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

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

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 $X_{1j}, \dots X_{nj}$ is denoted by the hat operator

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

for any response vector $(U_1, \dots, U_n)^\intercal$. The subscript j indicates that smoothing is done against the jth 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 $j = 1, \dots, p$.

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

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

   where $\mathbf{Y} = (Y_1, \dots, Y_n)^\intercal$ and $\hat{\mathbf{g}}_j = (\hat{g}_j(X_{1j}), \dots, \hat{g}_j(X_{nj}))^\intercal$. 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)
d.ozone ← read.table("/u/sfs/Archives-Data/Hastie-Tibsh.-GAM/ozone.dat",header=TRUE)
pairs(d.ozone, pch = ".", gap = 0.1) #  --> Scatterplot Matrix
```

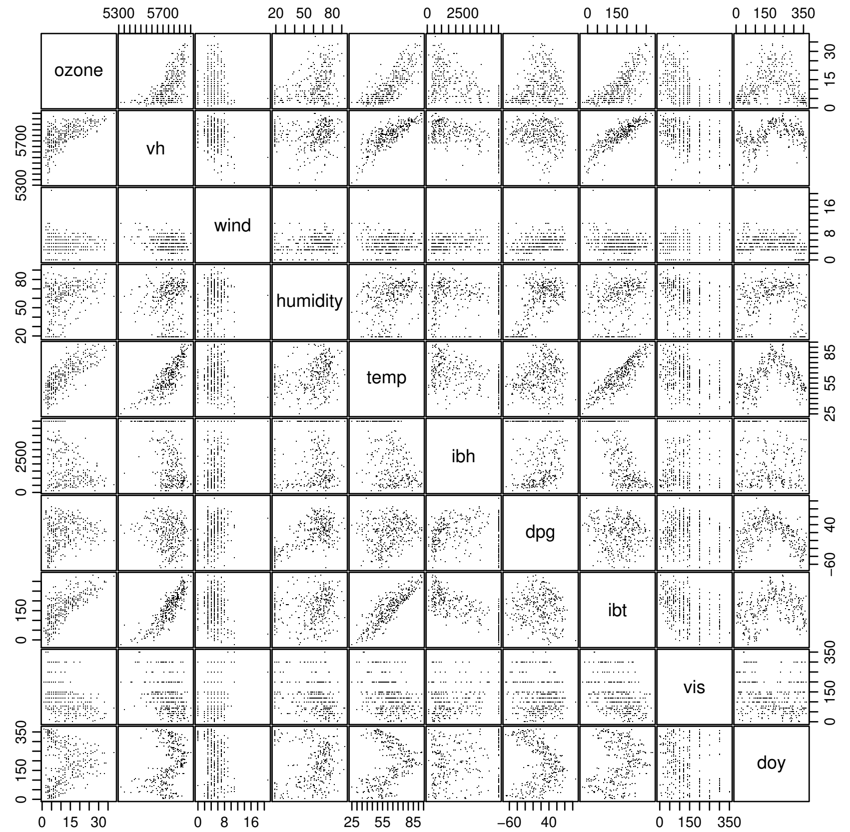

Figure 7.1: Daily ozone concentration in the Los Angeles basin as a function of 9 predictor variables.

```
fit ←
 gam(ozone ~ s(vh)+s(wind)+s(humidity)+s(temp)+s(ibh)+s(dpg)+s(ibt)+s(vis)+s(doy),
     data = d.ozone)
summary(fit)

Family: gaussian
Link function: identity
Formula:
ozone ~ s(vh) + s(wind) + s(humidity) + s(temp) + s(ibh) + s(dpg) + s(ibt) + s(vis) + s(doy)
```

```
Parametric coefficients:
              Estimate  std. err.    t ratio    Pr(>|t|)
(Intercept)     11.776     0.1988      59.22    < 2.22e-16

