Lecture 4: Non-Linear Prediction Methods

Alan Lee

Department of Statistics
STATS 784 Lecture 4

August 4, 2017

Outline

Today's agenda

In the next two lectures we continue our discussion of prediction methods, moving from linear methods to more general non-linear methods. Today we will cover

- smoothing
- gams

We will use the Boston housing data from *Introduction to Statistical Learning* as a running example. Note: references to HTF refer to Hastie, Tibshirani and Friedman, *The Elements of Statistical Learning, 2nd Ed*

The Boston housing data

The Boston housing data may be found in the MASS R package: it consists of data on 506 Boston suburbs. The target is the median value of owner-occupied homes, and there are 13 features. Type

```
> library(MASS)
> data(Boston)
> ?Boston
```

to see more details on the variables. The first two lines are

```
     crim zn indus chas   nox    rm  age    dis rad tax
1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296
2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242
  ptratio black lstat medv
1    15.3 396.9  4.98 24.0
2    17.8 396.9  9.14 21.6
```

Prediction Recap

Here we use data to make predictions about the future.

> *It's tough to make predictions, especially about the future.*

Yogi Berra, New York Yankees

Suppose our data follow a model

$$y = f(x_1, \ldots, x_K) + \epsilon$$

relating a target y to features $x_1, \ldots, x_K$, where f is some unknown smooth function, and K is possibly large.

We want to find an "automatic" estimate of f, to be used to predict future values of the response.

We do this by selecting an estimate of f from some class of functions (for example linear functions, as in the last lecture) by using some goodness of fit criterion, such as least squares.

Training Sets

We assume we have available a "training set", a set of data (usually rectangular) consisting of observations on both the target and potential features (inputs). Our task is to use these data to "train the predictor" i.e. to

- Choose a predictor $\hat{f}$ from some set of functions, hopefully a rich flexible set. (This is called "fitting the model" or "training the predictor". The selected predictor will be an approximation to the true f, hopefully a good one.

- Choose inputs to use from those in the training set. Not all may be used.

There are many possible classes of functions, for example linear functions, neural networks, trees, gams. They should be flexible (approximate the true unknown function well) and reasonably easy to fit.
Having fitted the model, given new data $x_1, \ldots, x_k$ we predict the corresponding value of y by

$$\hat{f}(x_1, \ldots, x_k).$$

Smoothing

If the number of inputs is small, say one or two, we can use smoothing techniques such as loess that is used at stage 2. This is based on the idea that the true relationship f between the inputs and the output is a smooth function.

If the number of inputs is large, we must use other methods and other classes of functions - see in a few slides.

Smoothing: loess

You can read about loess on p280 of ISLR.

- Assume that the number of inputs is small and that the variables are all continuous.
- At each data point, fit a weighted regression, with weights given by a kernel. Points close to the data point under consideration have higher weights. The regression can be quadratic or linear.
- Use regression to predict the fitted value at that point.
- Repeat for every point (subset if too many).
- Implemented using the R function `loess`.

Smoothing splines

Data are $(x_i, y_i), i = 1, \ldots n$, where x_i is of low dimension. (in fact dimension 1 for the rest of the discussion). Choose the function f to minimise

