# Chapter 7

# Flexible regression and classification methods

## 7.1 Introduction

The curse of dimensionality makes it virtually impossible to estimate a regression function $m(x) = \mathbb{E}[Y|X = x]$ or the probability functions $\pi_j(x) = \mathbb{P}[Y = j|X = x]$ (for classification) in a full nonparametric way without making some structural assumptions.

We are going to describe some flexible models and methods which are of nonparametric nature but making some structural assumptions. In the sequel, we will denote either a regression function $\mathbb{E}[Y|X = x]$ or the logit transform in a binary classification problem $\log(\pi(x)/(1 - \pi(x))$ by

$$g(\cdot) : \mathbb{R}^p \to \mathbb{R}.$$

## 7.2 Additive models

Additive models require additivity of the function: the model is

$$\begin{aligned} g_{add}(x) &= \mu + \sum_{j=1}^{p} g_j(x_j),\ \mu \in \mathbb{R} \\ g_j(\cdot) &: \mathbb{R} \to \mathbb{R}, \quad \mathbb{E}[g_j(X_j)] = 0 \ \ (j = 1, \ldots, p). \end{aligned}$$

The functions $g_j(\cdot)$ are fully nonparametric. The requirement about $\mathbb{E}[g_j(X_j)] = 0$ yields an identifiable model; note that otherwise we could add and subtract constants into the one-dimensional functions $g_j(\cdot)$.

Additive models are generalizations of linear models. While the additive functions are very general, they do not allow for interaction terms such as $g_{j,k}(x_j, x_k)$[1]. The curse of dimensionality is *not* present in additive models. When assuming continuous second derivatives for all the functions $g_j$, it can be shown that some estimators $\hat{g}_{add}(\cdot)$ have mean square error rate $O(n^{-4/5})$ as for estimating a single one-dimensional ($\mathbb{R} \to \mathbb{R}$) function.

[1] as a matter of fact, both the original Hastie-Tibshirani and the new algorithm in `mgcv` allow for bivariate interactions, optionally

### 7.2.1 Backfitting for additive regression models

Backfitting is a very general tool for estimation in additive (and other) structures. We can use "any" nonparametric smoothing technique for estimating one-dimensional functions. A smoother against the predictor variables $X_{1j}, \ldots X_{nj}$ is denoted by the hat operator

$$\mathcal{S}_j : (U_1, \ldots, U_n)^\intercal \mapsto (\hat{U}_1, \ldots, \hat{U}_n)^\intercal \ (j = 1, \ldots p)$$

for any response vector $(U_1, \ldots, U_n)^\intercal$. The subscript $j$ indicates that smoothing is done against the $j$th predictor variable. For example, $\mathcal{S}_j$ could be smoothing splines with the same degrees of freedom for all $j$, e.g. equal to 5 or estimated by (generalized) cross-validation. Or they could be Nadaraya-Watson Gaussian kernel estimators with the same bandwidth which may be estimated by (generalized) cross-validation.

Backfitting for additive regression then works as follows.

1. Use $\hat{\mu} = n^{-1} \sum_{i=1}^{n} Y_i$. Start with $\hat{g}_j(\cdot) \equiv 0$ for all $j = 1, \ldots, p$.

2. Cycle through the indices $j = 1, 2, \ldots, p, 1, 2, \ldots, p, 1, 2, \ldots$ while computing

$$\hat{\mathbf{g}}_j = \mathcal{S}_j(\mathbf{Y} - \hat{\mu}\mathbf{1} - \sum_{k \neq j} \hat{\mathbf{g}}_k),$$

   where $\mathbf{Y} = (Y_1, \ldots, Y_n)^\intercal$ and $\hat{\mathbf{g}}_j = (\hat{g}_j(X_{1j}), \ldots, \hat{g}_j(X_{nj}))^\intercal$. Stop the iterations if the individual functions $\hat{g}_j(\cdot)$ do not change much anymore, e.g.,

$$\frac{\|\hat{\mathbf{g}}_{j,new} - \hat{\mathbf{g}}_{j,old}\|_2}{\|\hat{\mathbf{g}}_{j,old}\|_2} \leq \text{tol}$$

   where tol is a tolerance such as $10^{-6}$.

3. Normalize the functions

$$\tilde{g}_j(\cdot) = \hat{g}_j(\cdot) - n^{-1} \sum_{i=1}^{n} \hat{g}_j(X_{ij}).$$

We may view backfitting as a method to optimize a high-dimensional (penalized) parametric problem. For example, with smoothing spline smoothers, we have seen in chapter 3 that the smoothing spline fit can be represented in terms of basis functions and we have to solve a penalized parametric problem. Backfitting can then be viewed as a coordinate-wise optimization method which optimizes one coordinate (corresponding to one predictor variable) at a time while keeping all others (corresponding to all other predictors) fixed. This coordinate-wise optimization may be slow but backfitting is a very general overall method which directly allows to use one-dimensional smoothing algorithms.

### 7.2.2 Additive model fitting in R

Additive models can be fitted in R with the function `gam` (**g**eneralized **a**dditive **m**odel) from package `mgcv`. The term "generalized" allows also to fit additive logistic regression, among other things.

The function `gam` uses for the smoothers $\mathcal{S}_j$ a penalized regression spline: i.e., a spline with selected knots including a penalty term (this is somewhat different than a smoothing spline). Interestingly, the function will choose the degrees of freedom, which may be different for every fitted function $\hat{g}_j(\cdot)$, via generalized cross-validation: in particular, this allows to use more degrees of freedom for "complex" functions and few degrees of freedom for functions which seem "simple".

We consider as an example the daily ozone concentration in the Los Angeles basin as a function of 9 predictor variables. The commands and output in R look as follows.

```
library(mgcv)
data(ozone, package = "gss")
d.ozone ← ozone; colnames(d.ozone)[c(1,2,3,4)] ← ..... # ‘‘better’’ names
pairs(d.ozone, pch = ".", gap = 0.1) #  --> Scatterplot Matrix
```

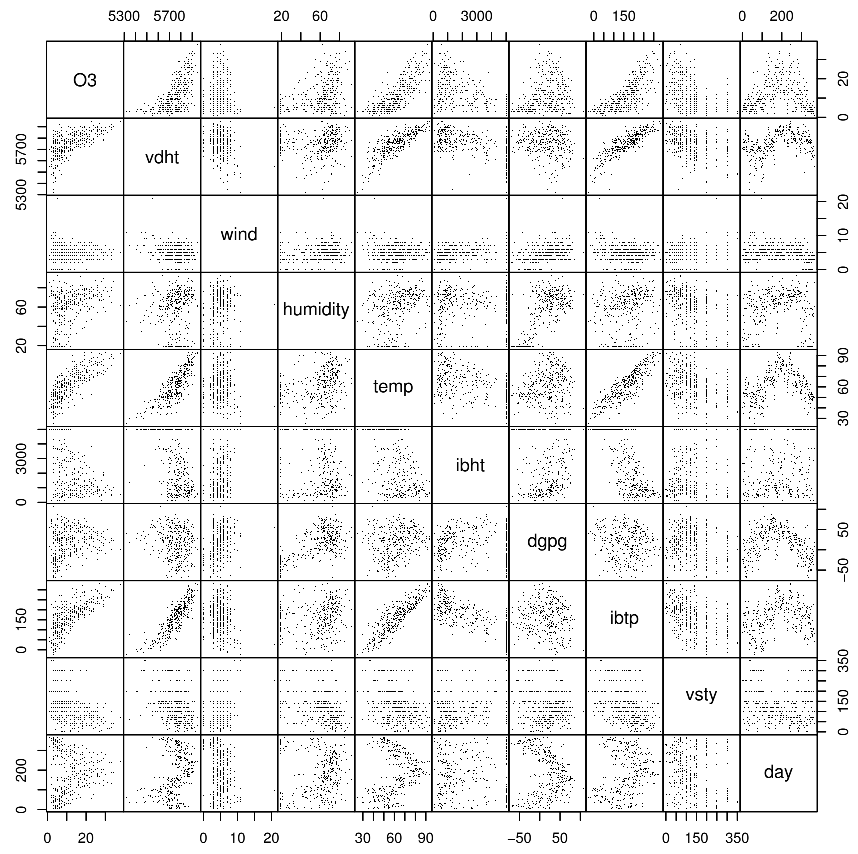

Figure 7.1: Daily ozone concentration in the Los Angeles basin as a function of 9 predictor variables.

```
fit ←
 gam(O3 ~ s(vdht)+s(wind)+s(humidity)+s(temp)+s(ibht)+s(dgpg)+s(ibtp)+s(vsty)+s(day),
     data = d.ozone)
summary(fit)



            . . . . . . . .

      Parametric coefficients:
                 Estimate Std. Error t value Pr(>|t|)
      (Intercept)  11.7758     0.1988   59.22   <2e-16 ***
```

```
Approximate significance of smooth terms:
              edf Est.rank      F  p-value
s(vdht)     1.000        1  9.360  0.00242 **
s(wind)     1.000        1  4.375  0.03729 *
s(humidity) 3.631        8  2.595  0.00933 **
s(temp)     4.361        9  4.694 7.56e-06 ***
s(ibht)     3.043        7  1.708  0.10658
s(dgpg)     3.230        7  7.916 7.94e-09 ***
s(ibtp)     1.939        4  1.809  0.12698
s(vsty)     2.232        5  3.825  0.00225 **
s(day)      4.021        9 10.174 1.04e-13 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

R-sq.(adj) =  0.797   Deviance explained = 81.2%
GCV score = 14.137   Scale est. = 13.046    n = 330
```

The column `edf` shows the estimated (via GCV) degrees of freedom: if they are equal to 1, a linear straight line is fitted. We get an indication about the relevance of a predictor