Approximate significance of smooth terms:
                   edf        chi.sq     p-value
       s(vh)         1        9.3598     0.0024152
     s(wind)         1        4.3752     0.037295
s(humidity)      3.631         21.37     0.013153
     s(temp)     4.361         42.25     7.5568e-06
      s(ibh)     3.043        15.753     0.077258
      s(dpg)      3.23        55.489     5.4967e-08
      s(ibt)     1.939         20.27     0.018874
      s(vis)     2.232         29.18     0.00088924
      s(doy)     4.021        91.569     1.0429e-13

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

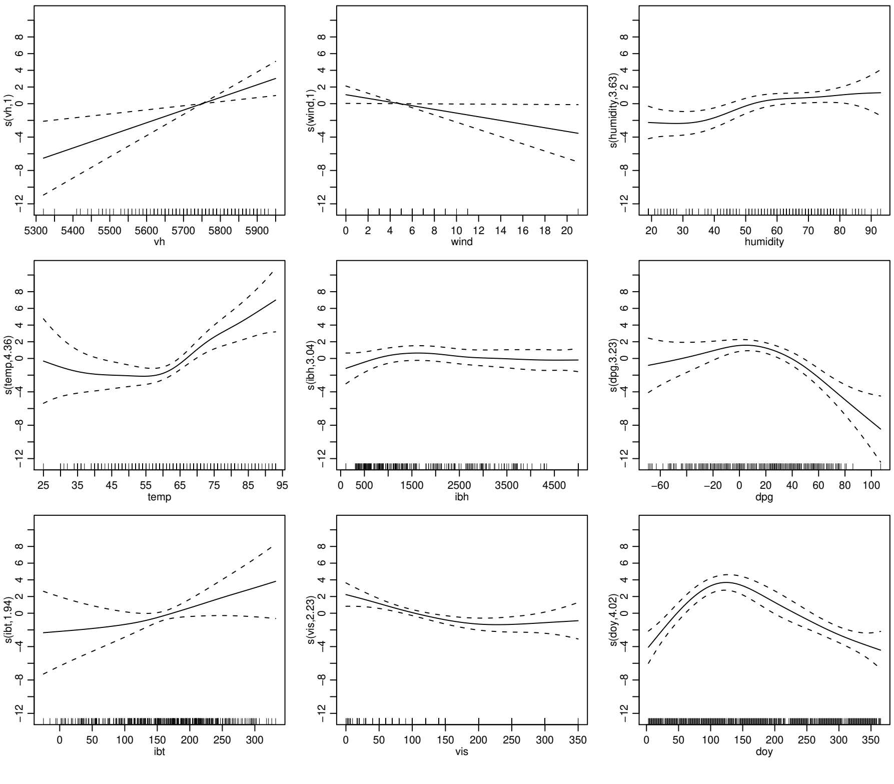

Figure 7.2: Estimated function $\hat{g}_j$ for the ozone data set.

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. The Tukey-Anscombe plot indicates heteroscedastic errors. We thus try the log-transform for the response variable and re-do the whole analysis. The results are given below.

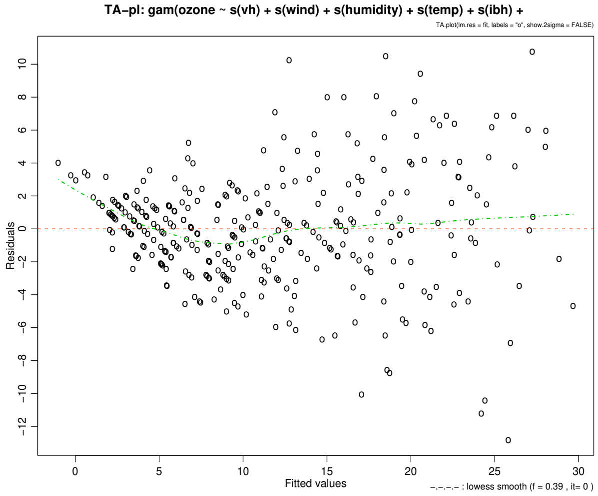

Figure 7.3: Tukey-Anscombe plot for the ozone data set.

