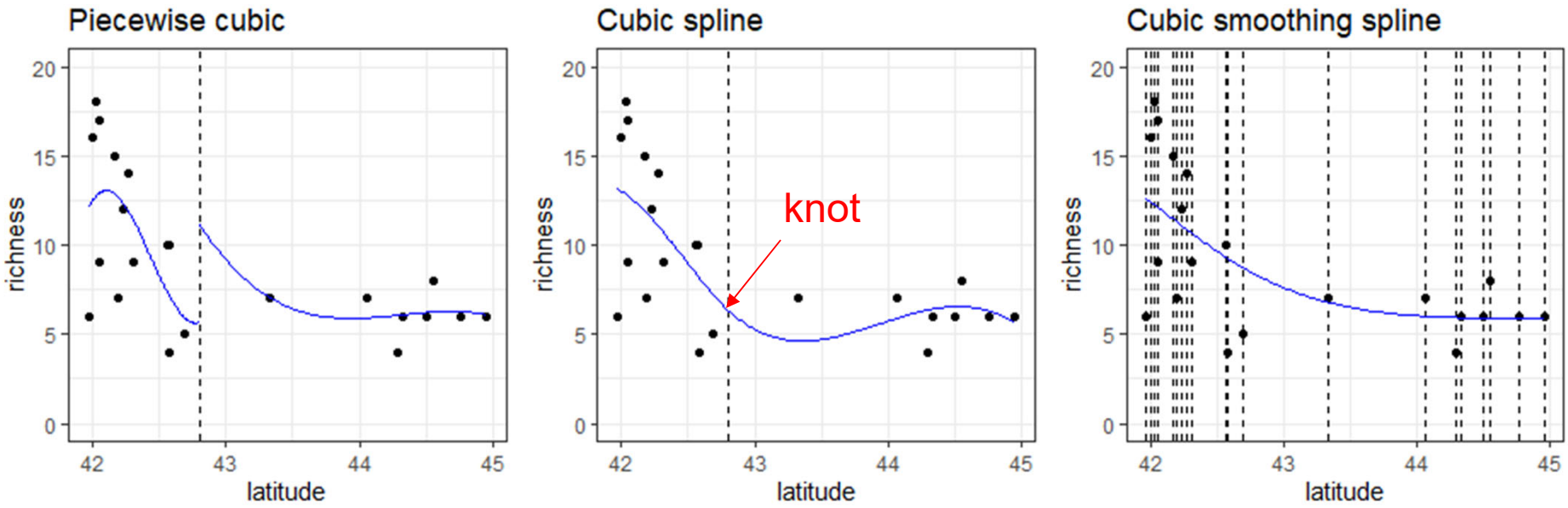


Cubic polynomials:
$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$

+ continuous 1st & 2nd derivatives at knot

Building a smoothing spline

Ants data



Cubic polynomials:
$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$

+ continuous 1st & 2nd derivatives at knot

+ knots at each datum

Smoothing spline training algo

Penalized least squares

Optimize this objective function:

$$\underbrace{\sum_{i=1}^{n}\left(y_i - f(x_i)\right)^2}_{\text{SSQ}} + \underbrace{\lambda \int f''(x)^2 dx}_{\text{Penalty}}$$

Regularization
Shrinks βs of polynomials toward zero

Penalty term = “wiggliness”

λ is held constant to optimize

Tuning parameter d.f. is a function of λ. We vary d.f. in an inference algorithm to find the d.f. (hence λ) with the best predictive performance.

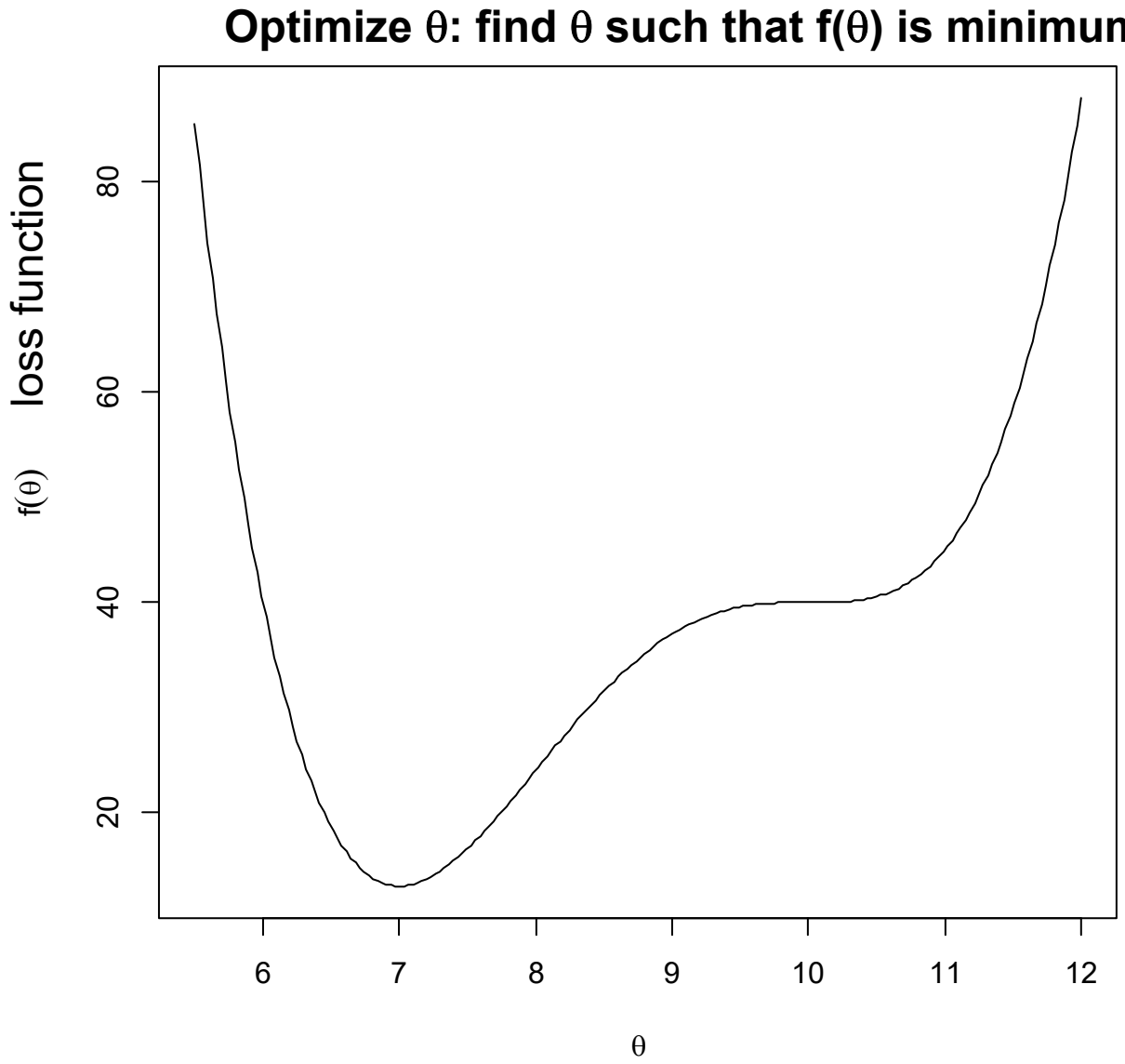
Smoothing spline training algo

Optimizing algorithm
Strategy 2: descent algorithm

Diving deep into the source code for smooth.spline() we come to the file sbart.c where we find the exact algorithm is due to Forsythe, Malcom & Moler, presumably circa 1977, in turn a slightly modified version of the Algol 60 (!!) procedure localmin from Brent (1973) *Algorithms for minimization without derivatives*, Prentice-Hall. Such legacy procedures and code are the case for much numerical work!

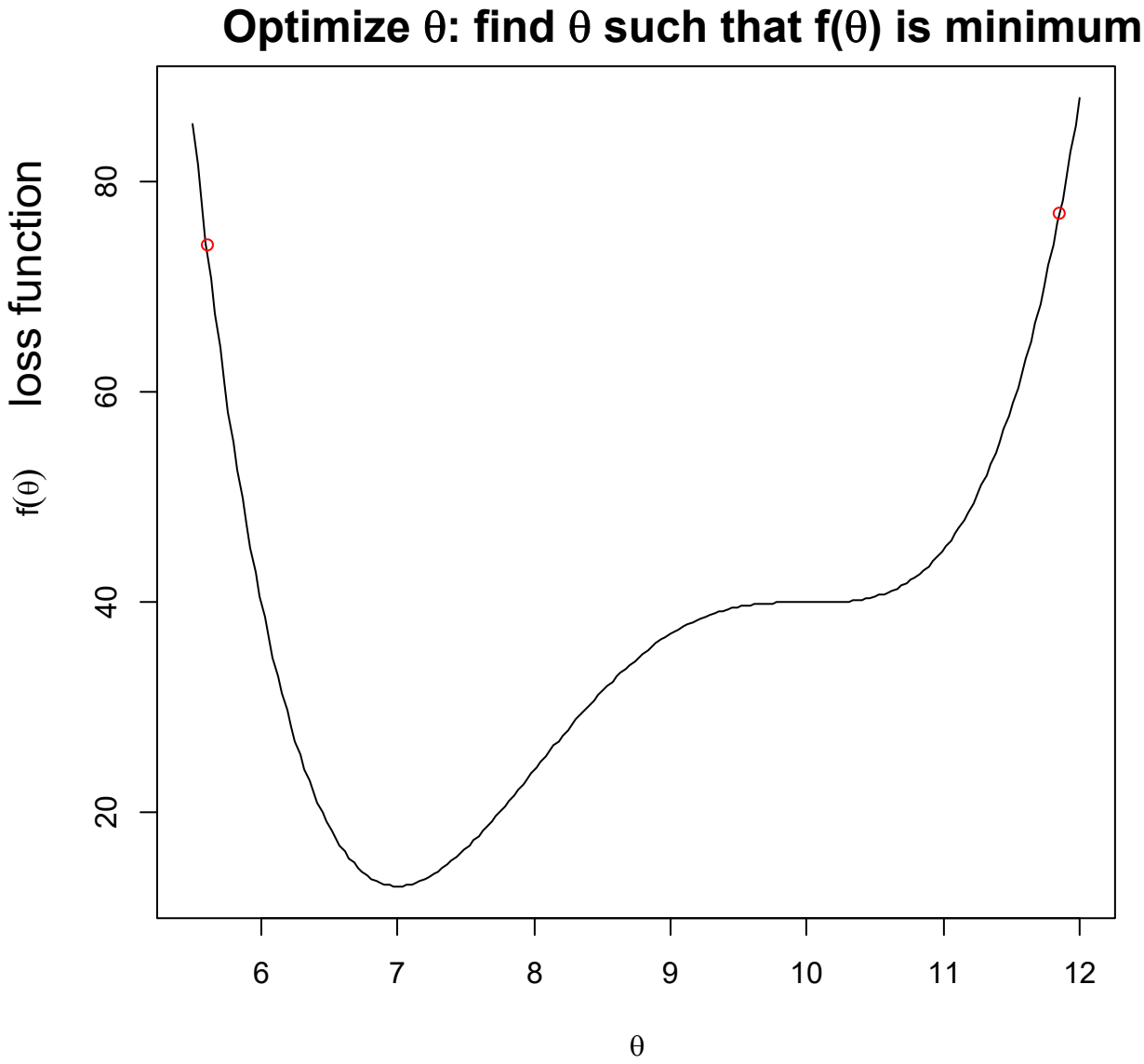
The algorithm uses a combination of golden section search and successive parabolic interpolation. We don't need to worry about these details but the following slides give a sense for descent algorithms in general using the simple bisection algorithm as an example.