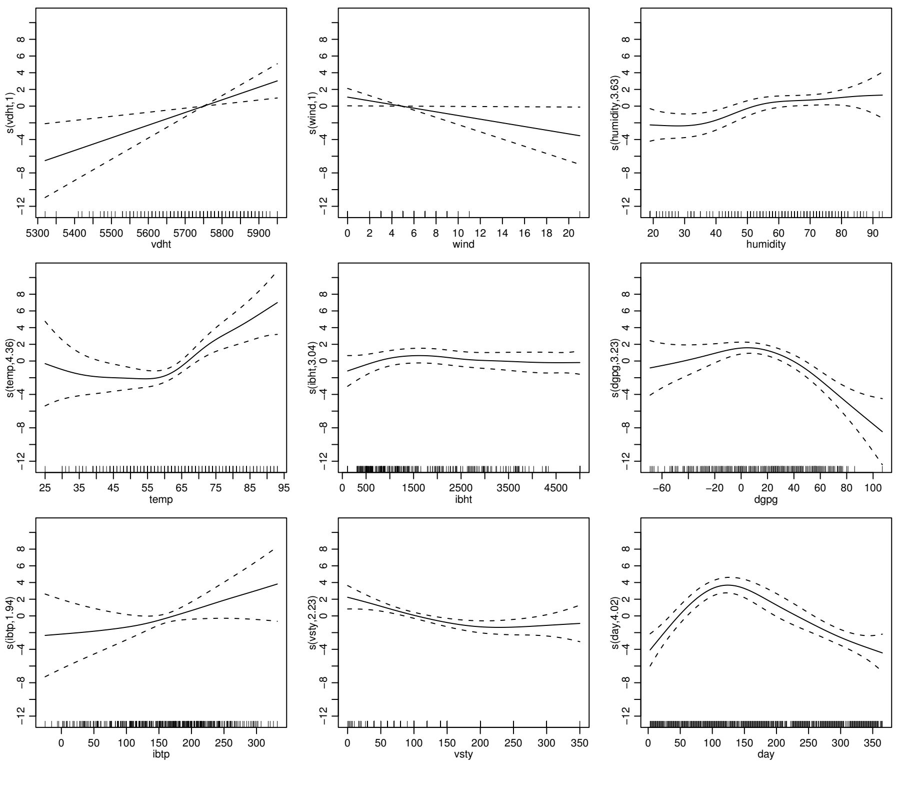

Figure 7.2: Estimated function $\hat{g}_j$ for the ozone data set.

either from Figure 7.2 or from the $P$-values in the summary output from R .

The fit of the model can be checked via residual analysis as in chapter 1. The Tukey-Anscombe plot indicates heteroscedastic errors. We thus try the log-transform for the response variable and re-do the whole analysis. Also, we omit observation (no. 72) which is an outlier wrt. wind speed. The results are given below. Further data analysis would

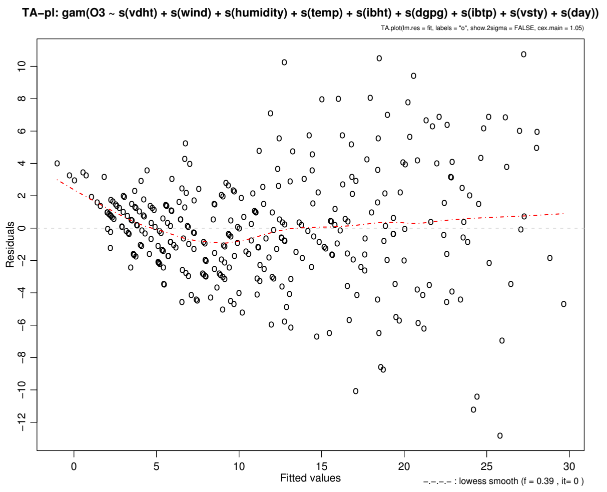

Figure 7.3: Tukey-Anscombe plot for the ozone data set.

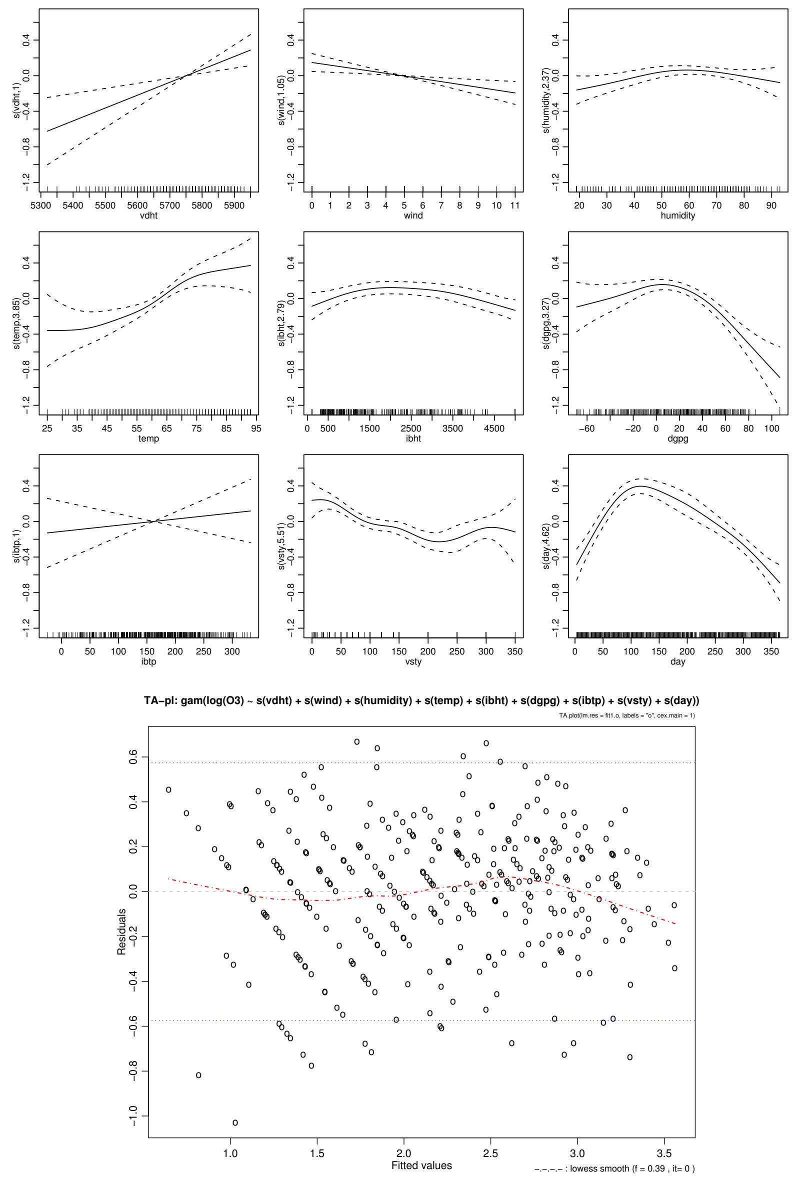

Figure 7.4: Estimated additive functions and Tukey-Anscombe plot for the ozone dataset with `log(O3)` as response variable and after omitting one outlier.

now also consider dropping non-significant variables, possible two-way interactions, etc. This is one motivation for the following section.

## 7.3 MARS

MARS is a shortcut for **m**ultivariate **a**daptive **r**egression **s**plines. MARS is an adaptive procedure for regression and it is often useful for high-dimensional problems with many predictor variables.

MARS uses expansions in piecewise linear basis functions of the form

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

and $(d - x_j)_+$. The value $d$ is a knot and $x_j$ ($j \in \{1, \dots, p\}$) is the $j$-th component of $x \in \mathbb{R}^p$. The pair $(x_j - d)_+$, $(d - x_j)_+$ is called a *reflected pair*. Note that we use $(x_j - d)_+$ as a function of $x$ ($\in \mathbb{R}^p$) where only the $j$th component of $x$ is relevant. The collection of reflected pairs is then a set of basis functions

$$\mathcal{B} = \{(x_j - d)_+, (d - x_j)_+; \qquad d \in \{x_{1,j}, x_{2,j}, \dots, x_{n,j}\},\ j \in \{1, \dots, p\}\},$$

where we note that in the univariate case ($p = 1$), $\mathcal{B}$ spans the space of continuous piecewise linear functions, i.e., linear splines.

MARS employs now a forward selection of reflected pairs of basis functions in $\mathcal{B}$ and their products. The model has the form

$$g(x) = \mu + \sum_{m=1}^{M} \beta_m h_m(x),$$

where each function $h_m(\cdot)$ is a function from the basis $\mathcal{B}$ or a product of functions from $\mathcal{B}$. The model building technique is going in a forward way as follows:

1. Start with the function $h_0(x) \equiv 1$. Initialize the model set $\mathcal{M} = \{h_0(\cdot) \equiv 1\}$. Fit the function $h_0$ by least squares regression, yielding the estimate $\hat{\mu} = n^{-1} \sum_{i=1}^{n} Y_i$.

2. For $r = 1, 2, \dots$ do the following:
   Search for the best pair of functions $((h_{2r-1}(\cdot), h_{2r}(\cdot))$ which are of the form

$$\begin{aligned} h_{2r-1}(\cdot) &= h_\ell(\cdot) \times (x_j - d)_+, \\ h_{2r}(\cdot) &= h_\ell(\cdot) \times (d - x_j)_+, \end{aligned} \tag{7.1}$$

   for some $h_\ell$ in the model set $\mathcal{M}$, and some basis functions in $\mathcal{B}$. The best pair of functions is defined to be the one which reduces residual sum of squares most.
   The model fit is then

$$\hat{g}(x) = \hat{\mu} + \sum_{m=1}^{2r} \hat{\beta}_m h_m(x),$$

   where the coefficients $\hat{\beta}_m$ are estimated by least squares.
   Enlarge the model set in every iteration (with index $r$) by

$$\mathcal{M} = \mathcal{M}_{old} \cup \{h_{2r-1}(\cdot), h_{2r}(\cdot)\},$$

   with the functions $h_{2r-1}, h_{2r}$ from (7.1).