$$\sum_{i=1}^{n}(y_i - f(x_i))^2 + \lambda \int |f''(x)|^2 dx.$$

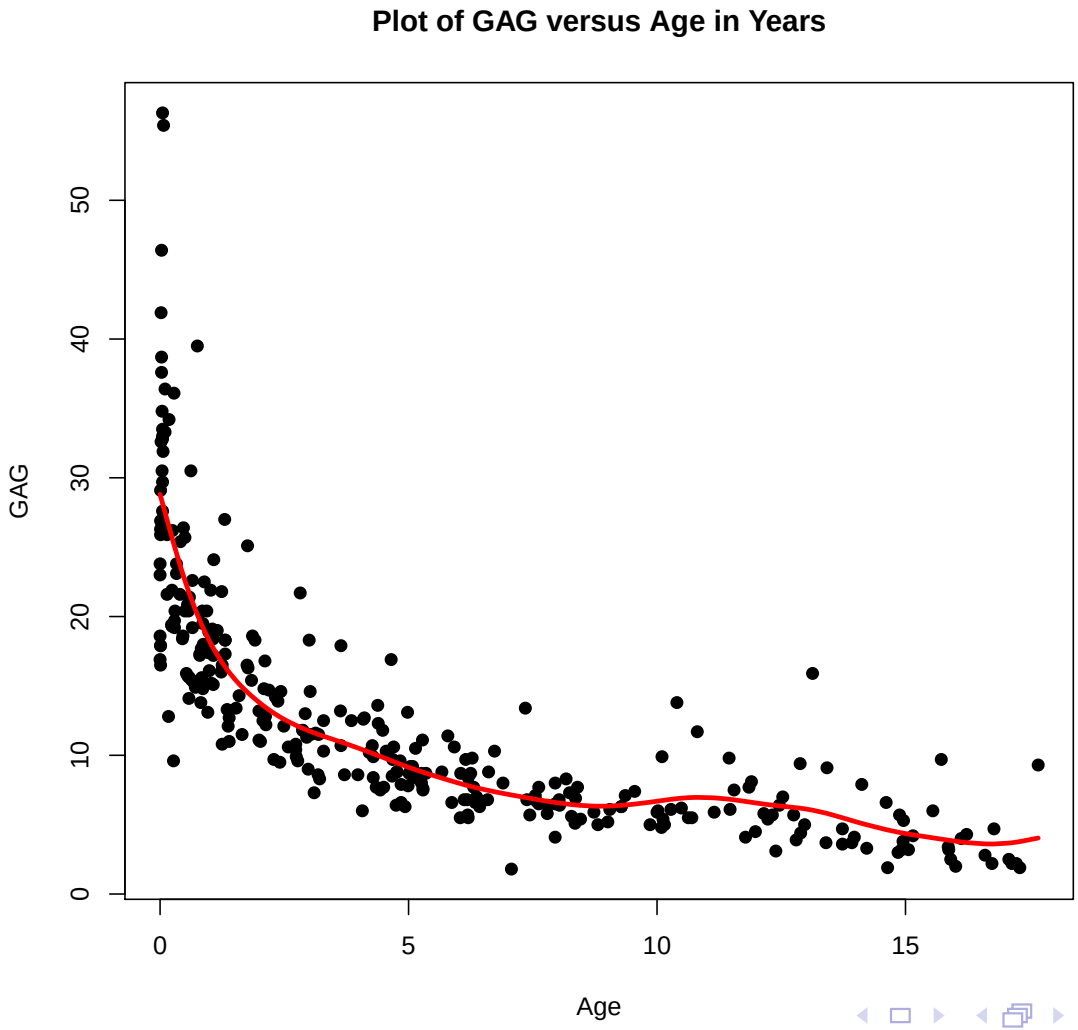
Note:

- The first term measures the goodness of fit: how close f is to the response data.
- The integral measures the size of the second derivative: in other words the smoothness of the function f.
- The parameter λ controls the trade-off between these two requirements of smoothness and goodness of fit, and is chosen by cross-validation.
- The minimising f is a "spline" (a piecewise cubic) with "knots" at every x-value in the data

Example: smoothing splines

```
library(MASS)
data(GAGurine)
plot(GAG~Age, pch=19,data=GAGurine, main =
         "Plot of GAG versus Age in Years")
#use GCV to choose smoothing parameter
smooth = smooth.spline(GAGurine$Age, GAGurine$GAG)
lines(smooth$x, smooth$y, lwd=3, col="red")
```

The fit



Regression splines

- Piecewise cubics
- Choose knots i.e. equally spaced or at percentiles
- Generate a "spline basis" in R
- Fit using linear regression