A feel for descent algorithms



Bisection algorithm

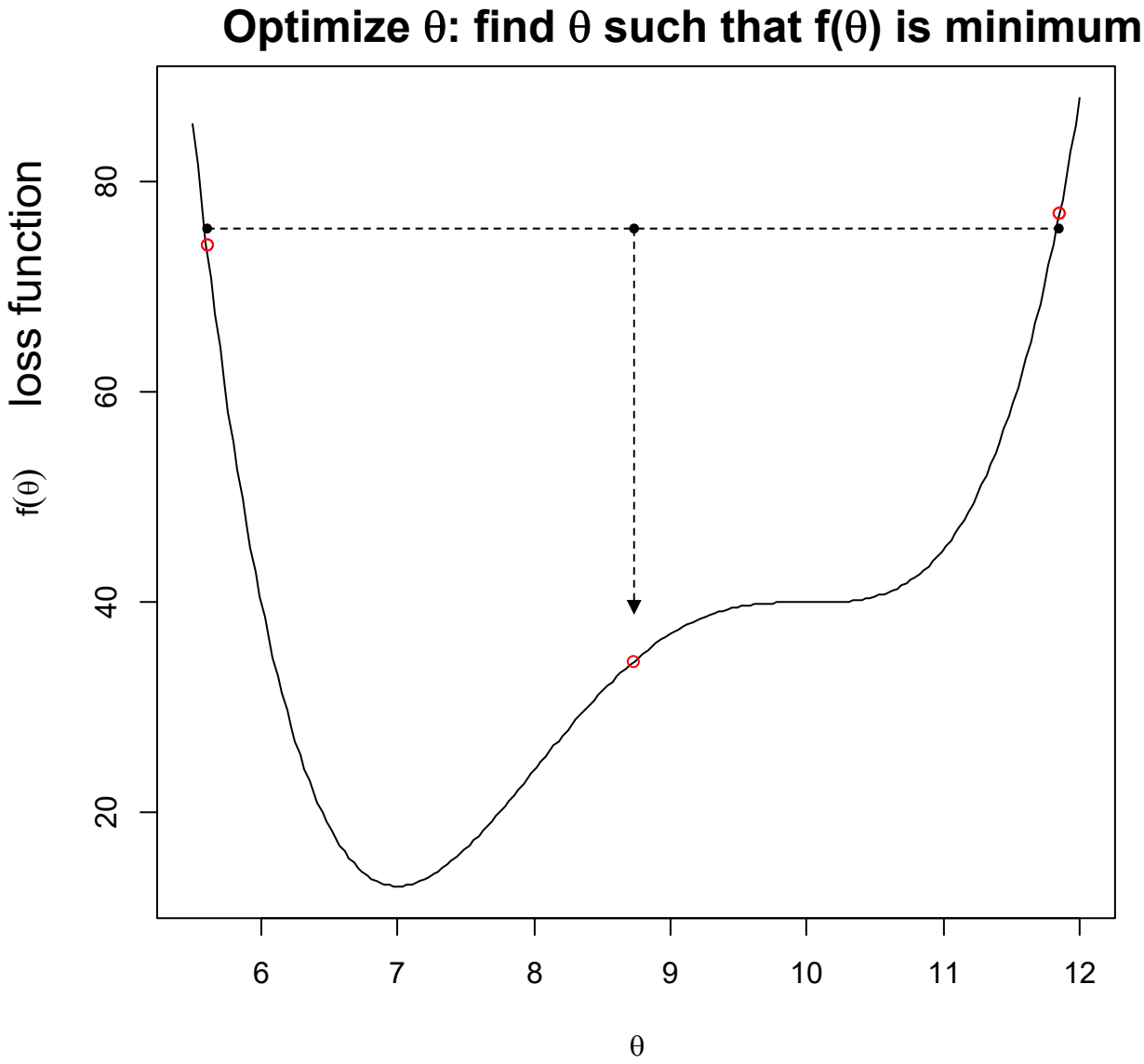
A feel for descent algorithms



Bisection algorithm

Start with 2 points

A feel for descent algorithms

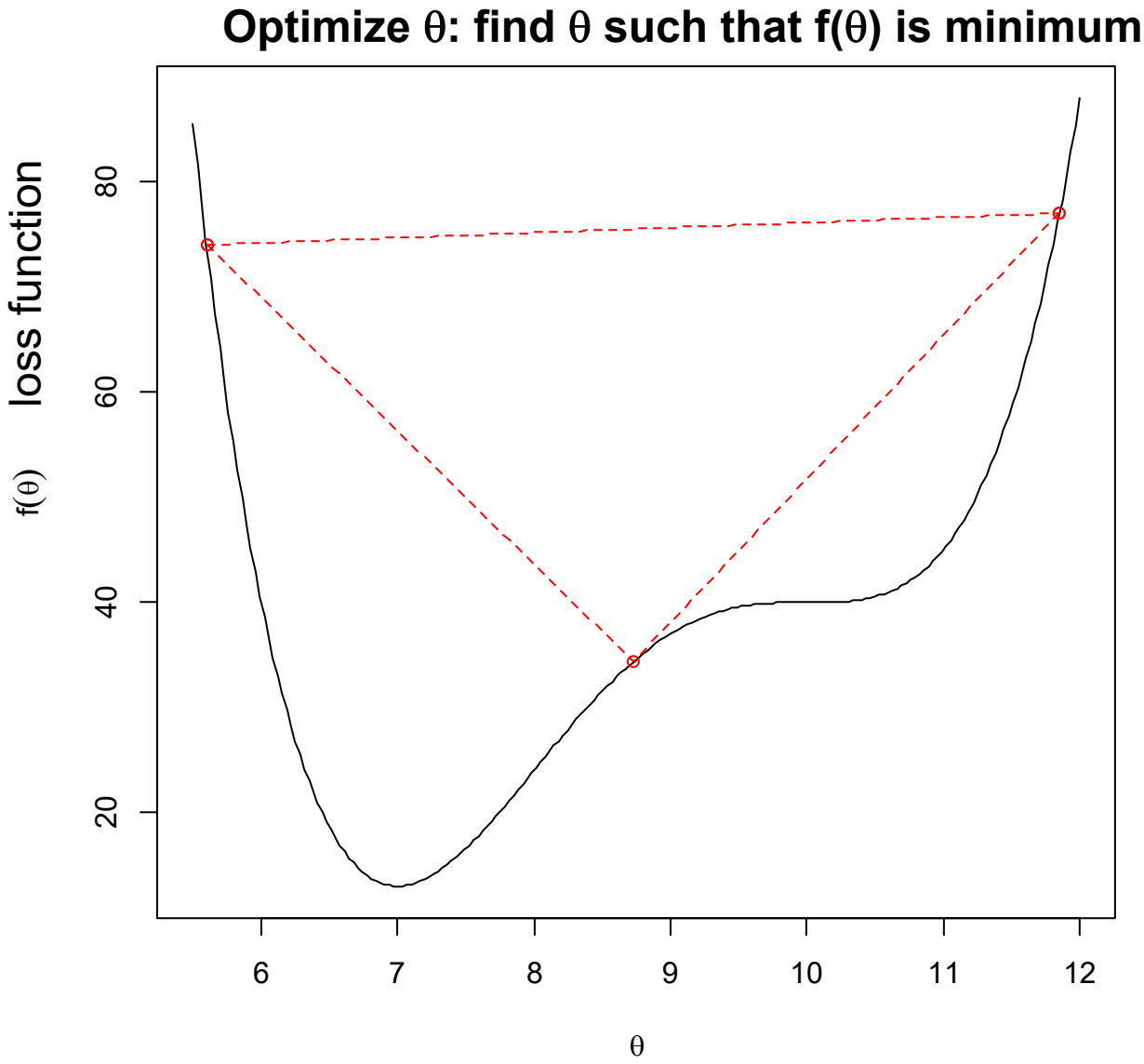


Bisection algorithm

Start with 2 points

Bisect

A feel for descent algorithms



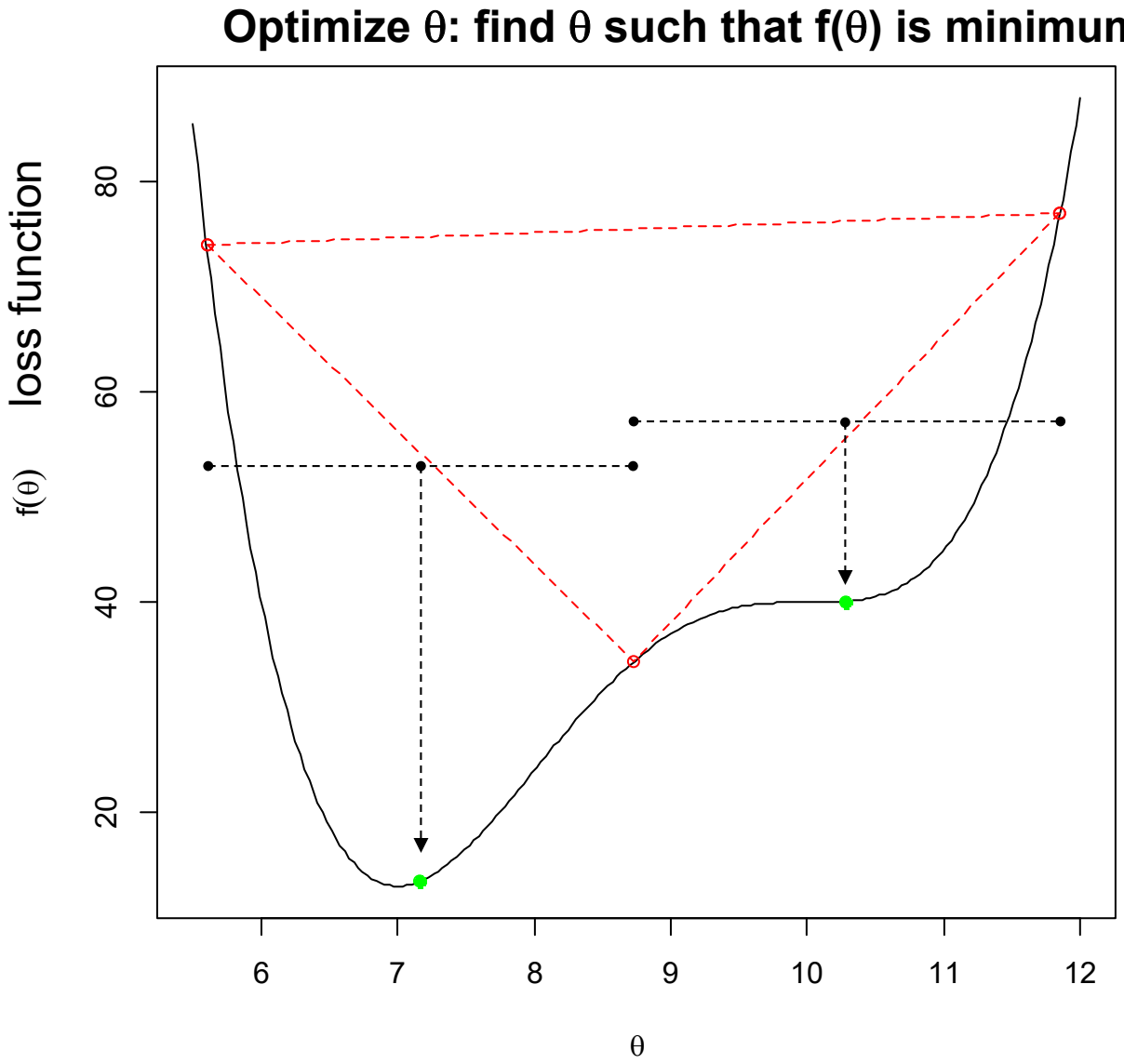
Bisection algorithm

Start with 2 points

Bisect

Make triangle

A feel for descent algorithms



Bisection algorithm

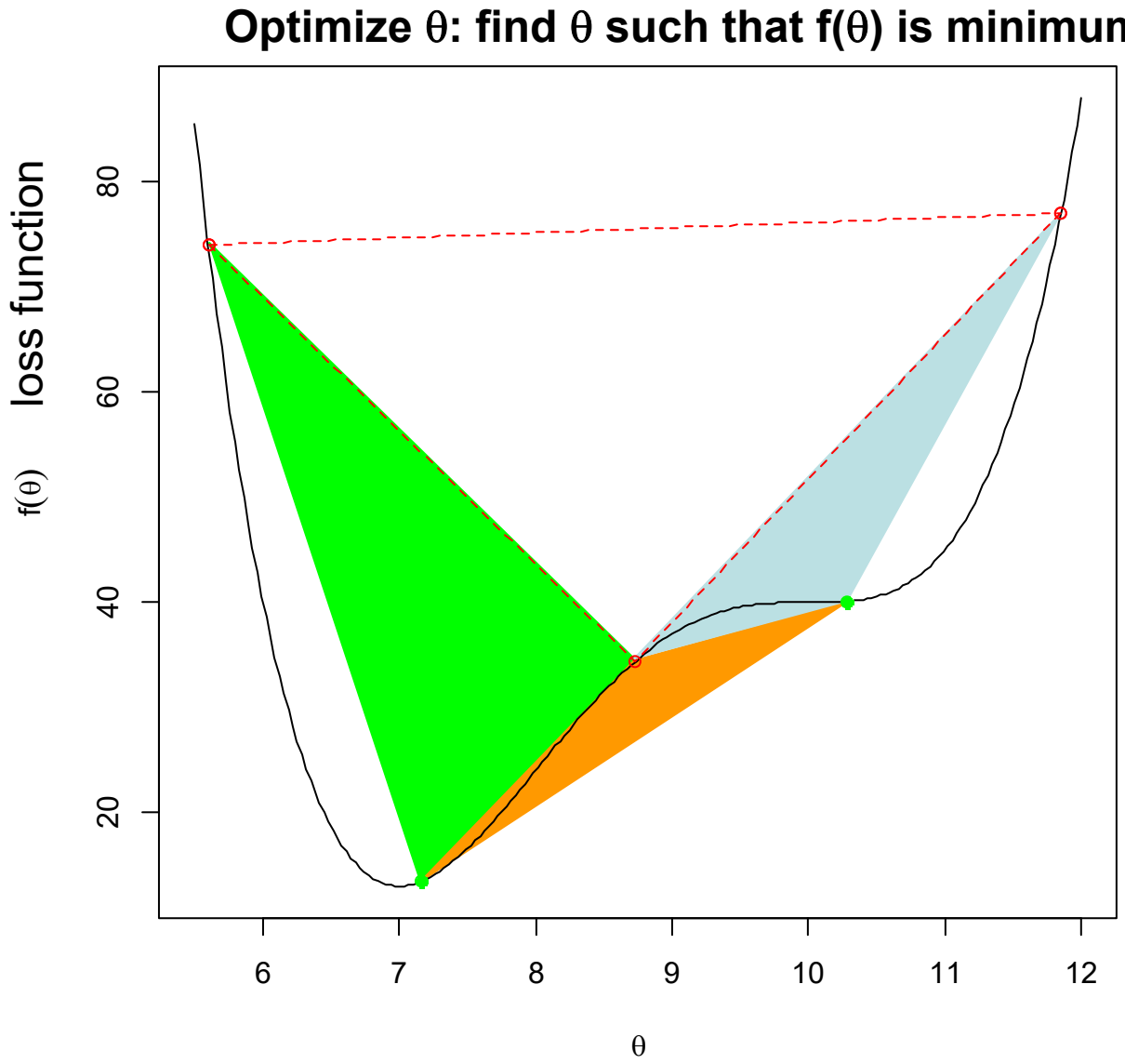
Start with 2 points

Bisect

Make triangle

Bisect lower sides

A feel for descent algorithms