3. Iterate step 2 until a large enough number of basis functions $h_m(\cdot)$ has been fitted.
4. Do backward deletion ("pruning"), allowing to remove single functions from a pair $h_{2r-1}(\cdot), h_{2r}(\cdot)$; i.e., of all single basis functions, delete the one which increases the residual sum of squares the least.
5. Stop the backward deletion by optimizing a GCV score.

For example, the trace of solutions could look as follows:

$$\begin{aligned}
h_0(x) &= 1, \ \ \mathcal{M} = \{1\}, \\
h_1(x) &= (x_2 - x_{72})_+, \ \ h_2(x) = (x_{72} - x_2)_+, \quad \mathcal{M} = \{1, (x_2 - x_{72})_+, (x_{72} - x_2)_+\}, \\
h_3(x) &= (x_{72} - x_2)_+ \cdot (x_1 - x_{51})_+, \ \ h_4(x) = (x_{72} - x_2)_+ \cdot (x_{51} - x_1)_+, \\
& \quad \mathcal{M} = \{1, (x_2 - x_{72})_+, (x_{72} - x_2)_+, \ (x_{72} - x_2)_+ \cdot (x_1 - x_{51})_+, \ h_4(x) \ \} \\
& \ldots
\end{aligned}$$

### 7.3.1 Hierarchical interactions and constraints

It becomes clear from the definition of MARS, that the algorithm proceeds hierarchically in the sense that a $d$-order interaction can only enter the model when an interaction of degree $d-1$ involving one predictor less is already in the model. For example, an interaction of degree 4 involving $x_2, x_4, x_7, x_9$ enters the model because an interaction of degree 3 involving the predictors $x_2, x_4, x_7$ is already in the model.

Quite often it is useful to restrict interactions to degree 2 or 3. Sometimes, we may even restrict the interaction degree to 1: MARS then yields an additive model where the predictor variables and the piecewise linear spline basis functions are included in a forward-backward adaptive way.

### 7.3.2 MARS in R

MARS has been implemented in S and R in the function `mars` from the package `mda`. A new, slightly more flexible alternative implementation is in package `earth` with function `earth` and syntax

```
> fit <- earth(Volume ~ ., data = trees)
> summary(fit)
.....
Expression:
  23.20824
  +  5.745962 * pmax(0,  Girth -   12.9)
  -  2.866452 * pmax(0,   12.9 -  Girth)
  + 0.7183364 * pmax(0, Height -     76)

Number of cases: 31
Selected 4 of 5 terms, and 2 of 2 predictors
Number of terms at each degree of interaction: 1 3 (additive model)
GCV: 11.48697        RSS: 213.4354        GRSq: 0.958859       RSq: 0.9736697

> predict(fit,
          data.frame(Girth= 5:15, Height= seq(60,80, length=11)))
.........
```

## 7.4 Neural Networks

Neural networks have been very popular in the 90's in the machine learning and artificial intelligence communities. From a statistical perspective, they can be viewed as high-dimensional nonlinear regression models.

We will focus here on feedforward neural nets with one hidden layer. The model is

$$g_k(x) \quad = \quad f_0\left(\alpha_k + \sum_{h=1}^{q} w_{hk}\phi\big(\tilde{\alpha}_h + \sum_{j=1}^{p} \tilde{w}_{jh}x_j\big)\right). \tag{7.2}$$

which is predicting multivariate $(g_k)_k$ where for regression only $g_0$ is used and for classification, one typically uses $g_0, \ldots, g_{J-1}$ and $\hat{g}(x) := \arg\max_j \hat{g}_j(x)$ which is called "softmax" in the NN literature. The function $\phi(\cdot)$ is usually the sigmoid function

$$\phi(x) = \frac{\exp(x)}{1+\exp(x)},$$

whereas $f_0(\cdot)$ is typically chosen as the identity for regression and as the sigmoid $\phi(x)$ for softmax classification. The $w_{hk}, \tilde{w}_{jh}, \alpha_k, \tilde{\alpha}_h$ all are unknown parameters. The so-called

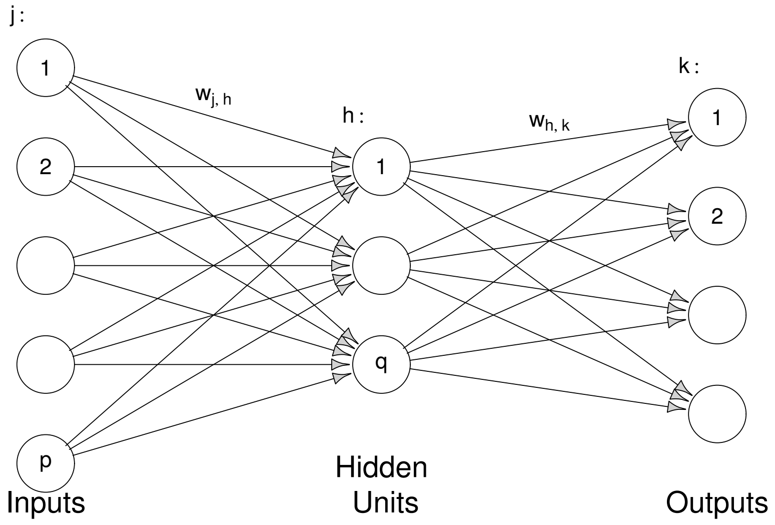

Figure 7.5: *Simple "feedforward" neural net with one hidden layer.*

input layer consists of the $p$ predictor variables; the values $w_{hk}\ \phi\left(\tilde{\alpha}_h + \sum_{j=1}^{p} \tilde{w}_{jh}x_j\right)$ $(h = 1, \ldots, q)$ make the so-called hidden layer with $q$ units. And the so-called output layer is just a single unit for univariate regression.

A useful variant of (7.2) is a model which includes a linear regression component:

$$g(x) = f_0\left(\alpha + \sum_{j=1}^{p} w_{j,lin}x_j + \sum_{k=1}^{q} w_k\phi(\alpha_k + \sum_{j=1}^{p} w_{jk}x_j)\right). \tag{7.3}$$

### 7.4.1 Fitting neural networks in R

Feedforward neural nets can be fitted in R with the function `nnet` from the package `nnet`; the algorithm is numerically maximizing the likelihood and basically equivalent to using `optim(*, method="BFGS")`.

In practice, it can be very important to center and scale all the predictor variables so that they are approximately on the same scale (about "1"). This avoids that gradient methods for optimizing the likelihood get stuck in the "flat regions" of the sigmoid functions. Further note that one can regularize the problem using so called *weight decay* which stabilizes the algorithm (less dependence on random starting values $w_{*,*}$) and diminishes the importance of choosing the number of hidden units $q$.

```
library(nnet)
## Using 'd.ozone' data from above, using  log(O3)
## and  scaling the x-variables (to mean = 0, sd = 1):
sc.ozone <- data.frame(scale(d.ozone)[, -1],
                       log.O3 = log(d.ozone[,"ozone"]))

> set.seed(22)  ## (also try others; nnet() uses random starting values!)
> fit <- nnet(log.O3 ~ . , data= sc.ozone, size = 3, skip = TRUE,
+             decay = 4e-4, linout = TRUE, maxit = 500)
 # weights:  43
initial  value 2943.804833
iter  10 value 568.319707
......
......
final  value 34.631788
converged
> sum(residuals(fit)^2) # -> 34.211
> summary(fit)
 a 9-3-1 network with 43 weights
a 9-3-1 network with 43 weights
options were - skip-layer connections  linear output units  decay=4e-04
 b->h1 i1->h1 i2->h1 i3->h1 i4->h1 i5->h1 i6->h1 i7->h1 i8->h1 i9->h1
 -2.28   1.27  -0.34  -2.57   1.46   0.03   0.10  -1.02  -0.39  -0.33
 b->h2 i1->h2 i2->h2 i3->h2 i4->h2 i5->h2 i6->h2 i7->h2 i8->h2 i9->h2
-12.43   5.08   2.04   8.19  -7.66  -7.01   2.40  -0.31   3.58  -1.19
 b->h3 i1->h3 i2->h3 i3->h3 i4->h3 i5->h3 i6->h3 i7->h3 i8->h3 i9->h3
-19.79  -6.65   1.49  -4.53  -3.95   2.28   6.06   5.19  10.06  -0.20
  b->o  h1->o  h2->o  h3->o  i1->o  i2->o  i3->o  i4->o  i5->o  i6->o
  2.50  -1.81   0.68   0.71   0.11  -0.09  -0.49   0.72   0.01  -0.03
 i7->o  i8->o  i9->o
  0.03  -0.29  -0.15


## without linear model component: skip=FALSE
set.seed(22)
> fit1 <- nnet(log.O3 ~ . , data= sc.ozone, size = 3, skip=FALSE,
+              decay = 4e-4, linout = TRUE, maxit = 500)
 ..........
final  value 42.909
converged
> sum(residuals(fit1)^2) # 41.865
> summary(fit1)
 a 9-3-1 network with 34 weights
options were - linear output units  decay=4e-04
 b->h1 i1->h1 i2->h1 i3->h1 i4->h1 i5->h1 i6->h1 i7->h1 i8->h1 i9->h1
```

```
-27.22   8.91    0.51    8.78  -8.26  10.55  -8.46  11.89 -11.51   8.51
 b->h2 i1->h2 i2->h2 i3->h2 i4->h2 i5->h2 i6->h2 i7->h2 i8->h2 i9->h2
  0.09  -0.15  -0.07   0.07   1.49  -0.43   0.05  -0.41  -0.01  -0.24
 b->h3 i1->h3 i2->h3 i3->h3 i4->h3 i5->h3 i6->h3 i7->h3 i8->h3 i9->h3
-16.97   6.53   0.56   2.30 -20.19   0.64   6.34  17.51  -1.67  -3.21
  b->o  h1->o  h2->o  h3->o
  0.86   0.46   2.42   0.86
```

`linout=TRUE` indicates that the function is fitted on the linear regression scale; for classification, we use the default `linout=FALSE`. `size = 3` chooses 3 hidden units, regularized by weight `decay = 4e-4`: these are tuning parameters, and `skip=TRUE` enforces a neural net with a linear model component. The random seed is stored because, by default, `nnet()` uses random starting values for the high-dimensional optimization.

