

Computational Statistics

Peter Bühlmann, Martin Mächler, and Nicolai Meinshausen

Seminar für Statistik
ETH Zürich

Version of May 19, 2023

Contents

1	Multiple Linear Regression	1
1.1	Introduction	1
1.2	The Linear Model	1
1.2.1	Stochastic Models	2
1.2.2	Examples	2
1.2.3	Goals of a linear regression analysis	3
1.3	Least Squares Method	4
1.3.1	The normal equations	4
1.3.2	Assumptions for the Linear Model	5
1.3.3	Geometrical Interpretation	6
1.3.4	Don't do many regressions on single variables!	7
1.3.5	Computer-Output from R	8
1.4	Properties of Least Squares Estimates	9
1.4.1	Moments of least squares estimates	10
1.4.2	Distribution of least squares estimates assuming Gaussian errors	10
1.5	Tests and Confidence Regions	10
1.5.1	Testing regression coefficients	10
1.5.2	Preview: computational alternative via bootstrap	11
1.5.3	Test of global null hypothesis	12
1.5.4	Permutation test as computational alternative	13
1.6	Analysis of residuals and checking of model assumptions	14
1.6.1	The Tukey-Anscombe Plot	14
1.6.2	The Normal Plot	15
1.6.3	Plot for detecting serial correlation	15
1.6.4	Generalized least squares and weighted regression	16
1.7	Model Selection	16
1.7.1	Penalized regression – Mallows C_p statistic	17
2	Nonparametric Density Estimation	19
2.1	Introduction	19
2.2	Estimation of a density	19
2.2.1	Histogram	19
2.2.2	Kernel estimator	20
2.3	The role of the bandwidth	21
2.3.1	Variable bandwidths: k nearest neighbors	22
2.3.2	The bias-variance trade-off	22
2.3.3	Asymptotic bias and variance	22
2.3.4	Estimating the bandwidth	24
2.4	Higher dimensions	24

2.4.1	The curse of dimensionality	25
3	Nonparametric Regression	27
3.1	Introduction	27
3.2	The kernel regression estimator	27
3.2.1	The role of the bandwidth	29
3.2.2	Inference for the underlying regression curve	29
3.3	Local polynomial nonparametric regression estimator	31
3.4	Smoothing splines and penalized regression	32
3.4.1	Penalized sum of squares	32
3.4.2	The smoothing spline solution	32
3.4.3	Shrinking towards zero	33
3.4.4	Relation to equivalent kernels	33
4	Cross-Validation	35
4.1	Introduction	35
4.2	Training and Test Set	35
4.3	Constructing training-, test-data and cross-validation	36
4.3.1	“Validation Set” or “Sample Splitting Once” Approach	36
4.3.2	Leave-one-out cross-validation	36
4.3.3	K -fold Cross-Validation	37
4.3.4	Many random divisions into test- and training-data	37
4.4	Properties of different CV-schemes	38
4.4.1	One random split into test- and training-data	38
4.4.2	Leave-one-out CV	38
4.4.3	Leave- d -out CV	39
4.4.4	K -fold CV; stochastic approximations ($B \gg 1$ splits)	39
4.5	Computational shortcut for some linear fitting operators	39
5	Bootstrap	41
5.1	Introduction	41
5.2	Efron’s nonparametric bootstrap	41
5.2.1	The bootstrap algorithm	42
5.2.2	The bootstrap distribution	43
5.2.3	Bootstrap consistence	43
5.2.4	Bootstrap confidence interval: a first approach	44
5.2.5	Bootstrap estimate of the generalization error	45
5.2.6	Out-of-bootstrap sample for estimation of the generalization error	46
5.3	Double bootstrap	47
5.4	Model-based bootstrap	50
5.4.1	Parametric bootstrap	50
5.4.2	Parametric bootstrap for model structures beyond i.i.d.	52
5.4.3	The model-based bootstrap for regression	53
6	Classification	55
6.1	Introduction	55
6.2	The Bayes classifier	55
6.3	The view of discriminant analysis	56
6.3.1	Linear discriminant analysis	56
6.3.2	Quadratic discriminant analysis	57
6.4	The view of logistic regression	57