Bisection algorithm

Start with 2 points

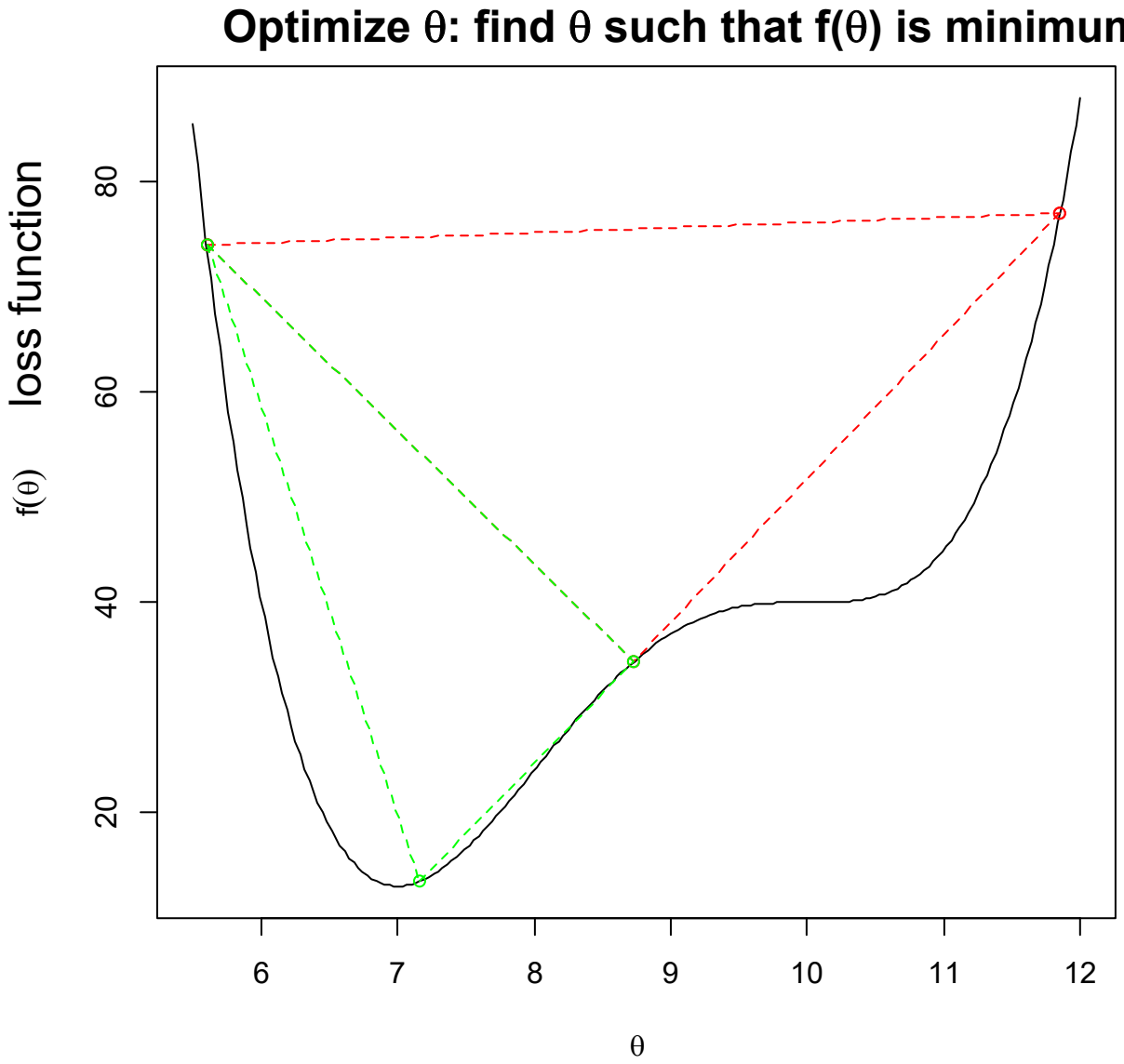
Bisect

Make triangle

Bisect lower sides

Make lowest triangle

A feel for descent algorithms



Bisection algorithm

Start with 2 points

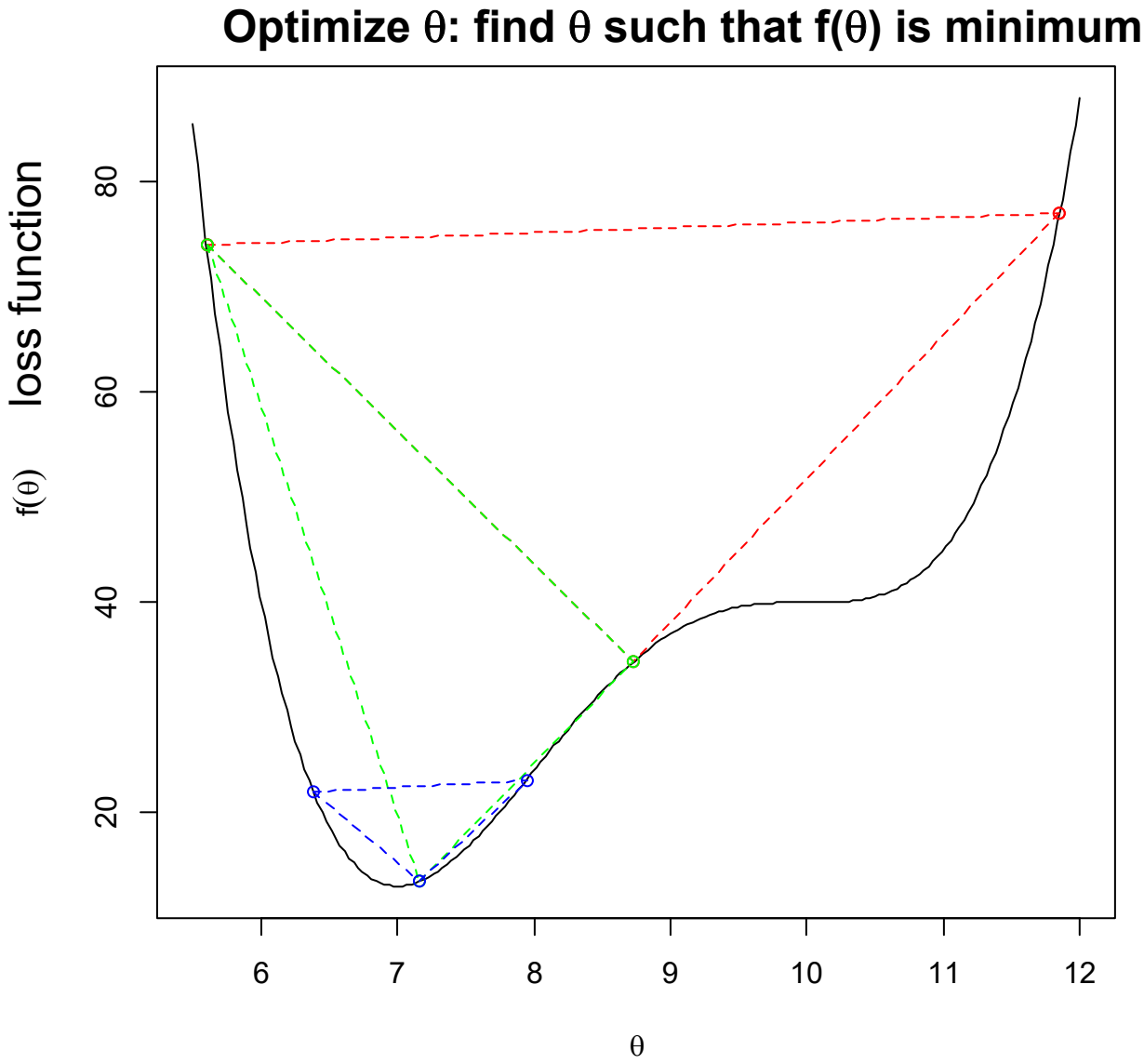
Bisect

Make triangle

Bisect lower sides

Make lowest triangle

A feel for descent algorithms



Bisection algorithm

Start with 2 points

Bisect

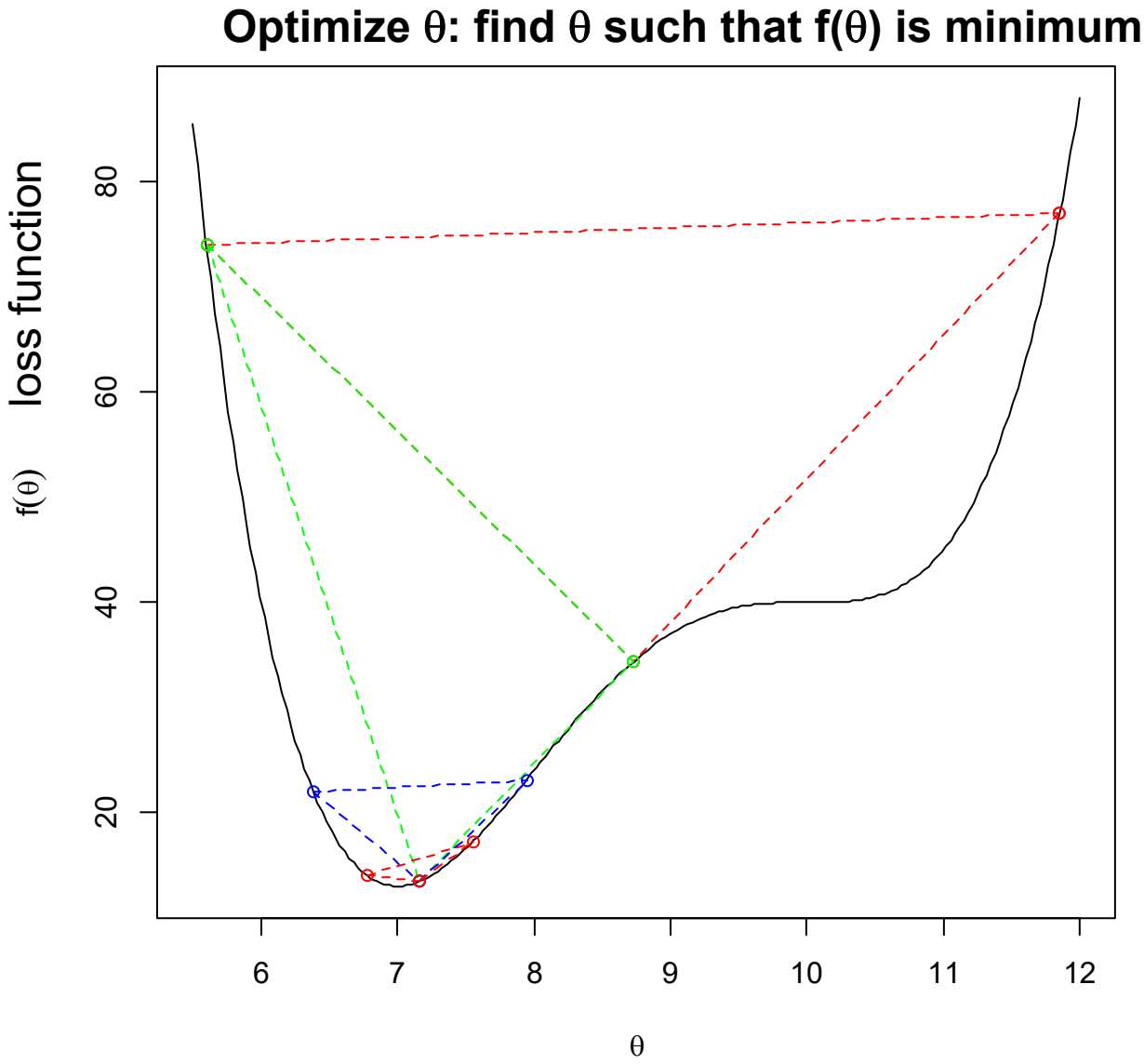
Make triangle

Bisect lower sides

Make lowest triangle

Keep repeating

A feel for descent algorithms



Bisection algorithm

Start with 2 points

Bisect