## 7.5 Projection pursuit regression

Projection pursuit regression (for regression problems) bears some similarities to feedforward neural networks. Instead of taking a linear combination of $q$ different sigmoid function outputs (from the $q$ hidden units in the hidden layer) in (7.2), we use the following model:

$$\begin{aligned} g_{\text{PPR}}(x) &= \mu + \sum_{k=1}^{q} f_k(\sum_{j=1}^{p} \alpha_{jk} x_j), \quad \text{where} \\ & \qquad \textstyle\sum_{j=1}^{p} \alpha_{jk}^2 = 1, \quad \mathbb{E}[f_k(\displaystyle\sum_{j=1}^{p} \alpha_{jk} X_j)] = 0, \quad \text{for all } k. \end{aligned}$$

The functions $f_k(\cdot) : \mathbb{R} \to \mathbb{R}$ are nonparametric (i.e. "arbitrary" smooth); the linear combinations $\sum_{j=1}^{p} \alpha_{jk} x_j$ are linear projections: For the unit vector $\boldsymbol{\alpha}_k = (\alpha_{1k}, \dots, \alpha_{pk})^\intercal$, $\boldsymbol{\alpha}_k{}^\intercal x$ is the projection of $x$ onto (a ray through $\vec{0}$ in direction) $\boldsymbol{\alpha}_k$. Note that the function $x \mapsto f_k(\sum_{j=1}^{p} \alpha_{jk} x_j)$ only varies along the direction $\boldsymbol{\alpha}_k$, hence the $f_k$'s are called *ridge functions* and the model "*projection pursuit*". This model typically requires much smaller $q$ than the hidden units in a neural network, at the expense of estimating ridge functions rather than using fixed sigmoid functions in neural nets.

Estimation of projection pursuit can be done using a backfitting algorithm. Quite often, projection pursuit yields very competitive prediction performance when choosing a reasonable number of ridge functions (e.g. by optimizing a CV score).

### 7.5.1 Projection pursuit regression in R

The function `ppr` in R can be used for fitting projection pursuit regression. The function in `ppr` re-scales the projection pursuit model to

$$\begin{aligned} & g_{\text{PPR}}(x) = \mu + \sum_{k=1}^{q} \beta_k f_k(\sum_{j=1}^{p} \alpha_{jk} x_j), \\ & \sum_{j=1}^{p} \alpha_{jk}^2 = 1, \quad \mathbb{E}[f_k(\sum_{j=1}^{p} \alpha_{jk} X_j)] = 0, \quad \operatorname{Var}(f_k(\sum_{j=1}^{p} \alpha_{jk} X_j)) = 1, \quad \text{for all } k. \end{aligned}$$

Consider the ozone data set from above.

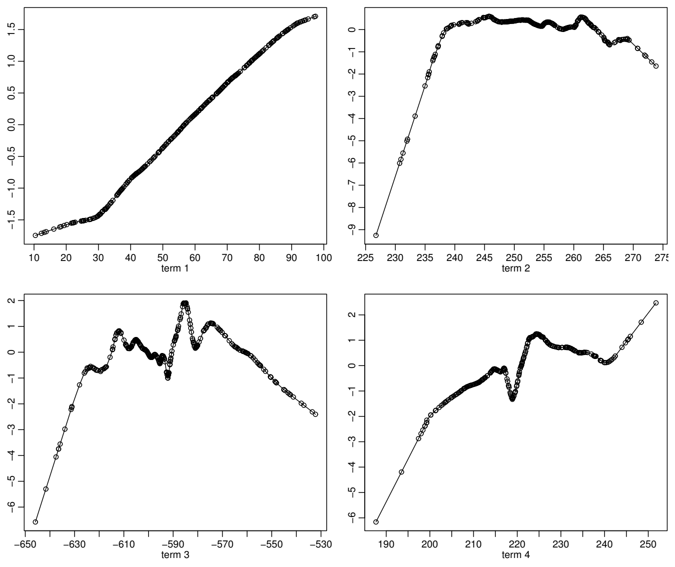

Figure 7.6: Four estimated ridge functions $f_k(\cdot)$ from a projection pursuit fit to the ozone data (`log(O3) ~ .`).

```
fit <- ppr(log(O3) ~ . , data = d.ozone, nterms=4)
                                # nterms specifies the number of ridge functions
sfsmisc::mult.fig(4) # or just  par(mfrow=c(2,2))
plot(fit) ## 4 terms -> 4 plots

predict(fit,xnew) # where 'xnew' is a new set of predictor variables
```

The estimated ridge functions are shown in Figure 7.6.

## 7.6 Classification and Regression Trees (CART)

Quite different from the previous methods are the so-called tree models. CART is the most well known tree model or algorithm in the statistics community.

The underlying model function for CART is

$$g_{tree}(x) = \sum_{r=1}^{R} \beta_r \mathbf{1}_{[x \in \mathcal{R}_r]},$$

where $\mathcal{P} = \{\mathcal{R}_1, \ldots, \mathcal{R}_R\}$ is a partition of $\mathbb{R}^p$. Thus, the function $g(\cdot)$ is modelled as *piecewise constant*.

### 7.6.1 Tree structured estimation and tree representation

Parameter estimation $\hat{\beta}_1, \ldots, \hat{\beta}_R$ is easy when the partition $\mathcal{P} = \{\mathcal{R}_1, \ldots, \mathcal{R}_R\}$ would be given. We use

$$\hat{\beta}_r = \sum_{i=1}^{n} Y_i \mathbf{1}_{[x_i \in \mathcal{R}_r]} / \sum_{i=1}^{n} \mathbf{1}_{[x_i \in \mathcal{R}_r]}$$

for regression and binary classification (we do not model $g(\cdot)$ on the logistic scale). (For $J$ class problems with $J > 2$, we can also use the tree model directly, using $\arg\max_j \#\{Y_i = j\}$, and do not need to go with a one against the rest approach as section 6.4.2).

The tricky issue is to get a data-driven estimate for the partition $\mathcal{P}$. Some restrictions will be made to obtain a computationally feasible algorithm. This will be discussed next.

### 7.6.2 Tree-structured search algorithm and tree interpretation

We restrict the search for a good partition to partition cells $\mathcal{R}$ which are **axes parallel rectangles**. Moreover, we proceed in a **greedy** way since the space of partitions consisting of axes parallel rectangles is still huge.

A tree-structured greedy algorithm then proceeds as follows.

1. Start with $R = 1$ subset, $\mathcal{P} = \{\mathcal{R}\} = \mathbb{R}^p$.
2. Refine $\mathcal{R}$ into $\mathcal{R}_{left} \cup \mathcal{R}_{right}$ where:

$$\begin{aligned}\mathcal{R}_{left} &= \mathbb{R} \times \mathbb{R} \times \ldots \times (-\infty, d] \times \mathbb{R} \ldots \times \mathbb{R},\\ \mathcal{R}_{right} &= \mathbb{R} \times \mathbb{R} \times \ldots \times (d, \infty) \times \mathbb{R} \ldots \times \mathbb{R},\end{aligned}$$

   where one of the axes is split at the split point $d$, where $d$ is from the finite set of mid-points between observed values. The search for the axes to split and the split point $d$ are determined such that the negative log-likelihood is maximally reduced with the refinement (search over $j \in \{1, \ldots, p\}$ and $d \in \{$mid-points of observed values$\}$). Build the new partition $\mathcal{P} = \{\mathcal{R}_1, \mathcal{R}_2\}$ with $\mathcal{R}_1 = \mathcal{R}_{left},\ \mathcal{R}_2 = \mathcal{R}_{right}$.

3. Refine the current partition $\mathcal{P}$ as in step 2 by refining **one** of the partition cells from the current partition $\mathcal{P}$. That is, we search for the best partition cell to refine which includes a search as in step 2 for the best axes to split and the best split point. Then, we up-date the partition:

$$\mathcal{P} = \mathcal{P}_{old} \setminus \text{partition cell selected to be refined} \cup \{\text{refinement cells } \mathcal{R}_{left}, \mathcal{R}_{right}\}.$$

4. Iterate step 3 for a large number of partition cells.
5. Backward deletion: prune the tree (see below) until a reasonable model size, typically determined via cross-validation, is achieved.

#### Tree representation

The search algorithm above has a useful tree representation. Figures 7.7 and 7.8 show for part of the kyphosis dataset (2 class problem but now only with two predictor variables instead of 3) how $\mathbb{R}^2$ is recursively partitioned in a tree-structured way. As a side result, we obtain a very useful interpretation of the model in terms of a tree! Such a decision tree corresponds to a **recursive** partitioning scheme as shown in Figure 7.8.