6.4.1	Binary classification	58
6.4.2	Multiclass case, $J > 2$	60
6.5	Receiver operating characteristic (ROC) curve	61
7	Flexible regression and classification methods	63
7.1	Introduction	63
7.2	Additive models	63
7.2.1	Backfitting for additive regression models	64
7.2.2	Additive model fitting in R	64
7.3	MARS	69
7.3.1	Hierarchical interactions and constraints	70
7.3.2	MARS in R	70
7.4	Neural Networks	71
7.4.1	Fitting neural networks in R	72
7.4.2	Deeper networks	74
7.5	Projection pursuit regression	74
7.5.1	Projection pursuit regression in R	75
7.6	Classification and Regression Trees (CART)	75
7.6.1	Tree structured estimation and tree representation	76
7.6.2	Tree-structured search algorithm and tree interpretation	76
7.6.3	CART in R	79
7.6.4	Pros and cons of trees	80
7.6.5	“RandomForest” for powerful stable prediction	81
7.7	Variable Selection, Regularization, Ridge and the Lasso	82
7.7.1	Introduction	82
7.7.2	Ridge Regression	83
7.7.3	The Lasso	84
7.7.4	Lasso extensions	85
8	Bagging and Boosting	89
8.1	Introduction	89
8.2	Bagging	89
8.2.1	The bagging algorithm	89
8.2.2	Bagging for trees	90
8.2.3	Subbagging	91
8.3	Boosting	91
8.3.1	L_2 Boosting	91

Chapter 1

Multiple Linear Regression

1.1 Introduction

Linear regression is a widely used statistical model in a broad variety of applications. It is one of the easiest examples to demonstrate important aspects of statistical modelling.

1.2 The Linear Model

Multiple Regression Model:

Given is one **response variable**: up to some random errors it is a linear function of several **predictors** (or **covariables**).

The linear function involves unknown parameters. The goal is to estimate these parameters, to study their relevance and to estimate the error variance.

Model formula:

$$Y_i = \beta_1 x_{i1} + \dots + \beta_p x_{ip} + \varepsilon_i \quad (i = 1, \dots, n) \quad (1.1)$$

Usually we assume that $\varepsilon_1, \dots, \varepsilon_n$ are i.i.d. (independent, identically distributed) with $\mathbb{E}[\varepsilon_i] = 0$, $\text{Var}(\varepsilon_i) = \sigma^2$.

Notations:

- $\mathbf{Y} = \{Y_i; i = 1, \dots, n\}$ is the vector of the **response variables**
- $\mathbf{x}^{(j)} = \{x_{ij}; i = 1, \dots, n\}$ is the vector of the j th predictor (covariable) ($j = 1, \dots, p$)
- $\mathbf{x}_i = \{x_{ij}; j = 1, \dots, p\}$ is the vector of predictors for the i th observation ($i = 1, \dots, n$)
- $\boldsymbol{\beta} = \{\beta_j; j = 1, \dots, p\}$ is the vector of the unknown parameters
- $\boldsymbol{\varepsilon} = \{\varepsilon_i; i = 1, \dots, n\}$ is the vector of the unknown random **errors**
- n is the sample size, p is the number of predictors

The parameters β_j and σ^2 are unknown and the errors ε_i are unobservable. On the other hand, the response variables Y_i and the predictors x_{ij} have been observed.

Model in vector notation:

$$Y_i = \mathbf{x}_i^\top \boldsymbol{\beta} + \varepsilon_i \quad (i = 1, \dots, n)$$

Model in matrix form:

$$\boxed{\begin{array}{ccccccc} \mathbf{Y} & = & X & \times & \boldsymbol{\beta} & + & \boldsymbol{\varepsilon} \\ n \times 1 & & n \times p & & p \times 1 & & n \times 1 \end{array}} \quad (1.2)$$

where X is a $(n \times p)$ -matrix with rows \mathbf{x}_i^\top and columns $\mathbf{x}^{(j)}$.

The first predictor variable is often a constant, i.e., $x_{i1} \equiv 1$ for all i . We then get an intercept in the model.

$$Y_i = \beta_1 + \beta_2 x_{i2} + \dots + \beta_p x_{ip} + \varepsilon_i.$$

We typically assume that the sample size n is larger than the number of predictors p , $n > p$, and moreover that the matrix X has full rank p , i.e., the p column vectors $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(p)}$ are linearly independent.