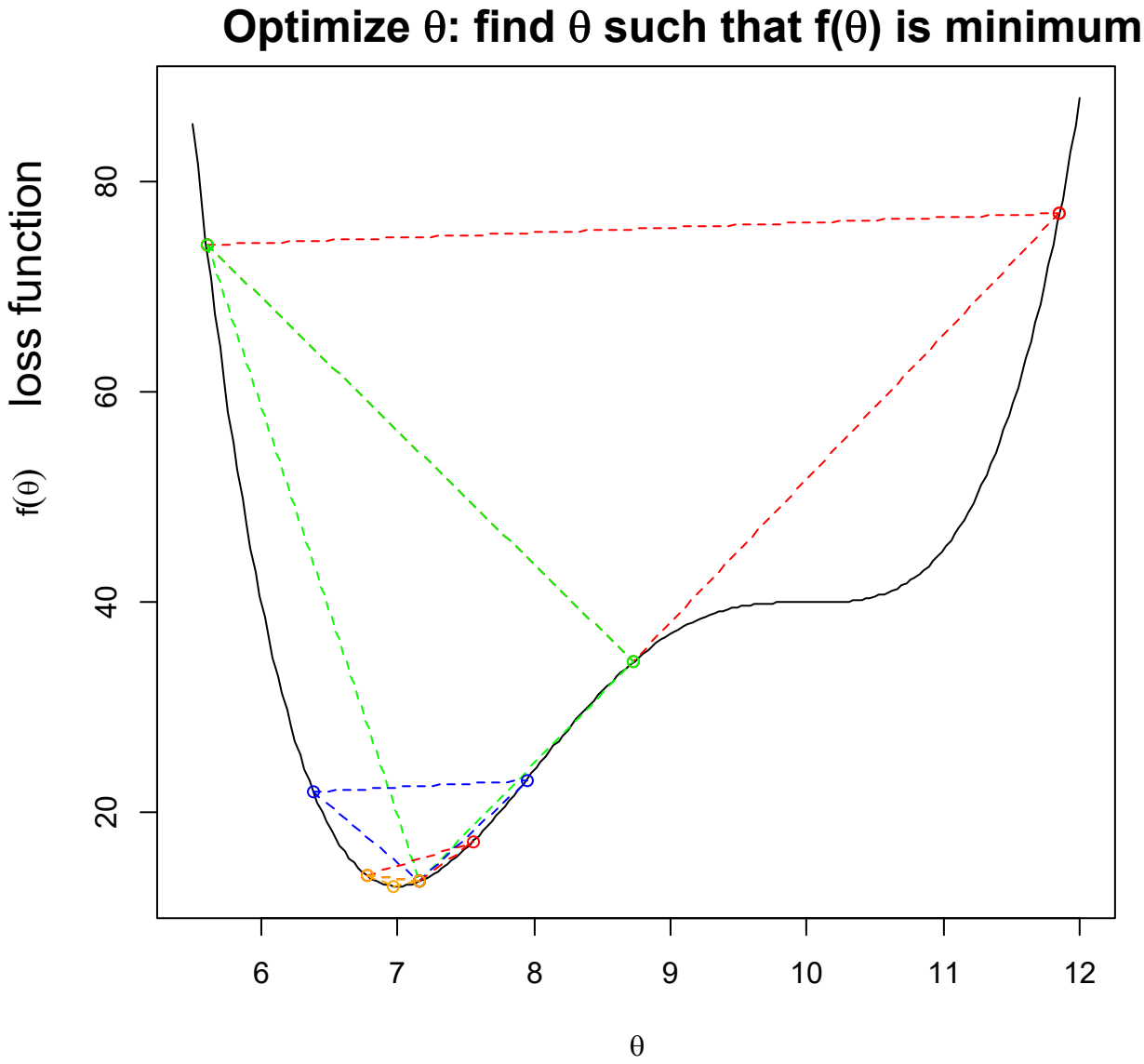
Make triangle

Bisect lower sides

Make lowest triangle

Keep repeating

A feel for descent algorithms



Bisection algorithm

Start with 2 points

Bisect

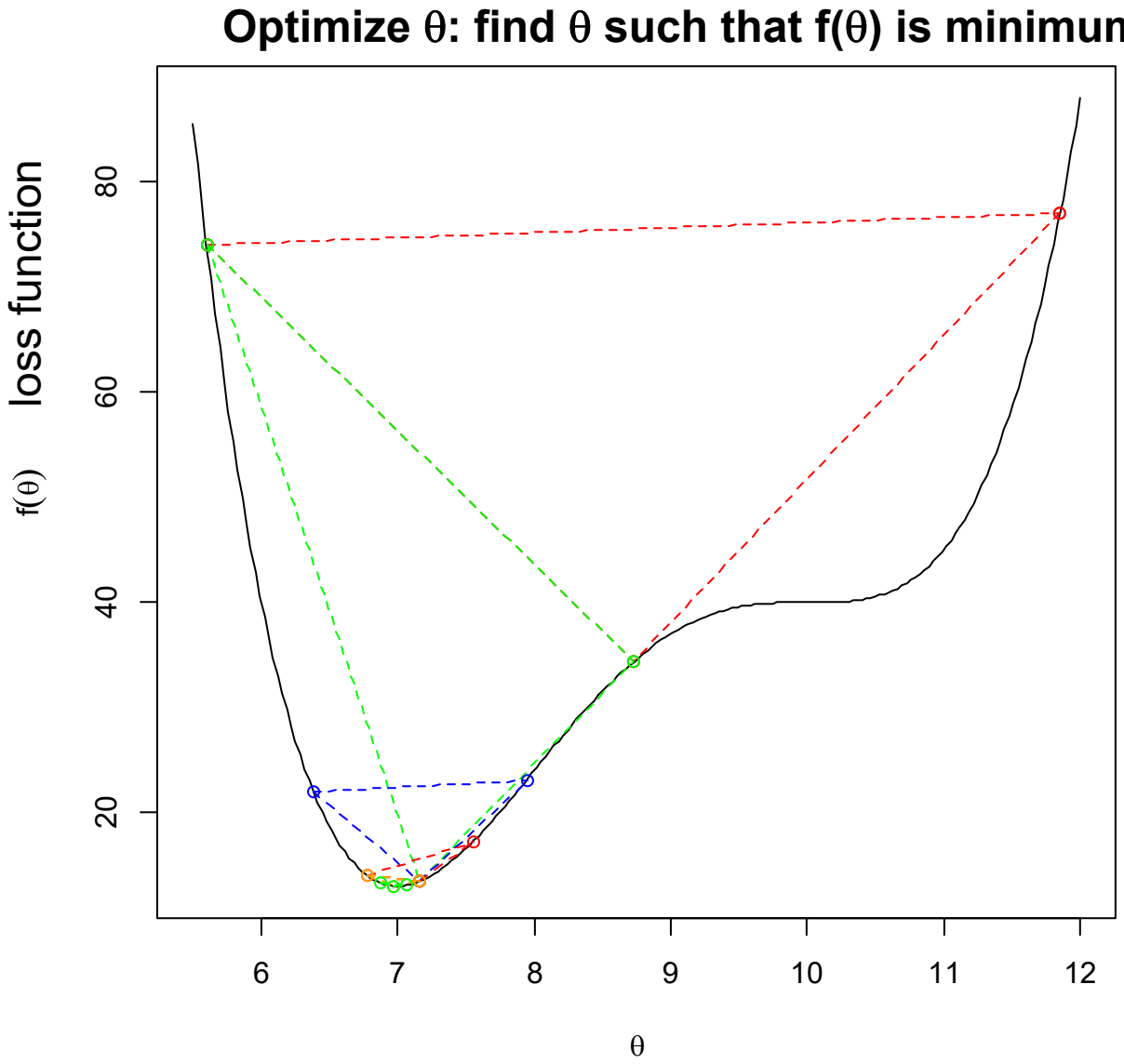
Make triangle

Bisect lower sides

Make lowest triangle

Keep repeating

A feel for descent algorithms



Bisection algorithm

Start with 2 points

Bisect

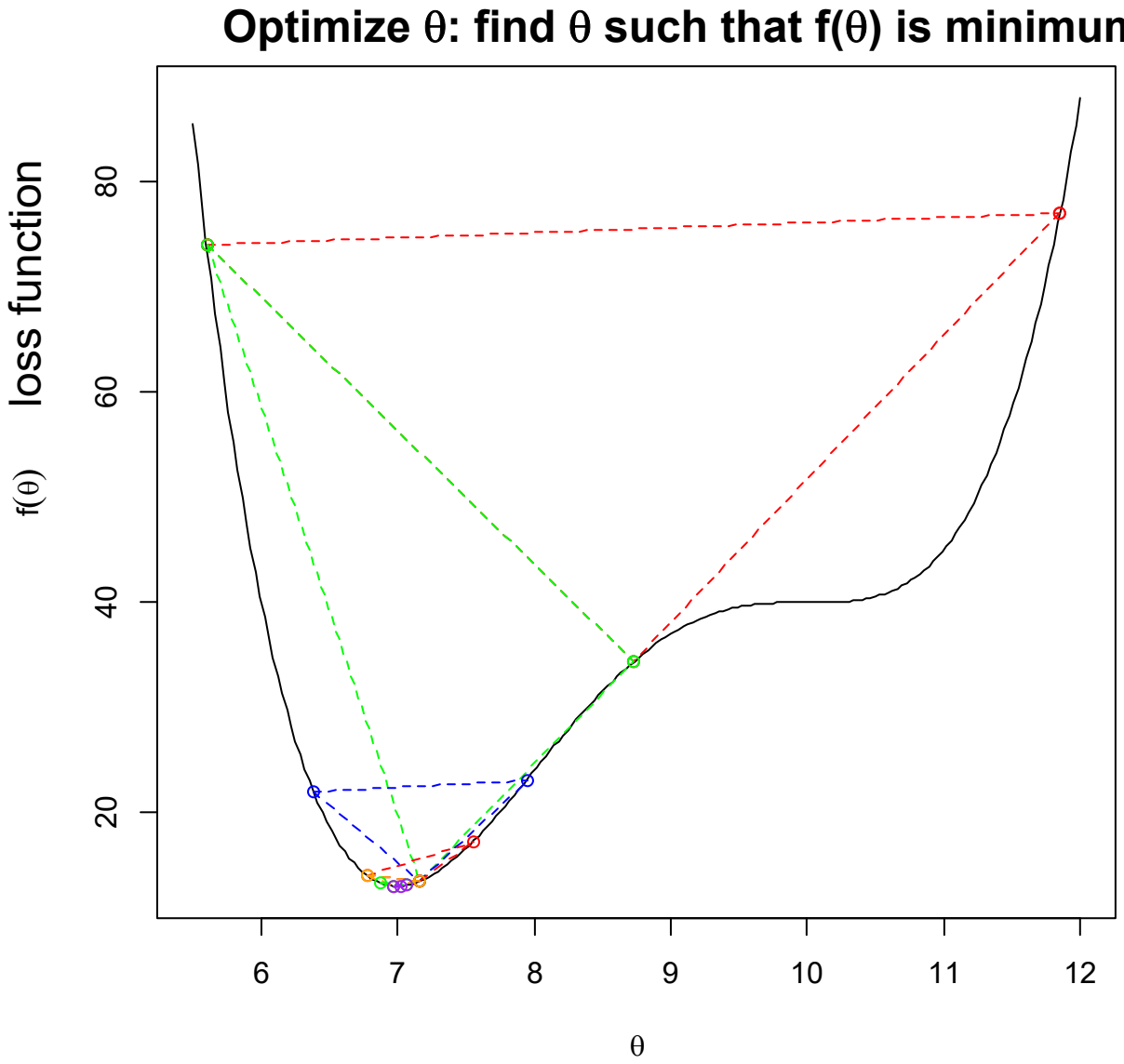
Make triangle

Bisect lower sides

Make lowest triangle

Keep repeating

A feel for descent algorithms



Bisection algorithm

Start with 2 points

Bisect

Make triangle

Bisect lower sides