The process of backward deletion in step 5 can now be more easily understood. Steps 1–4 result in a large tree $\mathcal{T}_M$ ($M = R - 1$). Backward elimination, or *tree pruning*, then deletes successively the terminal node in the tree with the smallest increase of negative log-likelihood (or another fit measure $R(.)$). This will produce a sequence of trees

$$\mathcal{T}_M \supset \mathcal{T}_{M-1} \supset \ldots \supset \mathcal{T}_1 = \mathcal{T}_\emptyset := \{\mathcal{R}_0\}, \quad \mathcal{R}_0 = \text{root tree} = \mathbb{R}^p, \qquad (7.4)$$

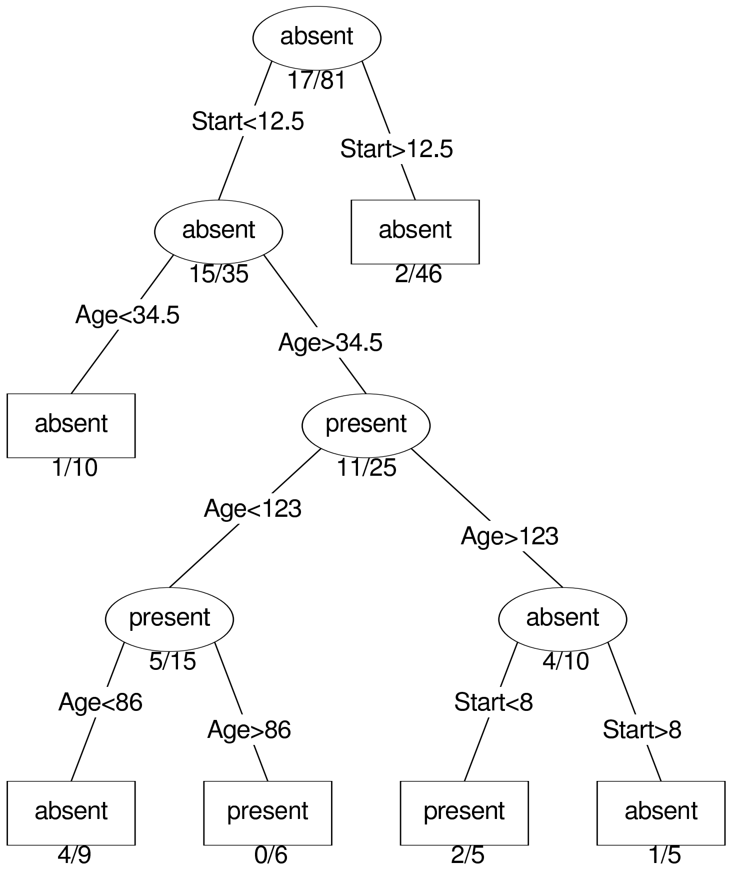

Figure 7.7: Classification tree for part of the kyphosis data set. "$a$" and "$b$" denote the frequencies of "presence" and "absence" of kyphosis in each node, and the probability estimate for kyphosis in a node is the $a/(a+b)$.

and we can select the "best" tree as follows:

As in regression, where we had a $C_p$ criterion, we apply "cost complexity (=: `cp`) pruning". The relevant measure is a penalized goodness of fit,

$$R_\alpha(\mathcal{T}) := R(\mathcal{T}) + \alpha \times \text{size}(\mathcal{T}), \qquad \alpha \geq 0 \tag{7.5}$$

where the *size* of a tree is the number of its leaves (or also 1+ the number of "cuts") and $R(.)$ is a quality of fit measure such as the deviance ($-2\cdot$ log.likelihood = sum of squares in the case of regression) or misclassification rate.

Now pruning is done such that for (conceptually) each $\alpha$ an optimally pruned tree is chosen, $\mathcal{T}(\alpha) := \arg\min_{\mathcal{T}} R_\alpha(\mathcal{T})$. It can be shown that the set $\{\mathcal{T}(\alpha) \mid \alpha \in [0,\infty)\}$ is *nested*, and is the same as (or subset of) the pruned trees in (7.4). For model selection, i.e., determining the amount of pruning, we now only need to choose the best $\alpha$ (or its normalization, `cp` = $\alpha/R(\mathcal{T}_\emptyset)$ in `rpart()`). For this, $k$-fold crossvalidation ($k = 10$ by default) is applied to compute CV error rates (in the case of classification) for each $\alpha$. Instead of directly using the minimal error rate to choose $\alpha$, the "1 S.E. rule" is used (see fig. 7.9): One takes the smallest tree (most sparse model) such that its error is at most one standard error larger than minimal one.

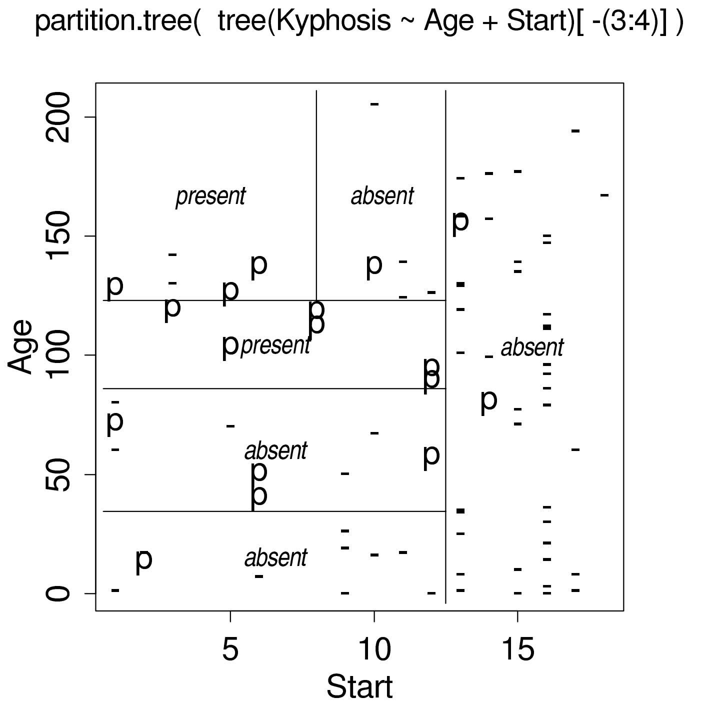

Figure 7.8: Partition of $\mathbb{R}^2$ for part of the kyphosis data set. "p" denotes presence of kyphosis, "-" denotes absence. Majority voting in each rectangle then determines the classification rule.

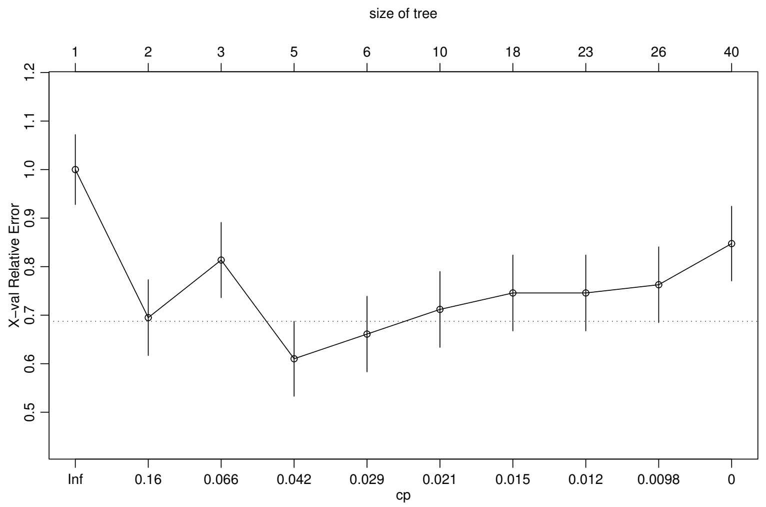

Figure 7.9: `plotcp(.)` for the Kyphosis example shows the full leave-1-out crossvalidated error rates of `rpart()`-models as function of $\alpha$ or `cp` = $\alpha/R(\mathcal{T}_\emptyset)$, respectively. A tree with five leaves (final nodes) seems therefore optimal.

### 7.6.3 Pros and cons of trees

One of the very nice properties of trees is their interpretation. Displaying information in terms of a tree structure is extremely useful and very popular in very many scientific fields. From the view of trees it also becomes clear that trees with depth $\ell$ may include interaction terms of degree $\ell$ between different predictor variables. Regarding performance, classification trees often yield quite good prediction results.
Also note that there is a nice approach for dealing with *missing values* via so called "surrogate splits" which allow to predict also when some predictor variable values are not known. Finally, it may be worthwhile to point out that tree models/algorithms are doing variable selection automatically.

The disadvantages of trees include the following. The regression function estimate, or probability estimate in classification, is piecewise constant: this is not usually the form one thinks of an underlying "true" function. This also implies that the prediction accuracy for regression curve estimation (or probability estimation in classification) is often not among the best. Finally, the greedy tree-type algorithm produces fairly unstable splits: in particular, if one of the first splits is "wrong", everything below this split (in terms of the tree) will be "wrong". Thus, interpretation of a tree – despite its attractiveness – should be done with caution.

In recent years, trees have also been used as building blocks of much more flexible (and computationally expensive) fitting methodologies, such as boosting, see chapter 8, or in "random forests". There[2], many trees are fit to the data and their variable selection and predictions combined into a final model.

### 7.6.4 CART in R

The function `rpart` from the package `rpart` in R can be used for fitting classification or regression trees.
As an example, we run a regression tree for the ozone data set and display the tree in Figure 7.10.

```
> fit <- rpart(log(O3) ~ . , data= d.ozone, method = "anova")
                                  # method = ‘‘anova’’ specifies a regression tree
> fit
n= 330

node), split, n, deviance, yval
      * denotes terminal node

 1) root 330 184.251800 2.2129670
   2) temp< 65.5 198  67.348040 1.7804120
     4) ibht>=3669 100  25.313780 1.4871350
       8) temp< 57.5 78  16.626380 1.3713510
        16) humidity>=70 16   4.480741 0.9331287 *
        17) humidity< 70 62   8.280081 1.4844410 *
       9) temp>=57.5 22   3.934431 1.8976390 *
     5) ibht< 3669 98  24.656390 2.0796750
      10) day>=332 13   1.852759 1.3895210 *
      11) day< 332 85  15.664550 2.1852280
        22) day< 47.5 24   3.217857 1.7721080 *
        23) day>=47.5 61   6.739110 2.3477670 *
   3) temp>=65.5 132  24.287510 2.8617990
```

[2]see R packages `randomForest`, `randomSurvivalForest`, `varSelRF` etc

```
     6) vsty>=145 18   3.854144 2.2880300 *
     7) vsty< 145 114  13.571930 2.9523940
      14) ibtp< 226.5 57   4.525832 2.7703180 *
      15) ibtp>=226.5 57   5.266831 3.1344690 *
```

```
> library(maptree) ; draw.tree(fit)      ## is slightly better than simply
## plot(fit) ; text(fit, use.n = TRUE) ## for plotting the tree
```