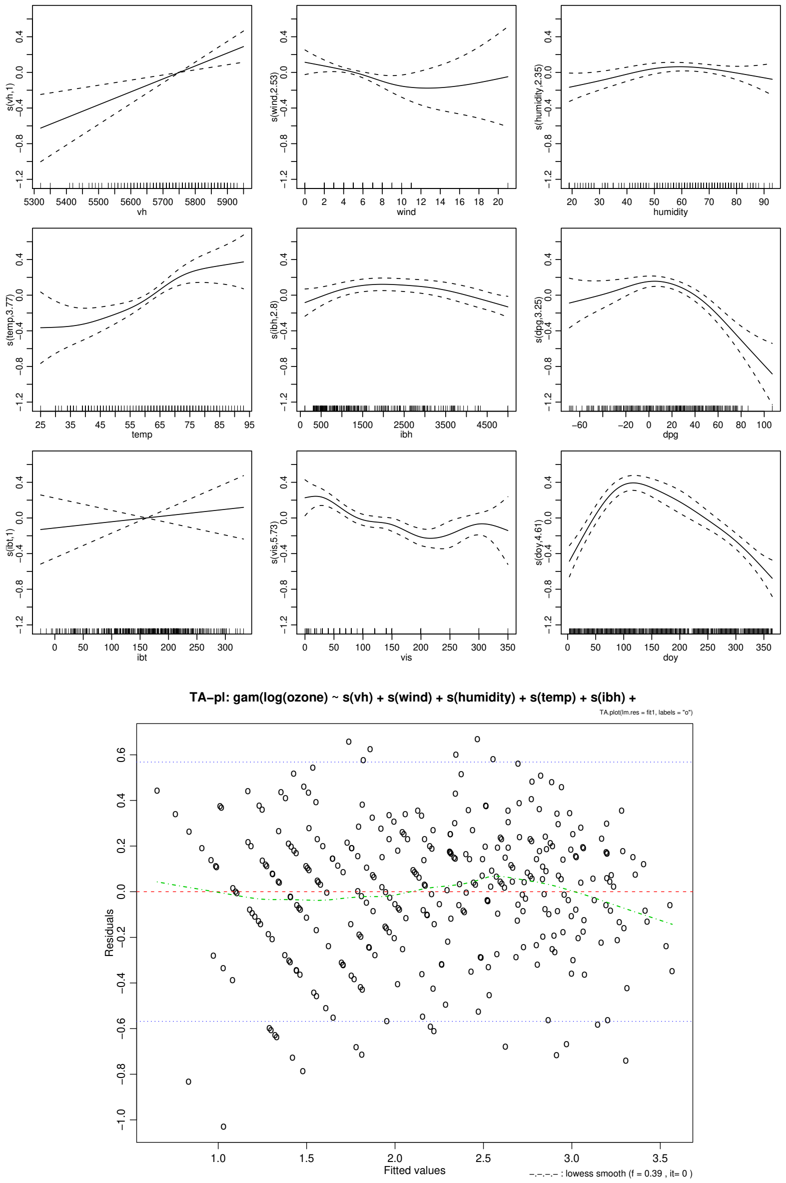

Figure 7.4: Estimated additive functions and Tukey-Anscombe plot for the ozone dataset with `log(ozone)` as the response variable.