Make lowest triangle

Keep repeating

Roll your own

Almost always we'll be using whatever optimization algorithm is implemented within the R function that trains the model, e.g. within lm() or smooth.spline(). Usually the function authors will have made an excellent choice for that particular model!

If you need to roll your own, an excellent starting place is the versatile Nelder-Mead simplex algorithm, a descent algorithm, and the default in the R function optim().

Tuning parameter

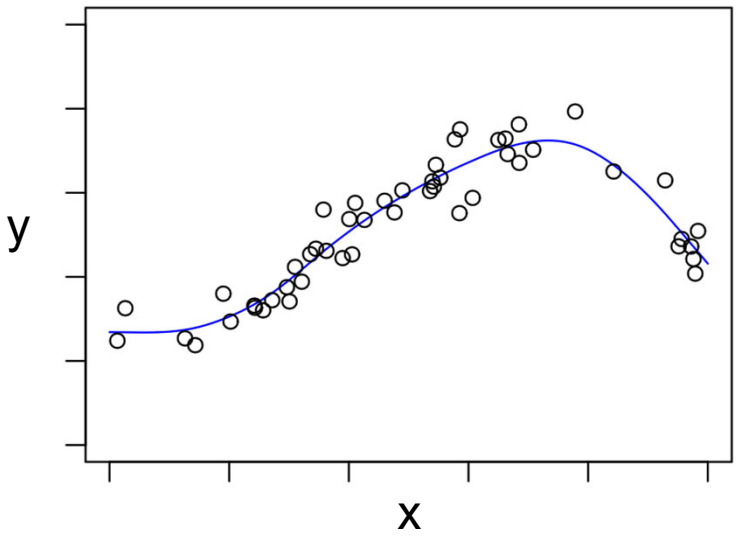
- Smoothing spline
- df = degrees of freedom
- df is a function of lambda