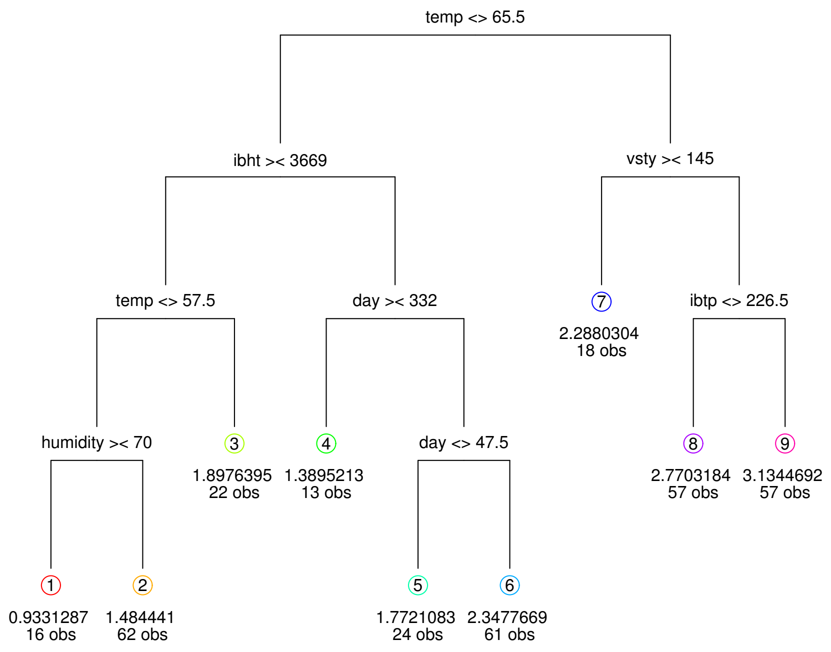

Figure 7.10: Regression tree for the ozone data set.

## 7.7 Variable Selection, Regularization, Ridging and the Lasso

### 7.7.1 Introduction

The following considerations and methods apply to both classification and regression in principle. For several reasons however we will focus on *regression* in this whole section.

In problems with many potential regressor or explanatory variables which may partly be highly correlated with each other, the classical least-squares regression approach may suffer from the fact that estimated regression coefficients can become quite ill-determined, i.e. have a high variance even when the fitted regression surface may be well determined. Also, in chapter 1's model equation (1.2) $Y_i = \mathbf{x}_i{}^{\intercal}\boldsymbol{\beta} + \varepsilon_i$, $(i = 1, \dots, n)$, equivalently

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{7.6}$$

where $\boldsymbol{\beta}, \mathbf{x}_i \in \mathbb{R}^p$, and $\mathbf{X}$ is the $n \times p$ matrix of rows $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ (or columns $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(p)}$), we had assumed that the number of observations $n$ was larger than the number of parameters $p$, i.e., $n > p$, in order e.g, for the matrix $X^{\intercal}X$ to be of full rank $p$.

One main motivation of this topic is the advent of data where there are many more potential explanatory variables than observations, in short, where $\boxed{p \gg n}$. Today the most important class of such applications is the analysis of genomic data, where e.g., for so called microarrays, one gets gene expression levels for thousands of genes ($p =$ 1000–50'000) for typically a few dozen cells ($n =$ 20–100). The target variable $Y$ here is often binary (e.g., "normal" vs "special") which means a classification problem with $J = 2$ classes; we assume here however that treating it as a regression problem (e.g., using least squares or $L_2$ error) is useful.

### 7.7.2 Ridge Regression

Let's consider the regression model (7.6), or

$$Y_i = \beta_0 + \beta_1 x_{i,1} + \ldots + \beta_p x_{i,p} + \varepsilon_i, \quad i = 1, \ldots, n.$$

For least squares, one can see that $\hat{\beta}'_0 = \overline{Y}(= 1/n \sum_{i=1}^n Y_i)$ when the model is re-written as

$$Y_i = \beta'_0 + \beta_1(x_{i,1} - \overline{x_{\cdot,1}}) + \ldots + \beta_p(x_{i,p} - \overline{x_{\cdot,p}}) + \varepsilon_i.$$

Hence, in all of the following, we will assume that all the variables ($Y_\cdot$, and $x_{\cdot,j} =: \mathbf{x}^{(j)}$) have been centered, i.e., mean subtracted, such that no intercept is needed, and we use the equivalent model for the transformed variables,

$$Y_i = \beta_1 x_{i,1} + \ldots + \beta_p x_{i,p} + \varepsilon_i, \quad i = 1, \ldots, n. \tag{7.7}$$

Note that in order have $\beta_j$ on a comparable scale, one also typically *scales* the $\mathbf{x}^{(j)}$'s such that $\|\mathbf{x}^{(j)}\| = 1$ for all $j$ .

Let's now assume that the variables (column vectors) $\mathbf{x}^{(j)} = (x_{i,j})_{i=1}^n$ and $\mathbf{x}^{(k)}$ are highly correlated (positively or negatively), or equivalently, since they both have mean 0, $\mathbf{x}^{(k)} \approx c\mathbf{x}^{(j)}$ (for $c \neq 0$) or $\mathbf{x}^{(j)} \approx 1/c\,\mathbf{x}^{(k)}$. As a consequence, in the following part of the model,

$$\begin{aligned} \beta_j \mathbf{x}^{(j)} + \beta_k \mathbf{x}^{(k)} &\approx \beta_j \mathbf{x}^{(j)} + \beta_k\,(c\mathbf{x}^{(j)}) \\ &= (\beta_j + \beta_k c)\mathbf{x}^{(j)} \\ &\approx (\beta_k + \beta_j/c)\mathbf{x}^{(k)} \end{aligned}$$

the coefficients of $\mathbf{x}^{(j)}$ and $\mathbf{x}^{(k)}$ are not well determined individually, but their model term $(\beta_j \mathbf{x}^{(j)} + \beta_k \mathbf{x}^{(k)})$ still is. Geometrically, in the $\beta_j$-$\beta_k$-plane the high confidence region forms narrow ellipses, i.e., the $\beta$ components themselves are linearly related, or that the coefficients of $\hat{\beta}_j$ and $\hat{\beta}_k$ themselves highly correlated but not be well determined individually, i.e., have a large variance. In the extreme case of "perfect" correlation, the matrix $\mathbf{X}$ would have columns $j$ and $k$ collinear and hence only have rank $\leq p-1$. When the correlation is less extreme, $\mathbf{X}$ is still of full rank $p$ and $\mathbf{X}^\intercal\mathbf{X}$ is close to a singular matrix.[3] One approach to this problem is to *regularize* it by improving the condition of the matrix corresponding to $\mathbf{X}^\intercal\mathbf{X}$.

To give a numerical example, say $\mathbf{x}^{(2)} \approx -2\mathbf{x}^{(1)}$, then $1\mathbf{x}^{(1)}$ is close to $3\mathbf{x}^{(1)} + \mathbf{x}^{(2)}$ and hence to $5\mathbf{x}^{(1)} + 2\mathbf{x}^{(2)}$ or $51\mathbf{x}^{(1)} + 25\mathbf{x}^{(2)}$. One way to make the linear combination more clearly determined is to restrict the coefficients to small (absolute) values, or, more conveniently requiring that $\sum_j \beta_j^2$ be "small". This leads to the so called *ridge regression*

$$\tilde{\boldsymbol{\beta}}(s) = \underset{\|\boldsymbol{\beta}\|^2 \leq s}{\arg\min} \|\mathbf{Y} - X\boldsymbol{\beta}\|^2 ,$$

[3]such that $\mathrm{Cov}(\widehat{\boldsymbol{\beta}}) = \sigma^2(X^\intercal X)^{-1}$ (section 1.4.1) will have very large entries corresponding to the high variance (and correlation) of $\hat{\beta}_j$ and $\hat{\beta}_k$.

which can be shown (by way of a Lagrange multiplier) to be equivalent to

$$\widehat{\boldsymbol{\beta}}^*(\lambda) = \arg\min_{\boldsymbol{\beta}} \{\|\mathbf{Y} - X\boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta}\|^2\}, \tag{7.8}$$

where there is a one-to-one relationship between $\lambda$ and the bound $s$ above, and the limit $\lambda \to 0$ corresponds to $s = \max \|\boldsymbol{\beta}\|^2 \to \|\widehat{\boldsymbol{\beta}}^{LS}\|^2$, i.e., the ordinary least squares of chapter 1. As there, by setting derivatives $\partial/\partial\beta$ to zero, this minimization problem is equivalent to the "normal equations"

$$(X^\intercal X + \lambda I)\widehat{\boldsymbol{\beta}}^* = X^\intercal \mathbf{Y}, \tag{7.9}$$

where the $p \times p$ matrix $(X^\intercal X + \lambda I)$ will be non-singular (and "well-conditioned") as soon as $\lambda > 0$ is large enough, even when $n < p$ and $X^\intercal X$ is clearly singular.

The ridge penalty entails that $\hat{\beta}_j(\lambda) \to 0$ (*"shrinking"*) when $\lambda \to \infty$, and also, in general, $\hat{\beta}_j \to \hat{\beta}_{j'}$ ("shrinking together") for two different coefficients.

Hence it seems intuitive that $\widehat{\boldsymbol{\beta}}^*$ will have some bias ($\mathbb{E}[\widehat{\boldsymbol{\beta}}^*] \neq \boldsymbol{\beta}$), but that its variance(s) can be considerably smaller than $\widehat{\boldsymbol{\beta}}^{LS}$ such that mean squared errors are smaller. As for smoothing (in particular, spline smoothing, sect. 3.4) we have a regularization parameter $\lambda$ which determines the trade-off between bias and variance, and as there, we'd use something like cross validation to determine an approximately optimal value for $\lambda$.

In the literature (and the R function `lm.ridge()` from the package `MASS`) there are "cheaper" approaches like GCV (see ch. 4) for determining an approximately optimal $\lambda$. In practice, one often wants to look at the *ridge traces*, i.e., a plot of the coefficients $\hat{\beta}_j(\lambda)$ vs $\lambda$. As an example we consider the longley macro economical data, for once modelling $y =$ `GNP.deflator` as function of the other six variables. The ridge traces $\hat{\beta}_j(\lambda)$ are shown in Figure 7.11. We have used a "relevant" interval for $\lambda$ where the shrinking towards zero is visible (but still somewhat distant from the limit).