$$y = b_1 f_1(x) + b_2 f_2(x) + \cdots + b_k f_k(x)$$

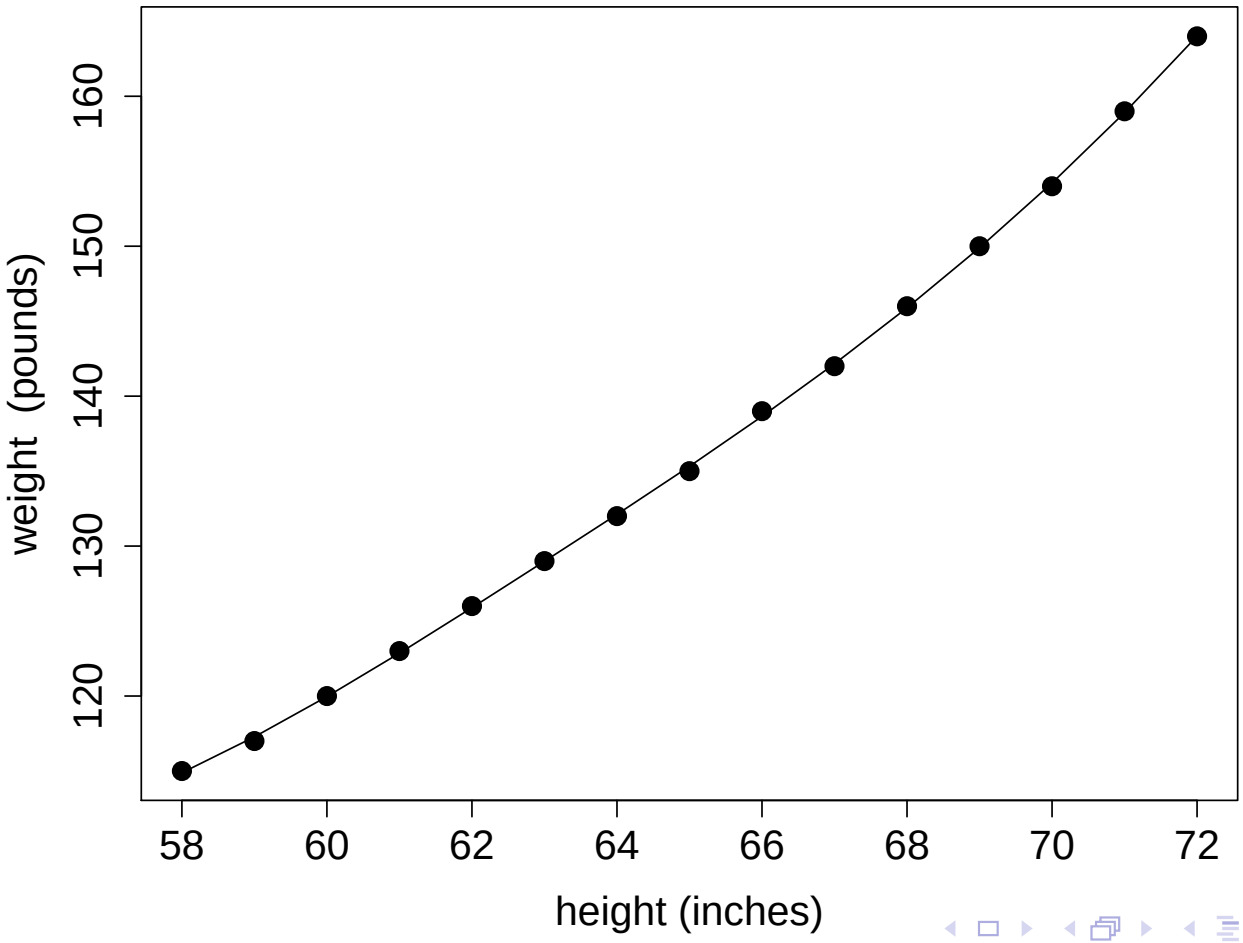
Example: regression splines

```
# Average height and weight of American women 30-39
> women
   height weight
1      58    115
2      59    117
3      60    120
4      61    123
5      62    126
6      63    129
7      64    132
8      65    135
9      66    139
10     67    142
11     68    146
12     69    150
13     70    154
14     71    159
15     72    164
```