Code

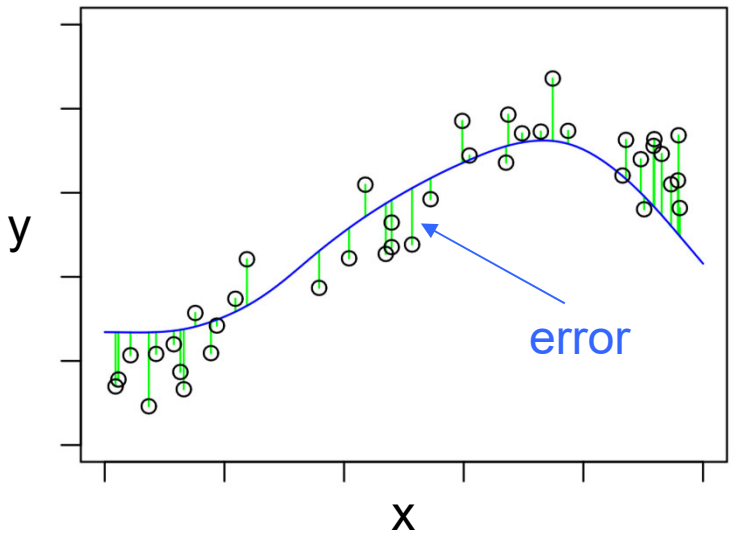
- 02_2_ants_cv_smooth.R

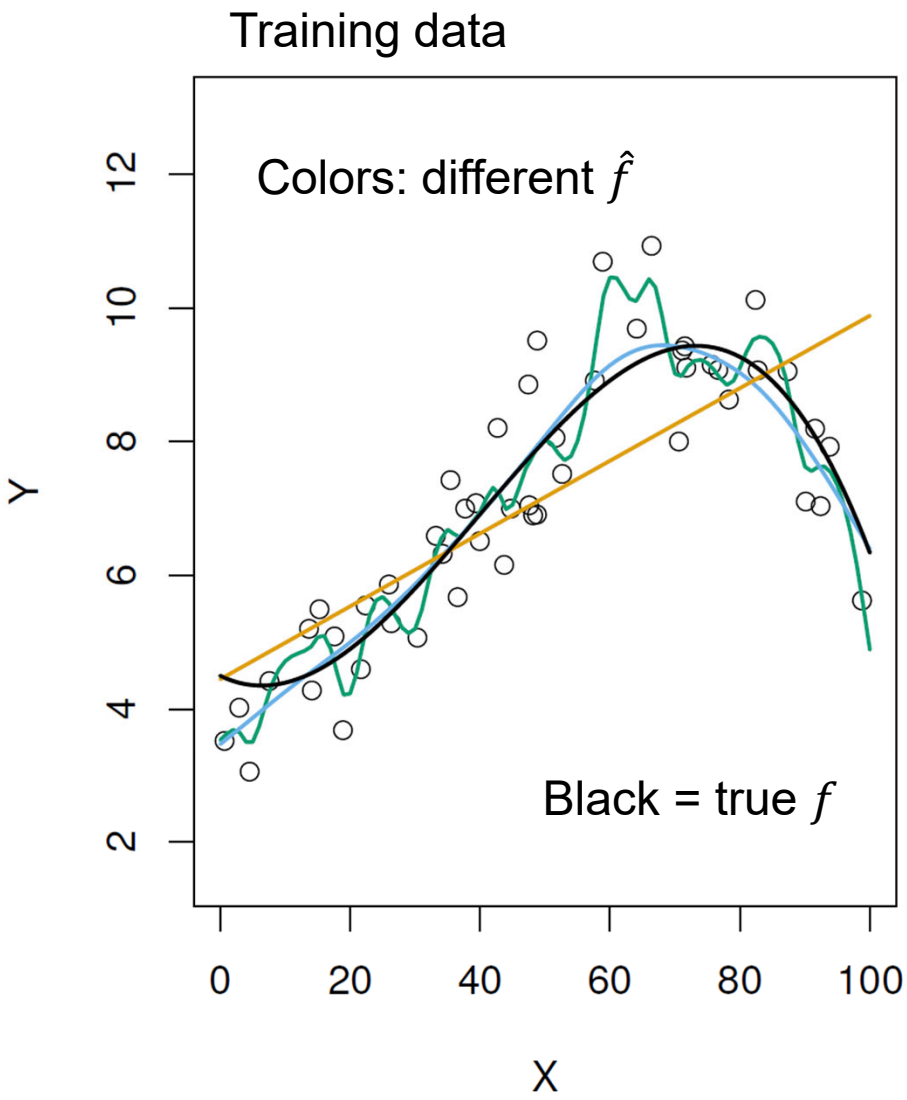
Bias-variance tradeoff

$\hat{f}$ fitted on training data

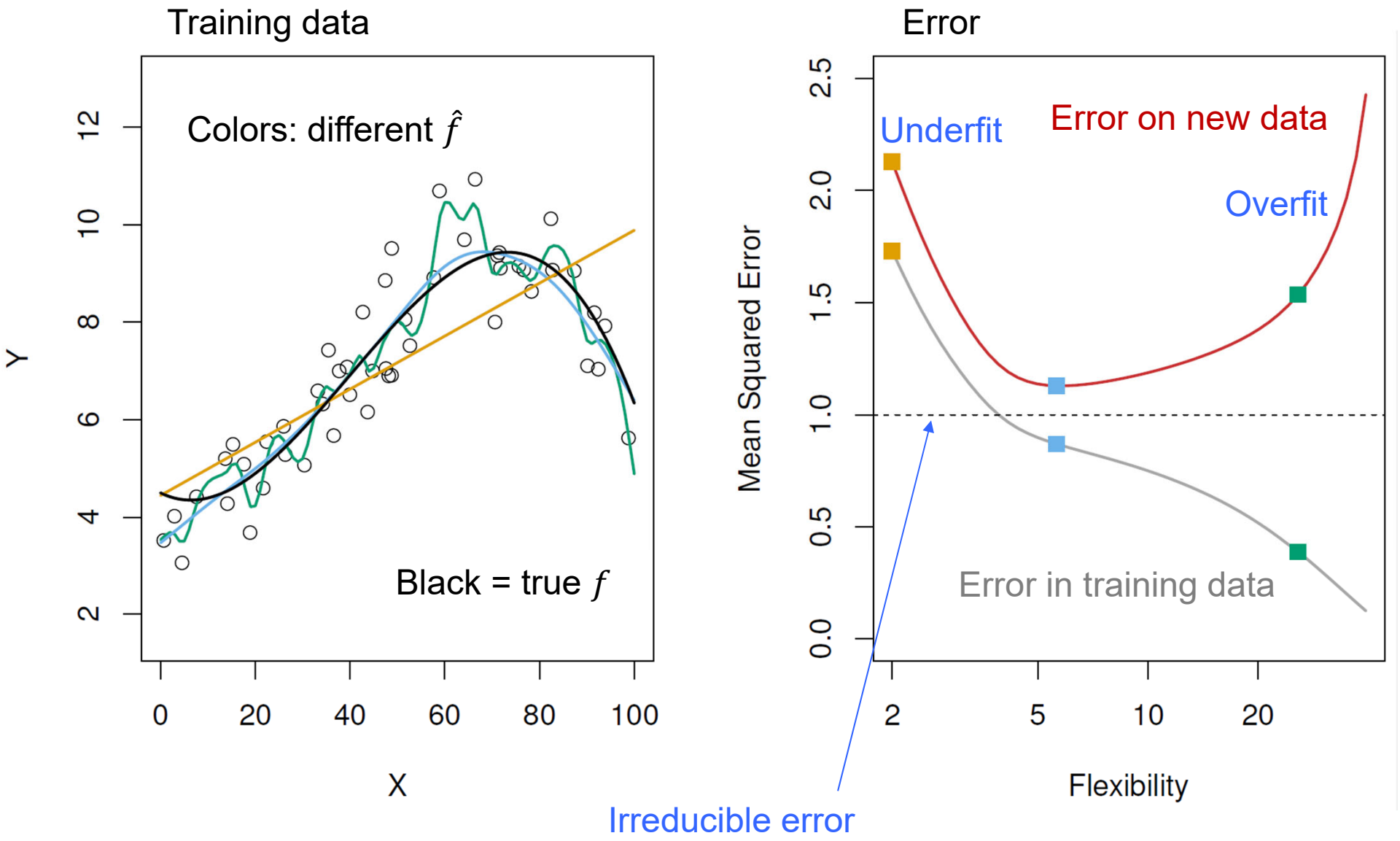


$\hat{f}$ predicting new data





Amount of "wiggliness" in $\hat{f}$



Goal: balance underfit and overfit

Bias-variance tradeoff

$$\text{Total error} = \text{Var}\left(\hat{f}\right) + \text{Bias}\left(\hat{f}\right)^2 + \text{Var}(\epsilon)$$

Amount $\hat{f}$ would change if fitted on different data

Average deviation of $\hat{f}$ from true f

Irreducible error

Bias-variance tradeoff

$$\text{Total error} = \text{Var}(\hat{f}) + \text{Bias}(\hat{f})^2 + \text{Var}(\epsilon)$$

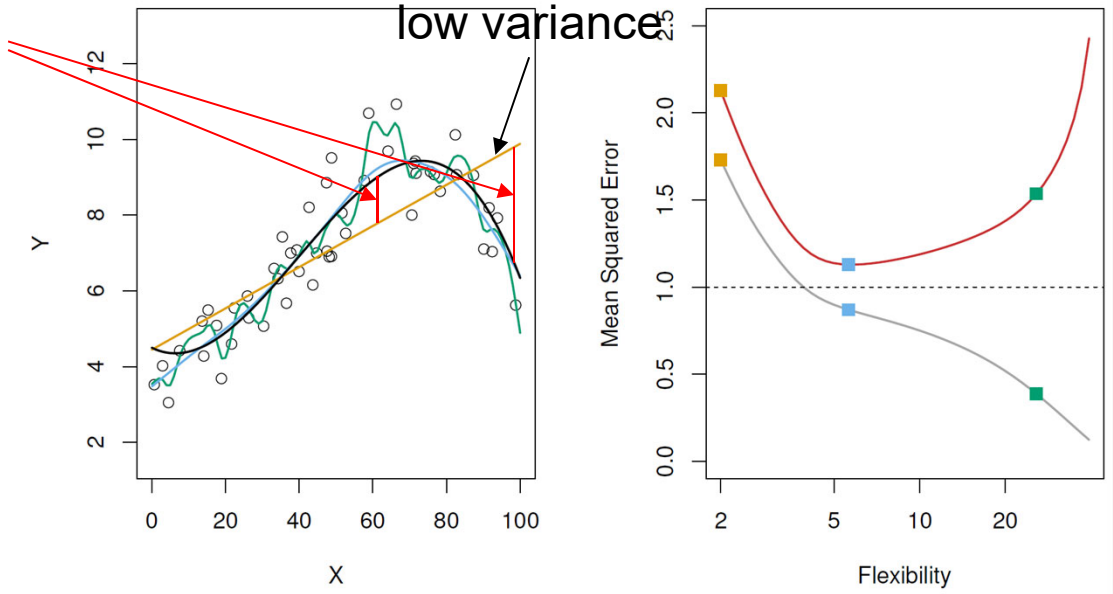
Amount $\hat{f}$ would change if trained on different data

Average deviation of $\hat{f}$ from true f

Irreducible error

bias: consistent deviation from true f across different datasets

High bias/ low variance



Bias-variance tradeoff

$$\text{Total error} = \text{Var}\left(\hat{f}\right) + \text{Bias}\left(\hat{f}\right)^2 + \text{Var}(\epsilon)$$

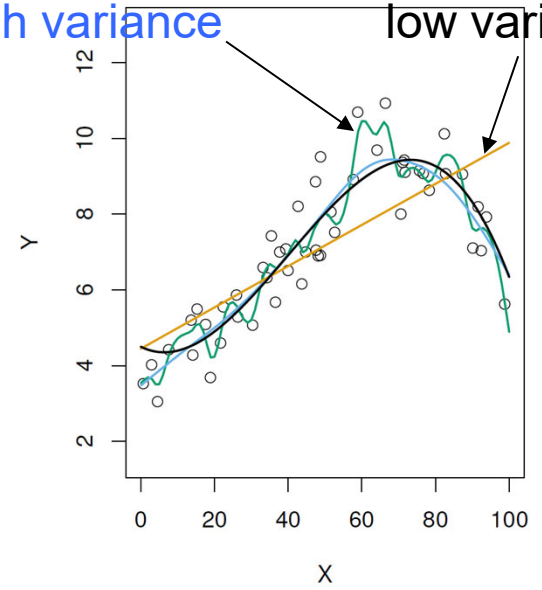
Amount $\hat{f}$ would change if trained on different data

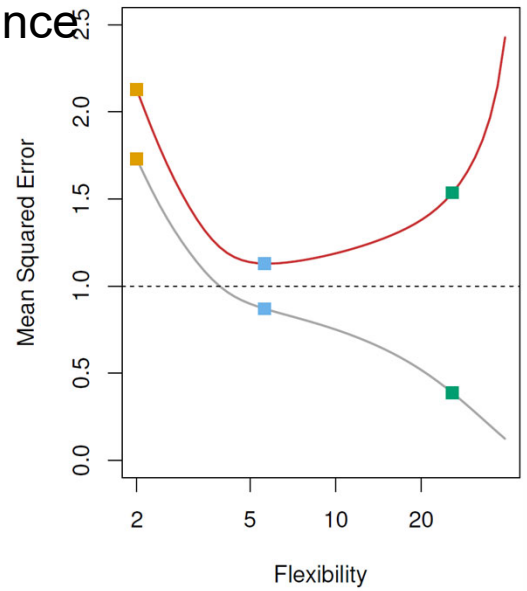
Average deviation of $\hat{f}$ from true f

Irreducible error

Low bias/
high variance

High bias/
low variance





Bias-variance tradeoff

$$\text{Total error} = \text{Var}\left(\hat{f}\right) + \text{Bias}\left(\hat{f}\right)^2 + \text{Var}(\epsilon)$$

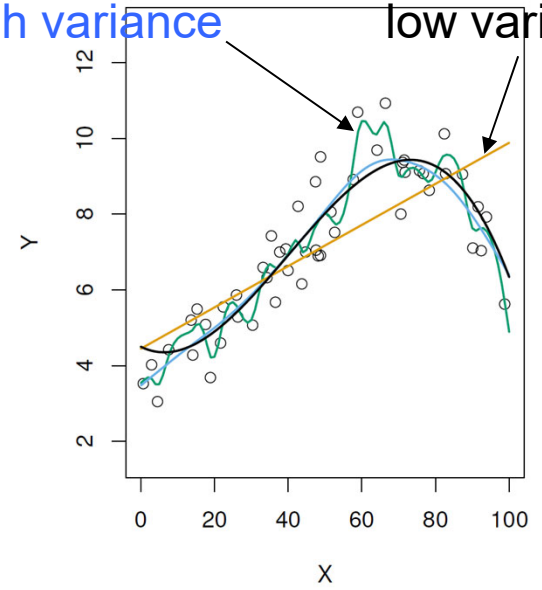
Amount $\hat{f}$ would change if trained on different data