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 $(j \in \{1, \ldots, p\})$ 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 jth component of x is relevant. The collection of reflected pairs is then a set of basis functions

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

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

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

   for some h_ℓ 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 of those pairs of functions $h_{2r-1}(\cdot), h_{2r}(\cdot)$ which increase 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:

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

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 is implemented in R in the function `mars` from the package `mda`. The syntax is as follows:

```
fit <- mars(x, y, degree=2)
#x is the n*p design matrix; y is the response vector

predict(fit,x=xnew)
#xnew is a new (test) set of predictors of dimension ntest*p
```

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

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

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

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

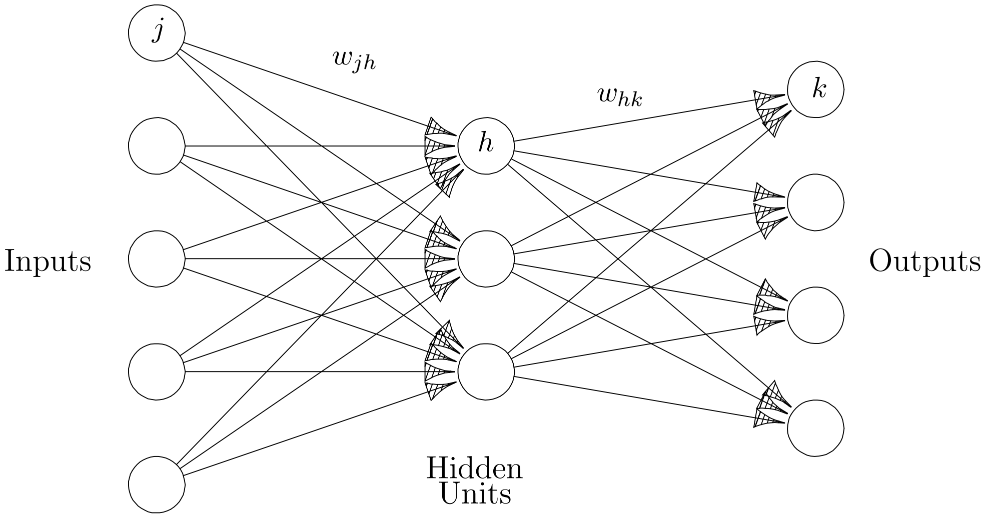

Figure 7.5: *Simple "feed-forward" neural net with one hidden layer.*

wheras $f_0(\cdot)$ is typically chosen as the identity for regression and as the sigmoid $\phi(x)$ for softmax classification. The $w_{hk}, w_{jk}, \tilde{\alpha}_h, \alpha_k$ all are unknown parameters. The so-called input layer consists of the p predictor variables; the values $w_{hk}\phi\left(\alpha_h + \sum_{j=1}^{p} w_{jh}x_j\right)$ $(h = 1, \ldots, q)$ 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`.

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 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, but use  log(ozone) instead of ozone
## and  scale (to mean = 0, sd = 1)
## 1. log(ozone)  2. scale
sc.ozone <- transform(d.ozone, log.O3 = log(ozone))[, -1]
sc.ozone <- scale(sc.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 668.015742
```

```
iter  10 value 178.027127
......
......
final  value 58.003685
converged
> sum(residuals(fit)^2) #-> 56.7
> summary(fit)
 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
 -5.03  -9.08   8.13   3.93  26.67   9.48  -2.16  -0.13 -10.43 -21.97
 b->h2 i1->h2 i2->h2 i3->h2 i4->h2 i5->h2 i6->h2 i7->h2 i8->h2 i9->h2
  0.50  -0.13   0.04   0.02   0.21  -0.06  -0.18   0.03   0.03  -0.80
 b->h3 i1->h3 i2->h3 i3->h3 i4->h3 i5->h3 i6->h3 i7->h3 i8->h3 i9->h3
 -0.91   0.40   5.21  -3.94  -7.23 -13.81   0.19 -18.24  19.47   5.40
  b->o  h1->o  h2->o  h3->o  i1->o  i2->o  i3->o  i4->o  i5->o  i6->o  i7->o
-12.94  -0.60  22.27  -0.66   0.65  -0.13   0.05  -0.29   0.00   0.62  -0.46
 i8->o  i9->o
 -0.19   3.58