### 7.7.3 The Lasso

In some sense, the "Lasso" (Tibshirani, 1996) regression is just a simple variant of ridge regression. However with the goal of *variable selection* in mind, the lasso brings a major improvement:

The lasso can be defined by restricting the *absolute* instead of the squared values of the coefficients, i.e.,

$$\tilde{\boldsymbol{\beta}}(s) = \arg\min_{\sum_j |\beta_j| \le s} \|\mathbf{Y} - X\boldsymbol{\beta}\|^2 ,$$

or

$$\begin{aligned}\widehat{\boldsymbol{\beta}}^*(\lambda) &= \arg\min_{\boldsymbol{\beta}} \{\|\mathbf{Y} - X\boldsymbol{\beta}\|^2 + \lambda \sum_{j=1}^{p} |\beta_j|\}, \\ &= \arg\min_{\boldsymbol{\beta}} \{\|\mathbf{Y} - X\boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta}\|_1\}. \end{aligned} \tag{7.10}$$

As opposed to the ridge regression case above, this problem is not solvable by simple linear algebra but rather needs quadratic programming or related algorithms.

On the other hand, the solution is much more interesting, because it will be frequent that $\hat{\beta}_j$ will become *exactly 0* as soon as $\lambda > \lambda_j$, in other words, choosing $\lambda$ here, automatically means model selection, namely only choosing regressor variables $\mathbf{x}^{(j)}$ with $\beta_j \neq 0$.

This can be visualized considering the "lasso traces".

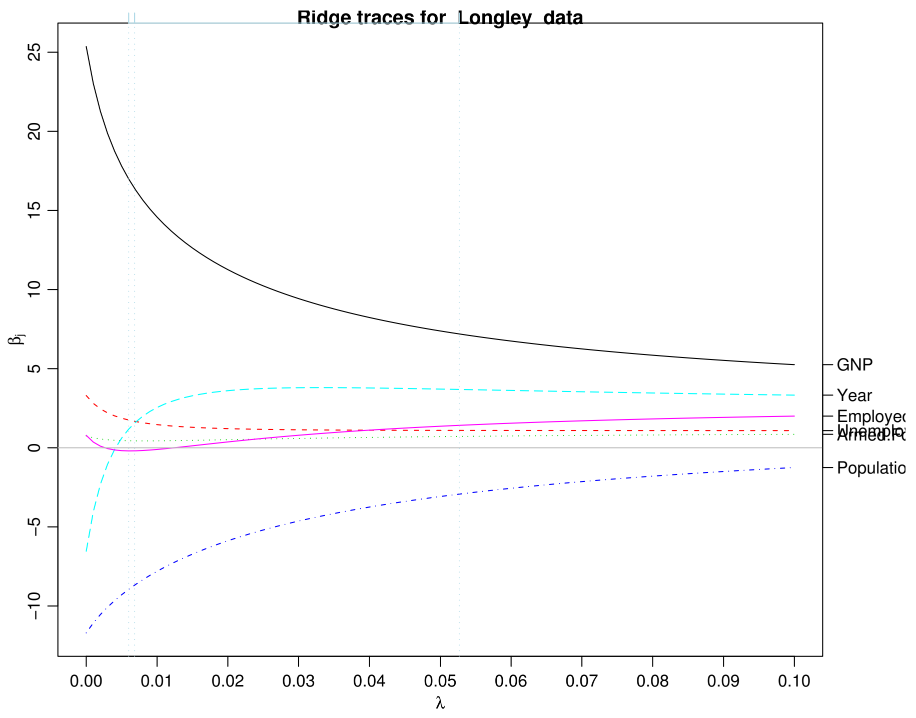

Figure 7.11: Ridge traces for the six coefficients $\beta_j(\lambda)$ $(j = 1, \ldots, 6)$ for the Longley data. The vertical lines indicate traditional estimates of the optimal ridge parameter $\lambda$.

### 7.7.4 Lasso extensions

#### 1) Regularization and variable selection via the elastic net

The elastic net has been propagated by Zou and Hastie (2003), *J.R. Statist. Soc. B*, and is implemented in R package `elasticnet`. For any fixed non-negative $\lambda_1$ and $\lambda_2$, we define the naïve elastic net criterion

$$L(\lambda_1, \lambda_2, \boldsymbol{\beta}) = \|\mathbf{Y} - X\boldsymbol{\beta}\|^2 + \lambda_2 \|\boldsymbol{\beta}\|^2 + \lambda_1 \|\boldsymbol{\beta}\|_1, \tag{7.11}$$

and the naïve elastic net estimator $\hat{\boldsymbol{\beta}}$ is the minimizer of (7.11), $\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} L(\lambda_1, \lambda_2, \boldsymbol{\beta})$.

This procedure can be viewed as a penalized least squares method. Let

$$\alpha = \lambda_2 / (\lambda_1 + \lambda_2),$$

then solving $\hat{\boldsymbol{\beta}}$ in (7.11) is equivalent to the optimization problem

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{Y} - X\boldsymbol{\beta}\|^2, \quad \text{subject to} \quad (1-\alpha)\|\boldsymbol{\beta}\|_1 + \alpha \|\boldsymbol{\beta}\|^2 \le t \quad \text{for some} \quad t. \tag{7.12}$$

We call the function $(1-\alpha)\|\boldsymbol{\beta}\|_1 + \alpha \|\boldsymbol{\beta}\|^2$ the *"elastic net penalty"*, which is a convex combination of the lasso and ridge penalty. When $\alpha = 1$, the naïve elastic net becomes simple ridge regression.

Here, we consider only $\alpha < 1$. For all $\alpha \in [0, 1)$, the elastic net penalty function is singular (without first derivative) at 0 and it is strictly convex for all $\alpha > 0$, thus having the characteristics of both the lasso and ridge regression. Note that the lasso penalty $(\alpha = 0)$ is convex but not strictly convex. These arguments can be seen clearly from Fig. 7.12.

## 2) The Adaptive Lasso and its Oracle Properties

Based on Hui Zou (2006), JASA. Let us consider the lasso with penalty weights,

$$\arg\min_{\boldsymbol{\beta}} \Big\| \mathbf{Y} - \sum_{j=1}^{p} \beta_j \mathbf{x_j} \Big\|^2 + \lambda \cdot \sum_{j=1}^{p} w_j \, |\beta_j| \,. \tag{7.13}$$

where $\mathbf{w}$ is a known weights vector. Zou shows that if the weights are data-dependent and cleverly chosen, the weighted lasso can possess so called oracle properties. The new methodology, called the *adaptive lasso*, is defined as follows. Suppose $\hat{\boldsymbol{\beta}}$ is a $\sqrt{n}$ consistent estimator of $\boldsymbol{\beta}$. For example, we can use the least squares $\widehat{\boldsymbol{\beta}_{ols}}$. Pick a $\gamma > 0$, and define the weight vector $\hat{\mathbf{w}}$ componentwise as $\widehat{w}_j = \frac{1}{\hat{\beta}_j^{\gamma}}$. The *adaptive lasso estimates* $\widehat{\boldsymbol{\beta}^*}_{(n)}$ are then given by

$$\widehat{\boldsymbol{\beta}^*}_{(n)} = \arg\min_{\boldsymbol{\beta}} \Big\| \mathbf{Y} - \sum_{j=1}^{p} \beta_j \mathbf{x_j} \Big\|^2 \;+\; \lambda \cdot \sum_{j=1}^{p} \widehat{w}_j \; |\beta_j| \,. \tag{7.14}$$

It is worth emphasizing that (7.14) is a convex optimization problem, and thus has no multiple local minimal and its global minimizer can be efficiently found.

## 3) The "Relaxed Lasso"

Recently developed by Nicolai Meinshausen, ETHZ and Berkeley, http://www.stat.berkeley.edu/~nicolai/ and implemented in R package `relaxo`.

Again motivated by the idea that variable selection on one hand and shrinking of the selected variables on the other hand should be controllable separately.

Let $\widehat{\boldsymbol{\beta}}^*(\lambda) = (\hat{\beta}_1^{\lambda}, \ldots, \hat{\beta}_p^{\lambda})^{\mathsf{T}}$ be the lasso estimated parameter vector of (7.10), and define

$$\mathcal{M}_{\lambda} = \{1 \le k \le p \mid \hat{\beta}_k^{\lambda} \ne 0\}, \tag{7.15}$$

the set of "significant" variables.

The *relaxed lasso estimator* is then defined for $\lambda \in [0, \infty)$ and $\phi \in [0, 1]$ as

$$\widehat{\boldsymbol{\beta}}^{\lambda,\phi} = \arg\min_{\boldsymbol{\beta}} \; n^{-1} \sum_{i=1}^{n} \Big( Y_i - \sum_{k \in \mathcal{M}_{\lambda}} \beta_k x_{i,k} \Big)^2 \;+\; \phi\lambda \cdot \|\boldsymbol{\beta}\|_1 \,. \tag{7.16}$$

For $\phi = 1$ this is the "classical" lasso, whereas for $\phi = 0$, this means to use lasso for variable selection and then re-estimate the selected non-zero parameters by least squares. Interesting is $0 < \phi < 1$ which can be shown to be much better than ordinary lasso (or OLS) in some situations. Astonishingly, there's a fast algorithm for computing solutions for all (relevant) $(\phi, \lambda)$ combinations which is only slightly more expensive than the fast lasso algorithm.