Average deviation of $\hat{f}$ from true f

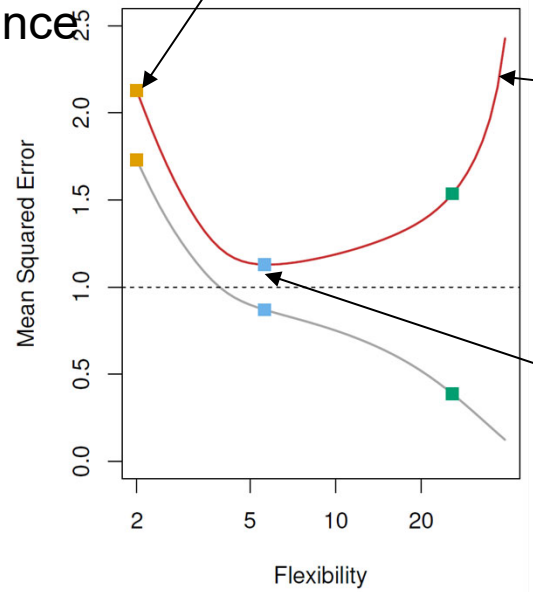
Irreducible error

Low bias/ high variance

High bias/ low variance



High bias/low variance

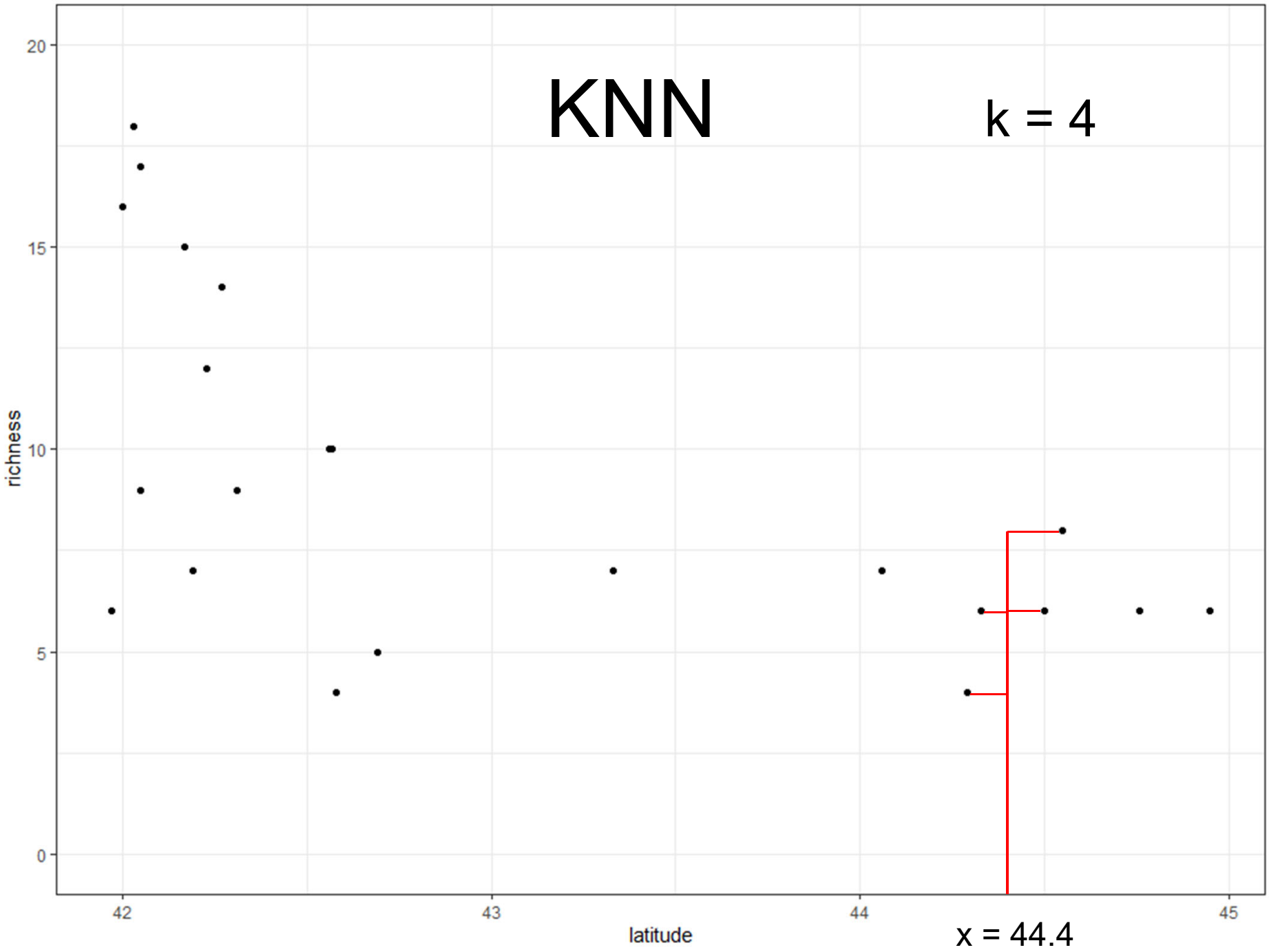


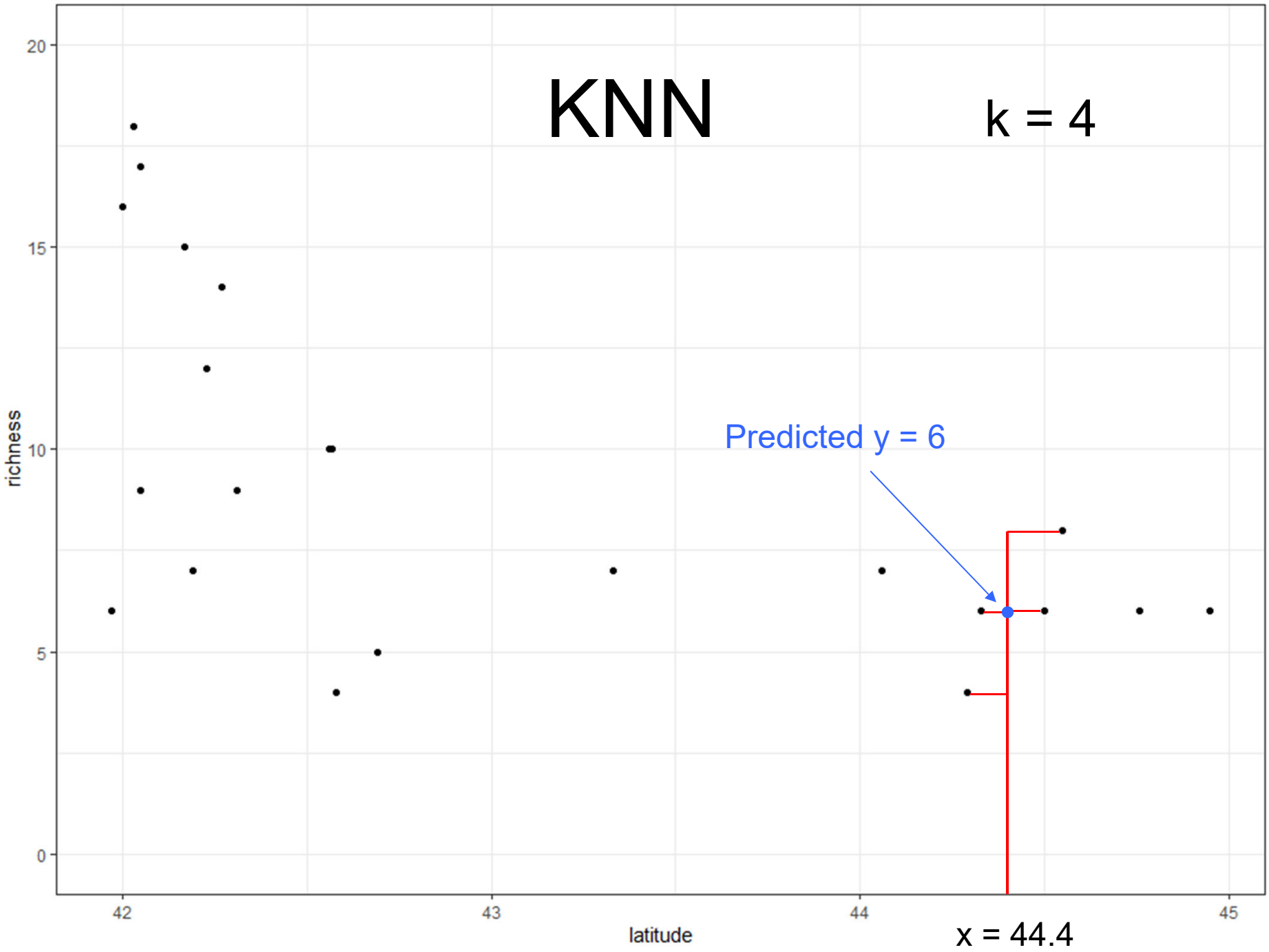
Low bias/ high variance

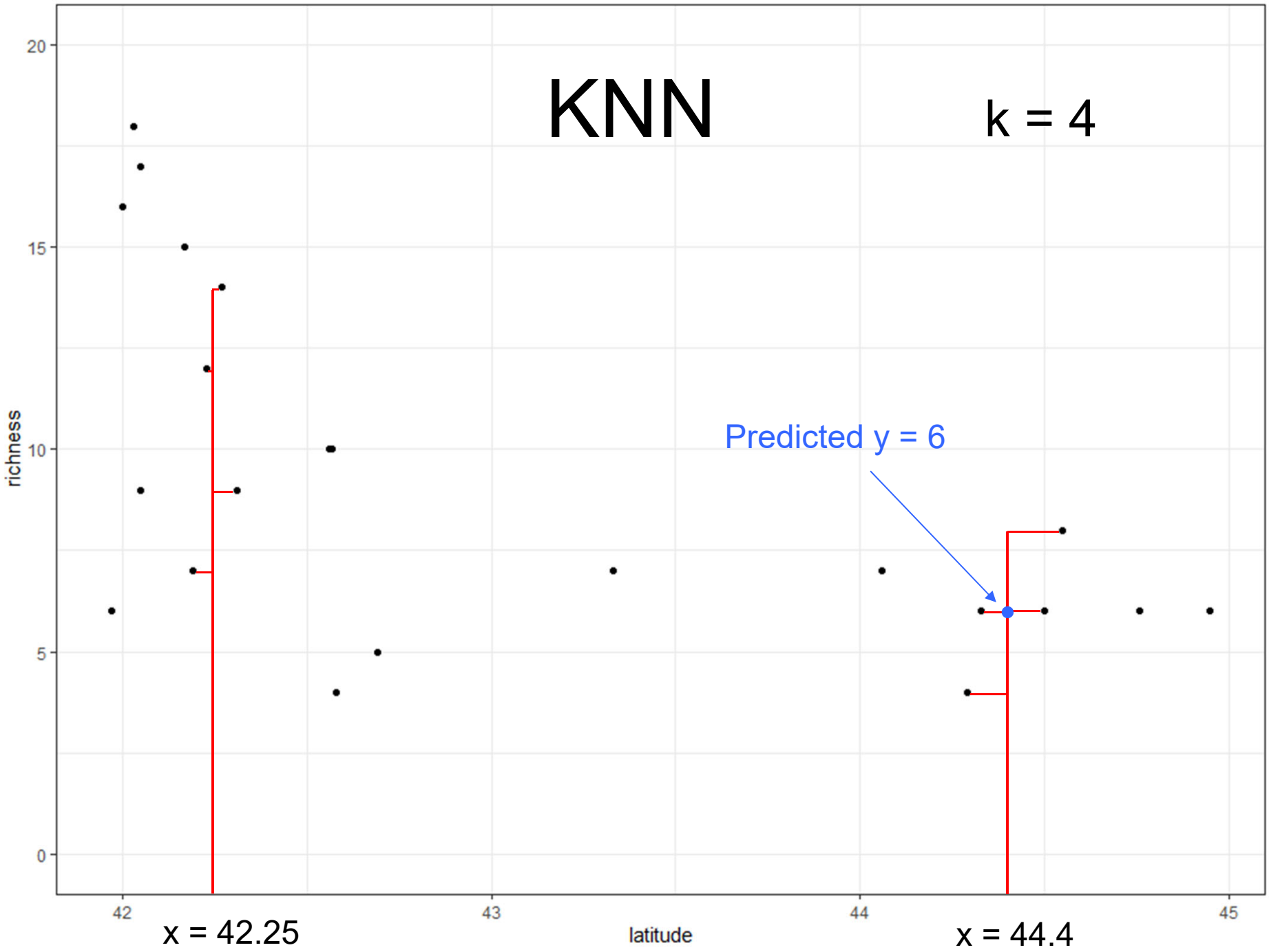
Sweet spot: moderate bias moderate variance