## 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 73.796049
converged
> sum(residuals(fit1)^2) #-> 72.5
> 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
 -0.16   0.56   0.01   1.23   0.15  -0.35  -0.39   0.77  -0.52  -0.04
 b->h2 i1->h2 i2->h2 i3->h2 i4->h2 i5->h2 i6->h2 i7->h2 i8->h2 i9->h2
 16.45   3.10  -3.41  -6.99  15.44  -0.61   9.99   6.60   7.97 -12.37
 b->h3 i1->h3 i2->h3 i3->h3 i4->h3 i5->h3 i6->h3 i7->h3 i8->h3 i9->h3
 -6.12 -23.80  -4.00 -22.99  27.27   2.57   4.91 -16.76   1.36   1.78
  b->o  h1->o  h2->o  h3->o
 -2.22   2.73   0.79   0.73
```

`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 a neural net uses per default 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:

$$g_{\mathrm{PPR}}(x) = \mu + \sum_{k=1}^{q} f_k(\sum_{j=1}^{p} \alpha_{jk} x_j), \quad \text{where}$$

$$\sum\nolimits_{j=1}^{p} \alpha_{jk}^2 = 1, \quad \mathbb{E}[f_k(\sum_{j=1}^{p} \alpha_{jk} X_j)] = 0, \quad \text{for all } k.$$

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 $\boldsymbol{\alpha}_k = (\alpha_{1k}, \ldots, \alpha_{pk})^\intercal$, $\alpha_k{}^\intercal x$ is the projection of x onto (a ray through $vec0$ in direction) α_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

$$g_{\mathrm{PPR}}(x) = \mu + \sum_{k=1}^{q} \gamma_k f_k(\sum_{j=1}^{p} \alpha_{jk} x_j),$$

$$\sum_{j=1}^{p} \alpha_{jk}^2 = 1, \quad \mathrm{Var}(f_k(\sum_{j=1}^{p} \alpha_{jk} X_j)) = 1, \quad \text{for all } k.$$

Consider the ozone data set from above.

```
fit <- ppr(ozone ~ . , data = d.ozone, nterms=4)
                     # nterms specifies the number of ridge functions
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*.

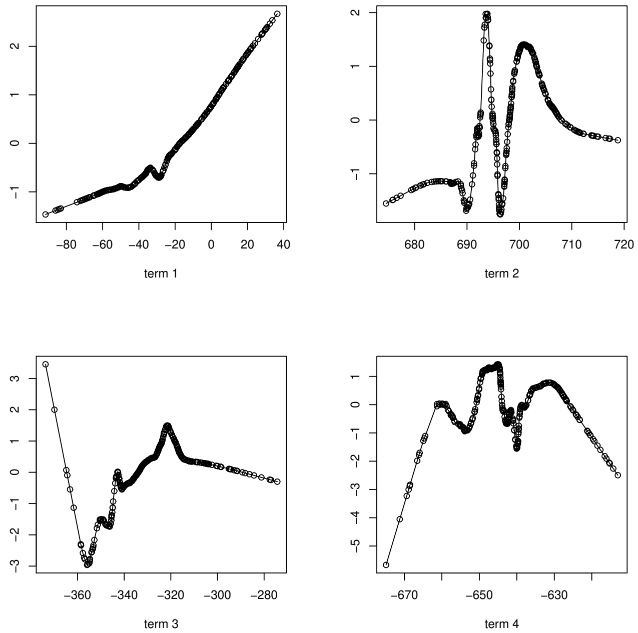

Figure 7.6: Four estimated ridge functions $f_k(\cdot)$ from a projection pursuit fit for the ozone data set.