R code

```
> round(bs(women$height, df = 5),5)
             1       2       3       4       5
 [1,] 0.00000 0.00000 0.00000 0.00000 0.00000
 [2,] 0.45344 0.05986 0.00164 0.00000 0.00000
 [3,] 0.59694 0.20335 0.01312 0.00000 0.00000
 [4,] 0.53380 0.37637 0.04428 0.00000 0.00000
 [5,] 0.36735 0.52478 0.10496 0.00000 0.00000
 [6,] 0.20016 0.59503 0.20472 0.00009 0.00000
 [7,] 0.09111 0.56633 0.33673 0.00583 0.00000
 [8,] 0.03125 0.46875 0.46875 0.03125 0.00000
 [9,] 0.00583 0.33673 0.56633 0.09111 0.00000
[10,] 0.00009 0.20472 0.59503 0.20016 0.00000
[11,] 0.00000 0.10496 0.52478 0.36735 0.00292
[12,] 0.00000 0.04428 0.37637 0.53380 0.04555
[13,] 0.00000 0.01312 0.20335 0.59694 0.18659
[14,] 0.00000 0.00164 0.05986 0.45344 0.48506
[15,] 0.00000 0.00000 0.00000 0.00000 1.00000
reg.spline<- lm(weight ~ bs(height, df = 5), data = women)
plot(women$height, women$weight)
lines(women$height, predict(reg.spline))
```

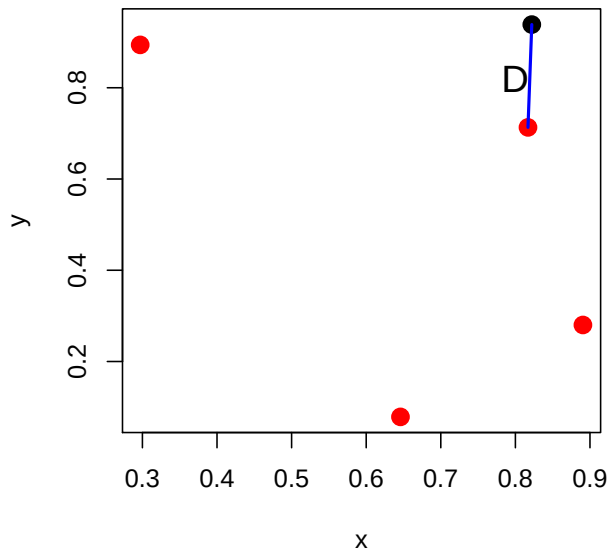
The fit


weight (pounds)
120
130
140
150
160
58
60
62
64
66
68
70
72
height (inches)

Curse of dimensionality

Why doesn't smoothing work in higher dimensions?

- Consider n points scattered at random in a K-dimensional unit cube.
- Let D be the distance between another random point and the closest of the n points. E.g for $d = 2$, $n = 4$:



Median of D

	n=100	n=200	n=500	n=1000	Max
dim = 1	0.005	0.002	0.001	0.001	1.000
dim = 2	0.052	0.036	0.023	0.016	1.414
dim = 5	0.298	0.250	0.207	0.178	2.236
dim = 10	0.674	0.614	0.555	0.505	3.162
dim = 20	1.221	1.157	1.094	1.045	4.472
dim = 100	3.504	3.444	3.378	3.332	10.000

Conclusions

- Smoothing doesn't work in high dimensions: points are too far apart, and smoothing works by averaging responses of cases whose covariates are close together.
- Solution: pick estimate of f from a class of functions that is flexible enough to match true f reasonably closely.
- We need to be able to compute the estimate easily.

Classes of functions and models

We will consider the following:

- Linear functions (see last lecture)
- Additive functions (additive models)
- Projection Pursuit
- MARS
- CART (regression trees)
- Neural networks

Generalised Additive Models (gams)

In the regression (i.e. continuous target) case, these are functions of the form

$$f(x) = \alpha + \phi_1(x_1) + \cdots + \phi_k(x_k),$$

where the ϕ's are smooth functions that are to be selected. We fit the model by the backfitting algorithm, that relies on smoothing in one dimension at a time. This allows us to avoid the curse of dimensionality, at the expense of a certain lack of flexibility. Note that to make α unique, we require that the averages of the ϕ's are zero.