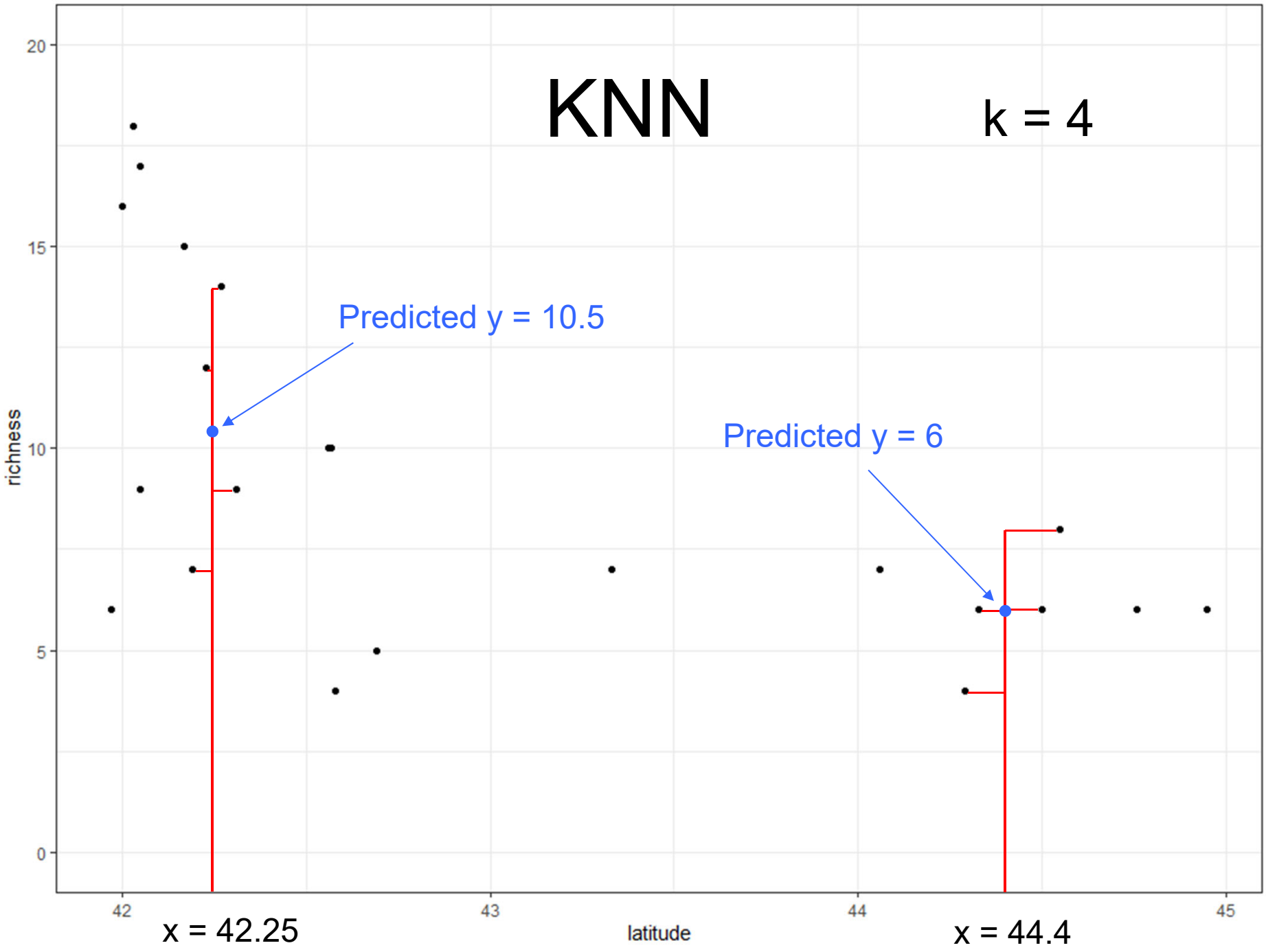
KNN model algorithm

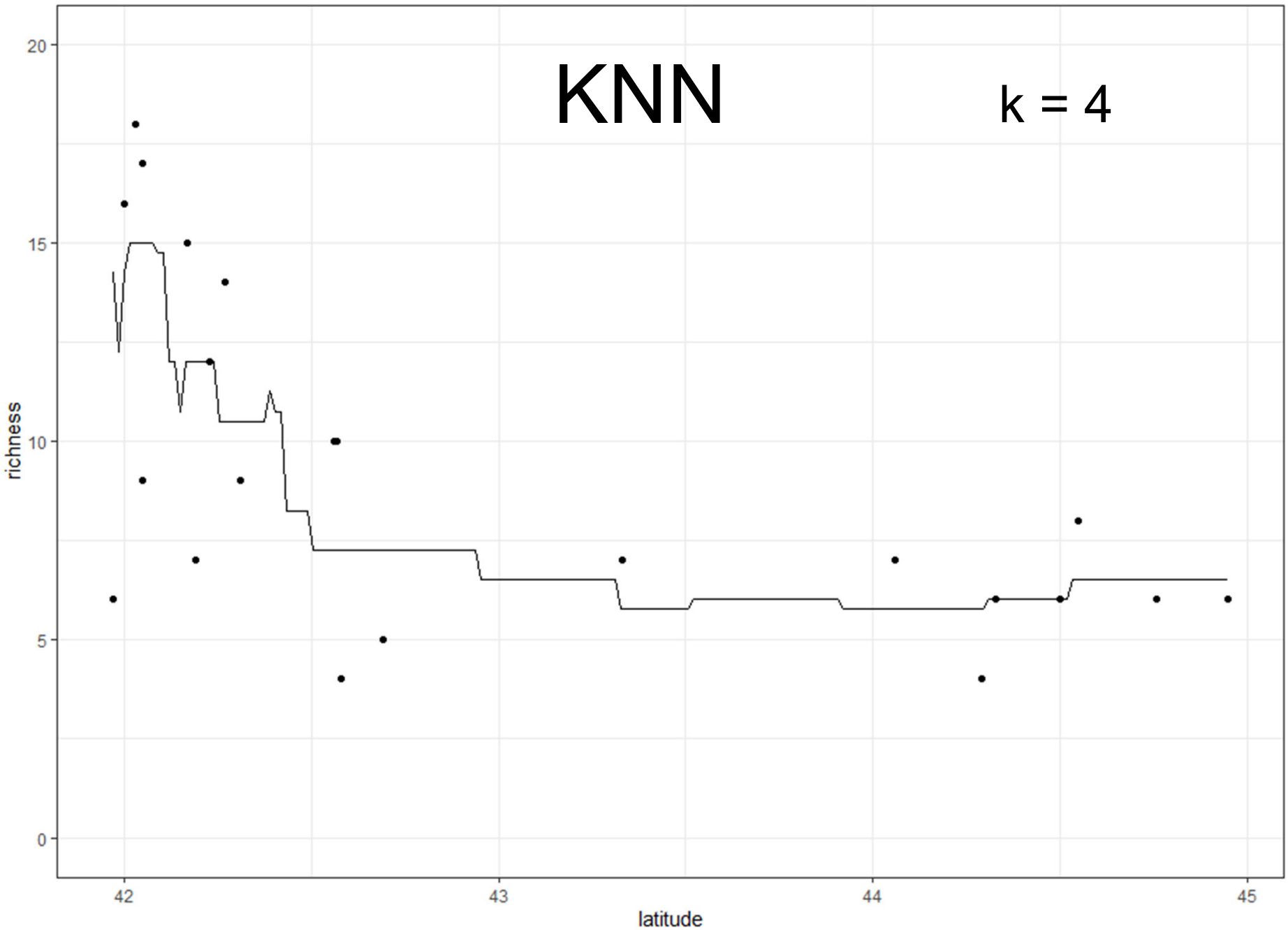
- KNN: k nearest neighbors
- e.g. of a more “algorithmic” model algorithm
- no parameters to be trained
- one tuning parameter

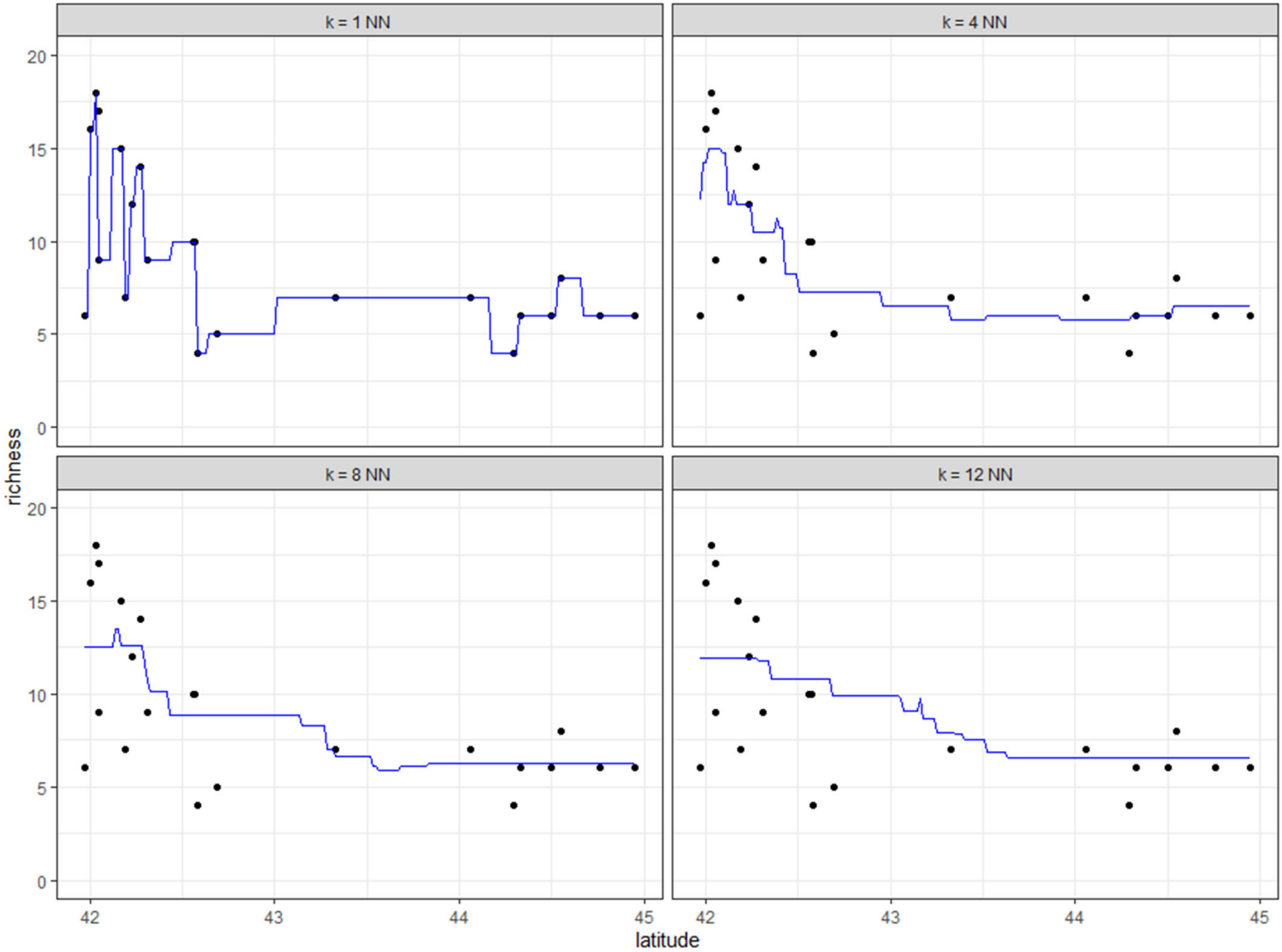

KNN
k = 4
richness
latitude
x = 44.4
20
15
10
5
0
42
43
44
45


KNN
k = 4
Predicted y = 6
x = 44.4
richness
latitude
20
15
10
5
0
42
43
44
45


KNN
k = 4
Predicted y = 6
richness
latitude
20
15
10
5
0
42
43
44
45
x = 42.25
x = 44.4


KNN
k = 4
Predicted y = 10.5
Predicted y = 6
x = 42.25
x = 44.4
richness
latitude
20
15
10
5
0
42
43
44
45


KNN
k = 4
richness
latitude
20
15
10
5
0
42
43
44
45


k = 1 NN
k = 4 NN
k = 8 NN
k = 12 NN
richness
latitude
0
5
10
15
20
42
43
44
45

KNN training algorithm

- There is no training algorithm!
- Automatic
- No parameters

KNN

Algorithm

Set k = number of nearest neighbors
Input (x, y) = x, y data pairs
Input x_new = x value at which to predict y_new
Calculate d = distance of x_new to other x
Sort y data ascending by d; break ties randomly
Predict new y = mean of k nearest neighbors;
i.e. mean of first k values in y_sort

Code

- ants_cv_knn.R
- k-fold CV for KNN models with different numbers of nearest neighbors