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 an also use the tree model directly and do not need to go with a one against the rest approach, see 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 $j \in \{1, \dots, p\}$ 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

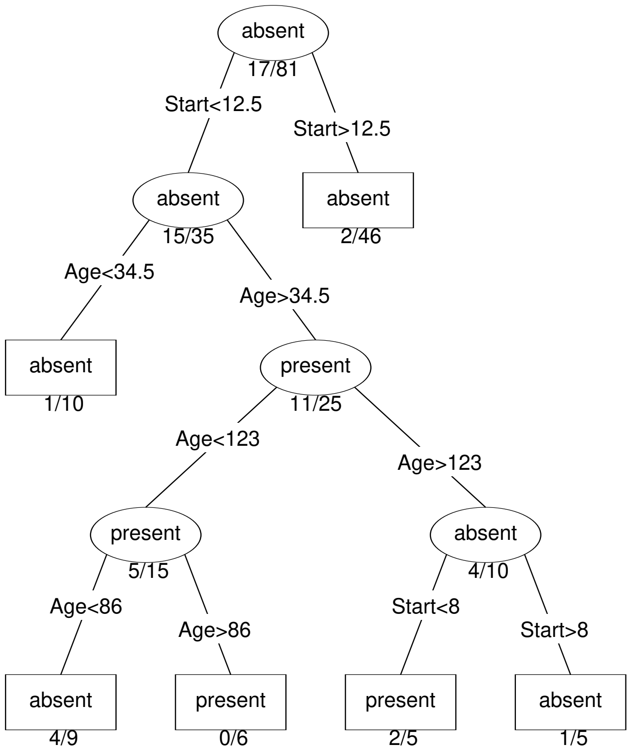

Figure 7.7: Classification tree for part of the kyphosis data set. "a" and "b" denote the frequencies of "presence" and "absence" of kyphosis in each node, and the probability estimate for kyphosis in a node is the $a/(a+b)$.

corresponds to a **recursive** partitioning scheme as shown in Figure 7.8.

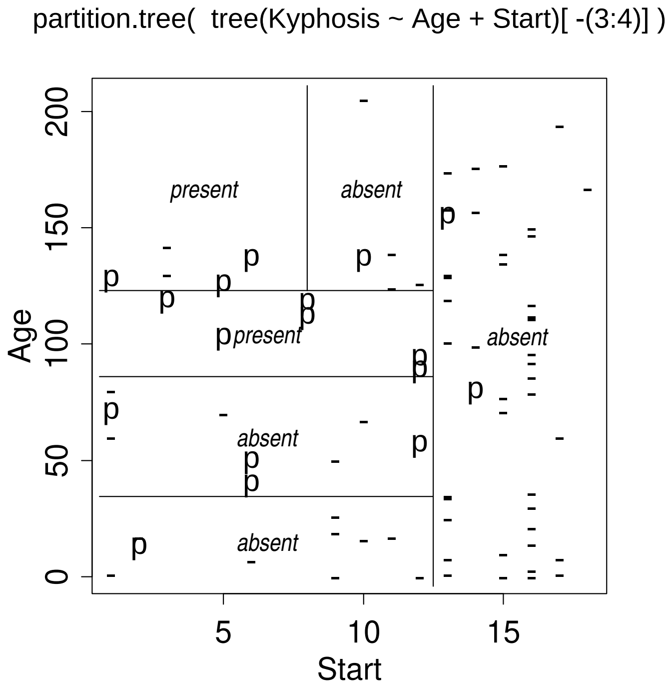

Figure 7.8: Partition of $\mathbb{R}^2$ for part of the kyphosis data set. "p" denotes presence of kyphosis, "-" denotes absence. Majority voting in each rectangle then determines the classification rule.

The process of backward deletion in step 5 can now be more easily understood. Steps 1–4 result in a large tree τ_M ($M = R - 1$). Backward elimination, or tree pruning, then deletes successively the terminal node in the tree which increases the negative log-likelihood the least. This will produce a sequence of trees

$$\tau_M \supset \tau_{M-1} \supset \ldots \supset \mathcal{R}_0 = \text{root tree} = \mathbb{R}^p,$$

and we can select the tree which has the best CV score.