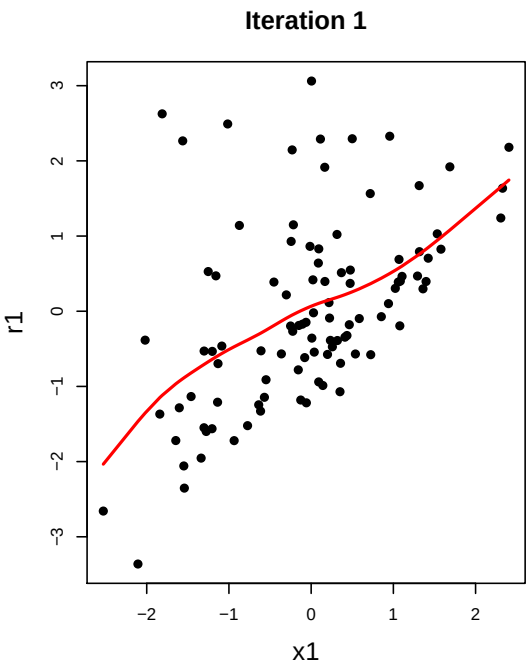
The backfitting algorithm

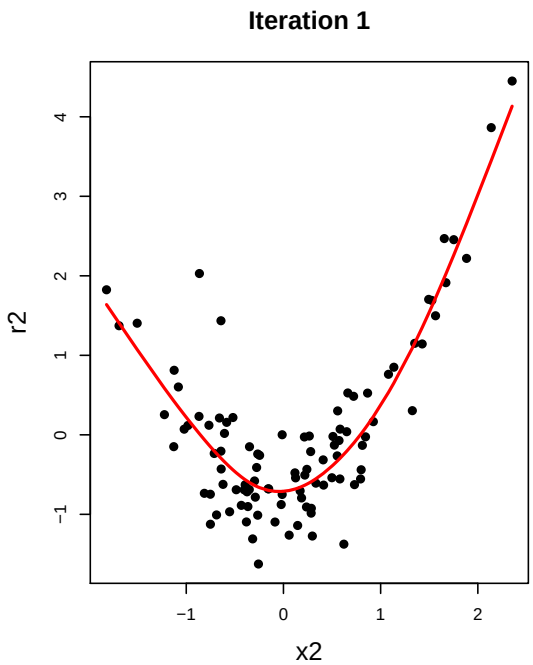
Illustrated for three features:

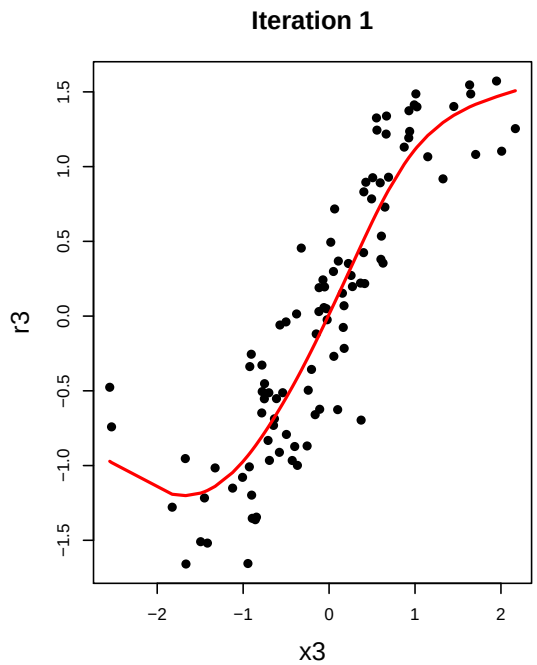
- Step 1: Set $\alpha = \overline{y}$, $\phi_2(x) = x$, $\phi_3(x) = x$.
- Step 2: Calculate $r_1 = y - \phi_2(x) - \phi_3(x)$.
- Step 3: Plot r_1 versus x_1 and smooth the plot using your favourite smoother. Let $\hat{r}_1$ be the smoothed version. Our first guess for ϕ_1 is $\hat{r}_1 - \overline{\hat{r}_1}$.
- Step 4: Set $r_2 = y - \phi_1(x) - \phi_3(x)$ and smooth as above to get $\phi_2 = \hat{r}_2 - \overline{\hat{r}_2}$.
- Step 5: Set $r_3 = y - \phi_1(x) - \phi_2(x)$ and smooth as above to get $\phi_3 = \hat{r}_3 - \overline{\hat{r}_3}$.

Repeat steps 2–4 until no further change.