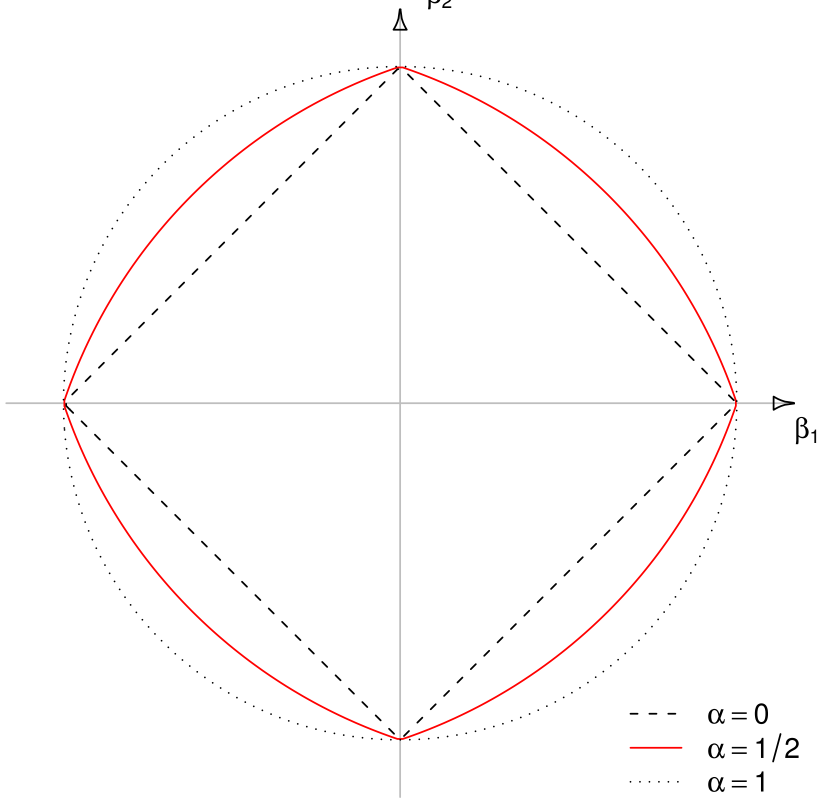

Figure 7.12: Two-dimensional contours of the ridge, lasso, and elastic net ($\alpha = \frac{1}{2}$) penalties.

# Chapter 8

# Bagging and Boosting

## 8.1 Introduction

**B**ootstrap **agg**regat**ing** (bagging) and boosting are useful techniques to improve the predictive performance of tree models. Boosting may also be useful in connection with many other models, e.g. for additive models with high-dimensional predictors; whereas bagging is most prominent for improving tree algorithms.

## 8.2 Bagging

Consider a base procedure, e.g. a tree algorithm such as CART, which yields a function estimate

$$\hat{g}(\cdot) : \mathbb{R}^p \to \mathbb{R}$$

(or $\hat{g}(\cdot)$ takes values in $[0, 1]$ for classification).

### 8.2.1 The bagging algorithm

Bagging works as follows.

1. Generate a bootstrap sample

$$(X_1^*, Y_1^*), \ldots, (X_n^*, Y_n^*)$$

and compute the bootstrapped estimator $\hat{g}^*(\cdot)$.

2. Repeat step 1 $B$ times, yielding

$$\hat{g}^{*1}(\cdot), \ldots, \hat{g}^{*B}(\cdot).$$

3. Aggregate the bootstrap estimates

$$\hat{g}_{Bag}(\cdot) = B^{-1} \sum_{i=1}^{B} \hat{g}^{*i}(\cdot).$$

The bagging algorithm is nothing else than an approximation

$$\hat{g}_{Bag}(\cdot) \approx \mathbb{E}^*[\hat{g}^*(\cdot)]$$

which can be made arbitrarily good by increasing $B$. The novel point is that we should use now $\mathbb{E}^*[\hat{g} * (\cdot)]$ as a new estimator.

A trivial identity hints at some properties of bagging: write (the theoretical version of bagging with $B = \infty$)

$$\begin{aligned} \hat{g}_{Bag}(\cdot) &= \hat{g}(\cdot) + (\mathbb{E}^*[\hat{g}^*(\cdot)] - \hat{g}(\cdot)) \\ &= \hat{g}(\cdot) + \text{ bootstrap bias estimate.} \end{aligned}$$

Instead of subtracting the bootstrap bias estimate, we are adding it! What we can hope for is a variance reduction at the price of a higher bias. This turns out to be true if $\hat{g}(\cdot)$ is a tree-based estimator.

### 8.2.2 Bagging for trees

It can be shown that for tree-based estimators $\hat{g}(\cdot)$,

$$\mathrm{Var}(\hat{g}_{Bag}(x)) \overset{\text{asymp.}}{<} \mathrm{Var}(\hat{g}(x)),$$

for very many $x$. Thus, bagging is a variance reduction technique. The reason for this is that a bagged tree turns out to be a product of probit functions $\Phi(d - \cdot)$ instead of indicator functions $\mathbf{1}_{[\cdot \le d]}$. This causes a variance reduction at the price of some bias. For example,

$$\mathrm{Var}(\mathbf{1}_{[X \le d]}) = \mathbb{P}[X \le d](1 - \mathbb{P}[X \le d]).$$

If $X \sim \mathcal{N}(0, 1)$ and $d = 0$, the latter quantity equals 1/4. On the other hand,

$$\mathrm{Var}(\Phi(-X)) = \mathrm{Var}(U) = 1/12, \quad U \ \sim \ \text{Unif.}([0, 1]),$$

which reduces the variance by the factor 3!

We should use large trees for bagging, because the variance reduction due to bagging asks for a large tree to balance the bias-variance trade-off.

### 8.2.3 Subagging

Subagging (**sub**sample **agg**regat**ing**) is a version of bagging: instead of drawing a bootstrap sample in step 1 of the bagging algorithm, we draw

$$(X_1^*, Y_1^*), \ldots, (X_m^*, Y_m^*) \text{ without replacement}$$

for some $m < n$. In some simple cases, it can be shown that $m = [n/2]$ is equivalent to bagging. Thus, subagging with $m = [n/2]$ can be viewed as a computationally cheaper version of bagging.

We consider a dataset about ozone concentration with $p = 8$ predictor variables (different from the previous ozone dataset). The performance of (su-)bagging for trees and MARS are shown in Figure 8.1. We see that bagging improves a regression tree substantially while it does not improve MARS at all (for this example).

The main drawback of bagging is the loss of interpretation in terms of a tree. It is by no means simple to interpret a linear combination of trees.

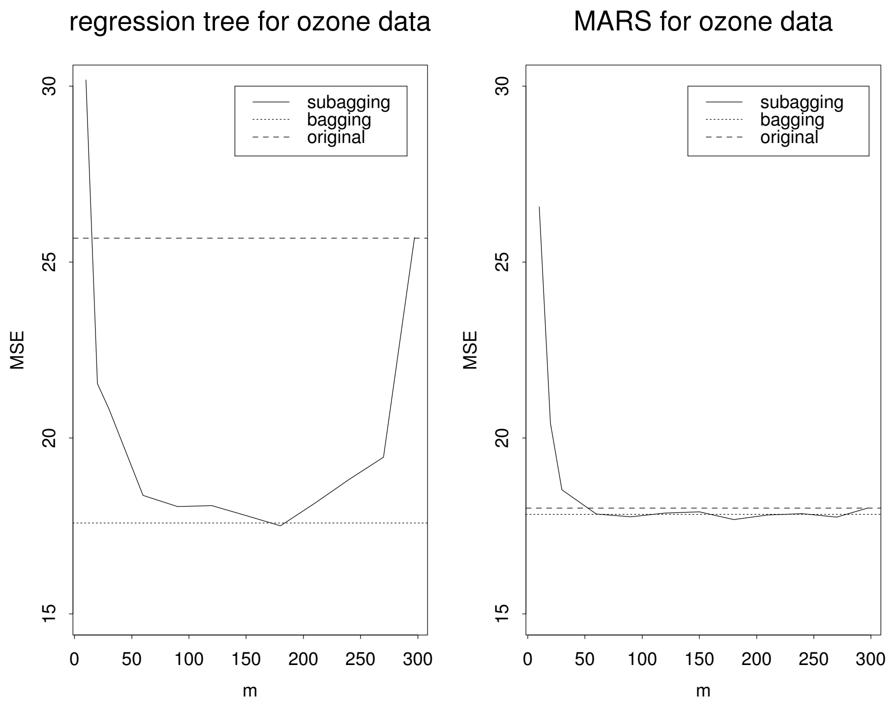

Figure 8.1: Mean squared error performance for a large regression tree and MARS and their (su-)bagged versions for an ozone data (different from the previous one).

## 8.3 Boosting

Boosting is a very different method to generate multiple predictions (function estimates) and combine them linearly. As with bagging, we have a base procedure yielding function estimates $\hat{g}(\cdot)$ (e.g. a tree algorithm).

### 8.3.1 $L_2$Boosting

The so-called $L_2$Boosting method (for regression) works as follows.

1. Fit a first function estimate from the data $\{(X_i, Y_i);\ i = 1, \ldots, n\}$ yielding a first function estimate $\hat{g}_1\cdot)$.
   Compute residuals
   $$U_i = Y_i - \hat{g}_1(X_i)\ (i = 1, \ldots, n).$$
   Denote by $\hat{f}_1(\cdot) = \nu\hat{g}_1(\cdot)\ (0 < \nu \leq 1)$.

2. For $m = 2, 3, \ldots, M$ do:
   Fit the residuals
   $$(X_i, U_i) \rightarrow \hat{g}_m(\cdot)$$
   and set
   $$\hat{f}_m(\cdot) = \hat{f}_{m-1}(\cdot) + \nu\hat{g}_m(\cdot).$$
   Compute the current residuals
   $$U_i = Y_i - \hat{f}_m(X_i)\ (i = 1, \ldots, n).$$

The shrinkage parameter $\nu$ can be chosen to be small, e.g. $\nu = 0.1$. The stopping parameter $M$ is a tuning parameter of boosting. For $\nu$ small we typically have to choose $M$ large.

Boosting is a bias reduction technique, in contrast to bagging. Boosting typically improves the performance of a single tree model. A reason for this is that we often cannot construct trees which are sufficiently large due to thinning out of observations in the terminal nodes. Boosting is then a device to come up with a more complex solution by taking linear combination of trees. In presence of high-dimensional predictors, boosting is also very useful as a regularization technique for additive or interaction modeling.