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 ℓ may include interaction terms of degree ℓ between different predictor variables. Regarding performance, classification trees often yield quite good prediction results. 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 isn't 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.

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 an regression tree for the ozone data set.

```
d.ozone <-
  read.table("...../Hastie-Tibsh.-GAM/ozone.dat", header=TRUE)

fit <- rpart(ozone ~ . , 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 21115.4100 11.775760
   2) temp< 67.5 214  4114.3040  7.425234
     4) ibh>=3573.5 108   689.6296  5.148148 *
     5) ibh< 3573.5 106  2294.1230  9.745283
      10) dpg< -9.5 35   362.6857  6.457143 *
      11) dpg>=-9.5 71  1366.4790 11.366200
        22) ibt< 159 40   287.9000  9.050000 *
        23) ibt>=159 31   587.0968 14.354840 *
   3) temp>=67.5 116  5478.4400 19.801720
     6) ibt< 226.5 55  1276.8360 15.945450
      12) humidity< 59.5 10   167.6000 10.800000 *
      13) humidity>=59.5 45   785.6444 17.088890 *
     7) ibt>=226.5 61  2646.2620 23.278690
      14) doy>=280.5 8   398.0000 16.000000 *
      15) doy< 280.5 53  1760.4530 24.377360
        30) vis>=55 36  1149.8890 22.944440 *
        31) vis< 55 17   380.1176 27.411760 *

plot(fit)            #plotting the tree
text(fit, use.n=TRUE)
```

The fitted tree is displayed in Figure 7.9.

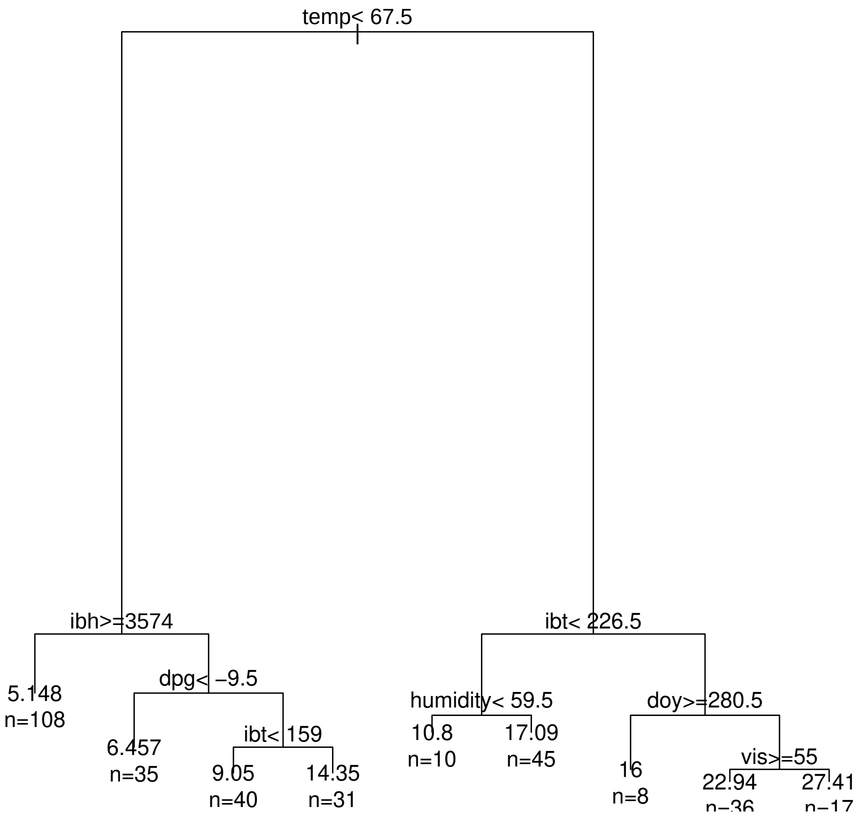

Figure 7.9: Regression tree for the ozone data set.