Example: first iteration

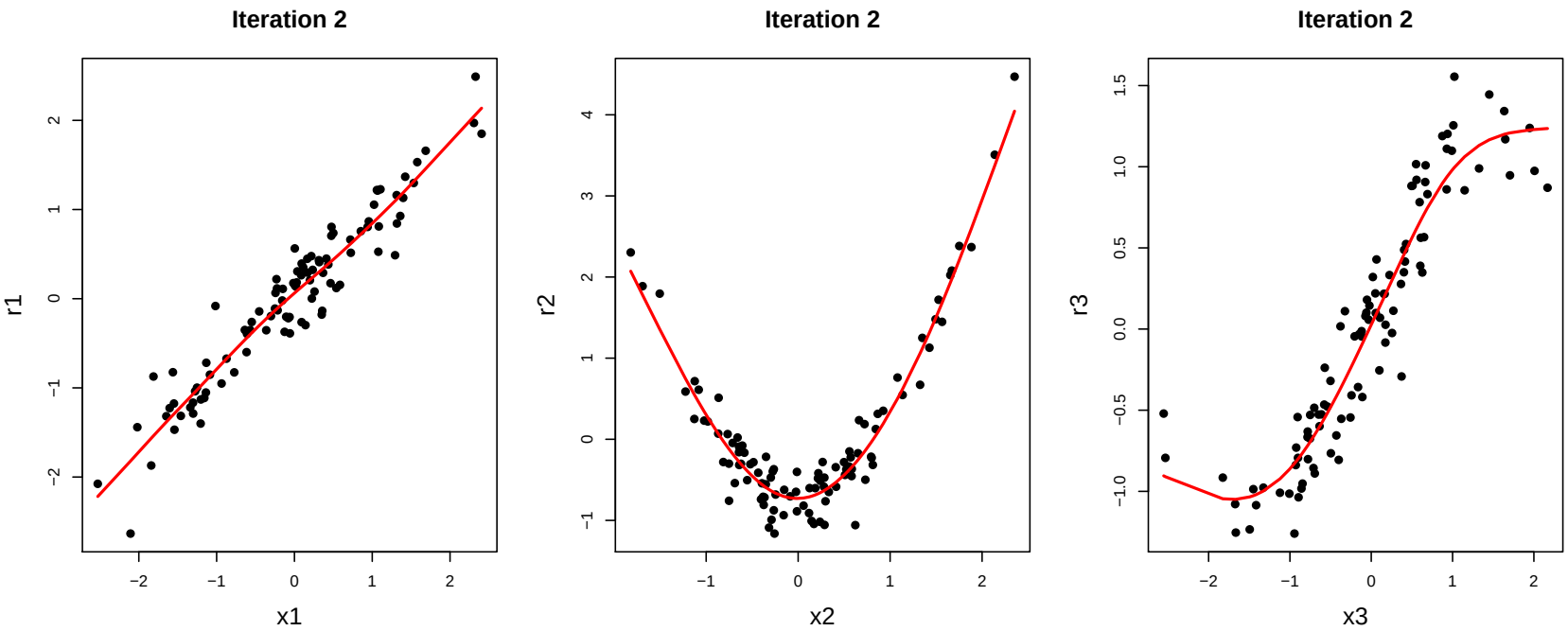






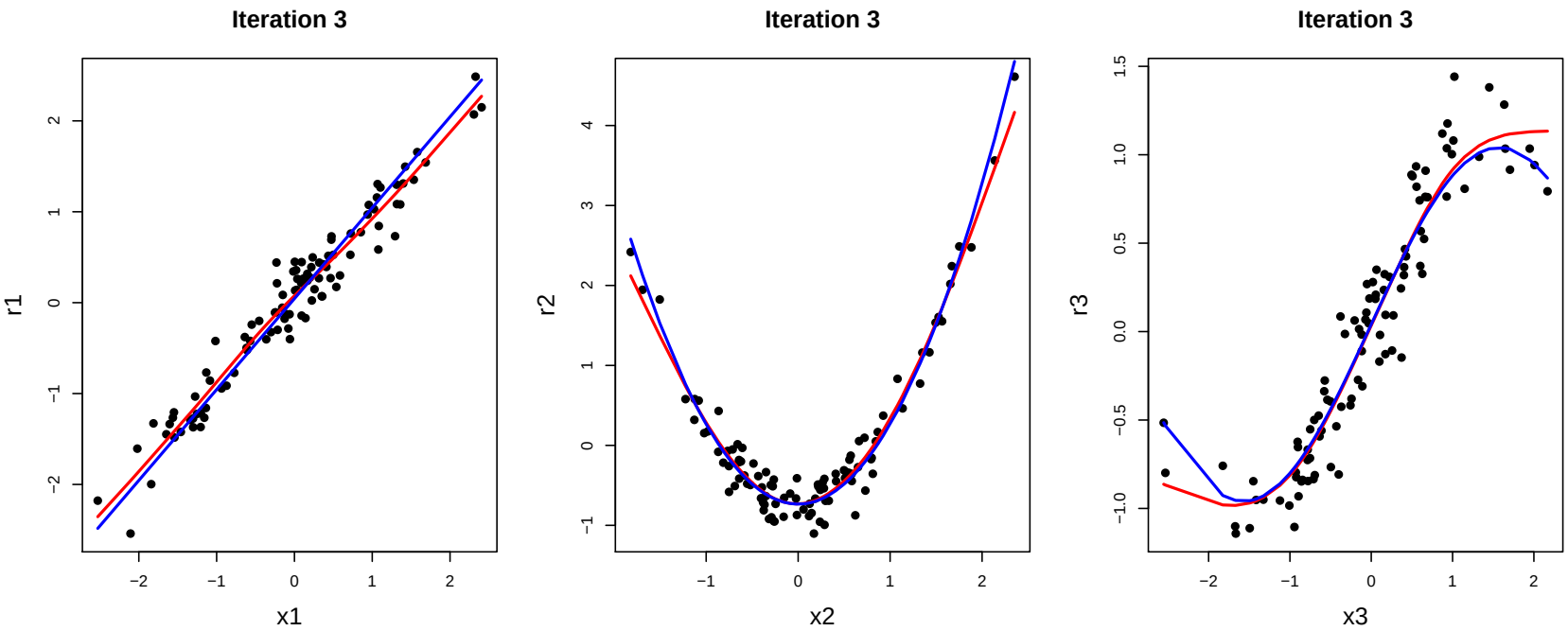
Red lines are smooth estimates of ϕ_1, ϕ_2, ϕ_3.

Example: second iteration



Red line are smooth estimates of ϕ_1, ϕ_2, ϕ_3.

Example: third iteration



Red lines are smooth estimates of ϕ_1, ϕ_2, ϕ_3, blue lines are the actual functions: the model $y = x1 + x_2^2 + sin(x_3) + error$ was used to generate the data.

Example: Boston housing data

```
library(MASS)
data(Boston)
library(mgcv)
# use logs as target
gam.stuff = gam(log(medv) ~ s(crim) + s(zn) + s(indus) +
          factor(chas) + s(nox) + s(rm) + s(age) + s(dis) +
          factor(rad) + s(tax) +s(ptratio) + s(black) +
          s(lstat),
          data=Boston, family=gaussian())
summary(gam.stuff)
```

Boston housing data (cont)

```
Formula:
log(medv) ~ s(crim) + s(zn) + s(indus) + factor(chas) + s(nox) +
    s(rm) + s(age) + s(dis) + factor(rad) + s(tax) + s(ptratio) +
    s(black) + s(lstat)

Parametric coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)    2.824106   0.061884  45.636  < 2e-16 ***
factor(chas)1  0.043694   0.027655   1.580 0.114828
factor(rad)2  -0.001444   0.052593  -0.027 0.978107
factor(rad)3   0.104128   0.047967   2.171 0.030472 *
factor(rad)4   0.043428   0.042143   1.030 0.303339
factor(rad)5   0.071866   0.044981   1.598 0.110815
factor(rad)6   0.036157   0.052403   0.690 0.490560
factor(rad)7   0.131475   0.056039   2.346 0.019404 *
factor(rad)8   0.133793   0.056912   2.351 0.019162 *
factor(rad)24  0.618078   0.181891   3.398 0.000739 ***
```

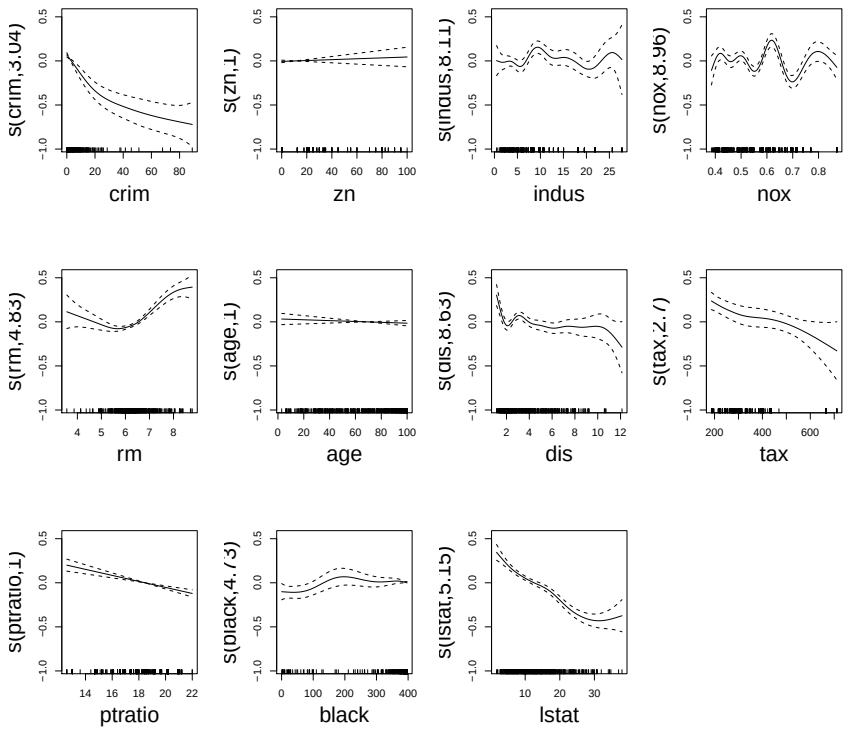
Boston housing data (cont)

```
Approximate significance of smooth terms:
             edf Ref.df      F  p-value
s(crim)    3.041  3.795 26.812  < 2e-16 ***
s(zn)      1.000  1.000  0.642 0.423241
s(indus)   8.108  8.680  3.776 0.000165 ***
s(nox)     8.961  8.997 20.004  < 2e-16 ***
s(rm)      4.827  5.972 17.004  < 2e-16 ***
s(age)     1.000  1.000  1.000 0.317835
s(dis)     8.630  8.957  5.299 8.06e-07 ***
s(tax)     2.701  3.374  8.742 4.90e-06 ***
s(ptratio) 1.000  1.000 34.930 6.58e-09 ***
s(black)   4.733  5.736  2.740 0.013430 *
s(lstat)   5.155  6.321 34.123  < 2e-16 ***
R-sq.(adj) =  0.885   Deviance explained = 89.8%
GCV = 0.021815  Scale est. = 0.019264  n = 506
```

The $R^2$is 89% - a linear model has an R^2 of 79%

Boston housing data (cont)

```
par(mfrow=c(3,4))
plot(gam.stuff)
```



Effective degrees of freedom

For both smoothing splines (with a fixed λ) and regression splines, the fitted values ($\hat{y}$) and and the target are related by the equation

$$\hat{y} = Sy,$$

where S is a matrix that depends on x and λ but not on y.

The trace (i.e. sum of the diagonal elements) of S is called the **effective degrees of freedom** (edf) of the smooth. This is by analogy with linear regression where the same thing happens: here the trace is the model degrees of freedom. The EDF relates to how smooth the fitted function is: an EDF of 1 means the function is essentially linear, an edf of 2 means the function is roughly quadratic and so on.

Prediction

To use the gam predictor to predict a future value, we use the same code as we did for linear models. In fact, all the prediction methods we discuss use identical code. Suppose we want to predict the median price of a house in a suburb whose fetaures are in a data frame newHouse:

```
> newHouse
       crim zn indus chas   nox    rm  age
222 0.40771  0   6.2    1 0.507 6.164 91.3
  dis rad tax ptratio  black lstat
3.048   8 307    17.4 395.24 21.46
> newHousePred = predict(gam.stuff, newdata=newHouse)
> exp(newHousePred)
18.71334
```

Recall that we modelled log median price, and the median prices