1.2.1 Stochastic Models

The linear model in (1.1) involves some stochastic (random) components: the error terms ε_i are random variables and hence the response variables Y_i as well. The predictor variables x_{ij} are here assumed to be non-random. In some applications, however it is more appropriate to treat the predictor variables as random.

The stochastic nature of the error terms ε_i can be assigned to various sources: for example, measurement errors or inability to capture all underlying non-systematic effects which are then summarized by a random variable with expectation zero. The stochastic modelling approach will allow to quantify uncertainty, to assign significance to various components, e.g. significance of predictor variables in model (1.1), and to find a good compromise between the size of a model and the ability to describe the data (see section 1.7).

The observed response in the data is always assumed to be realizations of the random variables Y_1, \dots, Y_n ; the x_{ij} 's are non-random and equal to the observed predictors in the data.

1.2.2 Examples

Two-sample model:

$$p = 2, \quad X = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ \vdots & \vdots \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ \vdots & \vdots \\ 0 & 1 \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}.$$

Main questions: $\mu_1 = \mu_2$? Quantitative difference between μ_1 and μ_2 ?

From introductory courses we know that one could use the two-sample t -test or two-sample Wilcoxon test.

Regression through the origin: $Y_i = \beta x_i + \varepsilon_i$ ($i = 1, \dots, n$).

$$p = 1, \quad X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \quad \beta = \beta.$$

Simple linear regression: $Y_i = \beta_1 + \beta_2 x_i + \varepsilon_i$ ($i = 1, \dots, n$).

$$p = 2 \quad X = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}, \quad \beta = \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix}.$$

Quadratic regression: $Y_i = \beta_1 + \beta_2 x_i + \beta_3 x_i^2 + \varepsilon_i$ ($i = 1, \dots, n$).

$$p = 3, \quad X = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{pmatrix}, \quad \beta = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix}.$$

Note that the fitted function is quadratic in the x_i 's but *linear* in the coefficients β_j and therefore a special case of the linear model (1.1).

Regression with transformed predictor variables:

$Y_i = \beta_1 + \beta_2 \log(x_{i2}) + \beta_3 \sin(\pi x_{i3}) + \varepsilon_i$ ($i = 1, \dots, n$).

$$p = 3, \quad X = \begin{pmatrix} 1 & \log(x_{12}) & \sin(\pi x_{13}) \\ 1 & \log(x_{22}) & \sin(\pi x_{23}) \\ \vdots & \vdots & \vdots \\ 1 & \log(x_{n2}) & \sin(\pi x_{n3}) \end{pmatrix}, \quad \beta = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix}.$$

Again, the model is *linear* in the coefficients β_j but nonlinear in the x_{ij} 's.

In summary:

The model in (1.1) is called linear because it is linear in the coefficients β_j . The predictor (and also the response) variables can be transformed versions of the original predictor and/or response variables.

1.2.3 Goals of a linear regression analysis

- **A good “fit”.** Fitting or estimating a (hyper-)plane over the predictor variables to explain the response variables such that the errors are “small”. The standard tool for this is the method of *least squares* (see section 1.3).
- **Good parameter estimates.** This is useful to describe the change of the response when varying some predictor variable(s).
- **Good prediction.** This is useful to predict a new response as a function of new predictor variables.
- **Uncertainties and significance for the three goals above.** Confidence intervals and statistical tests are useful tools for this goal.

- **Development of a good model.** In an interactive process, using methods for the goals mentioned above, we may change parts of an initial model to come up with a better model.

The first and third goal can become conflicting, see section 1.7.

1.3 Least Squares Method

We assume the linear model $\mathbf{Y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon}$. We are looking for a “good” estimate of $\boldsymbol{\beta}$. The least squares estimator $\hat{\boldsymbol{\beta}}$ is defined as

$$\hat{\boldsymbol{\beta}} = \arg \min_{\boldsymbol{\beta}} \|\mathbf{Y} - X\boldsymbol{\beta}\|^2, \quad (1.3)$$

where $\|\cdot\|$ denotes the Euclidean norm in \mathbb{R}^n .

1.3.1 The normal equations

The minimizer in (1.3) can be computed explicitly (assuming that X has rank p). Computing partial derivatives of $\|\mathbf{Y} - X\boldsymbol{\beta}\|^2$ with respect to $\boldsymbol{\beta}$ (p -dimensional gradient vector), evaluated at $\hat{\boldsymbol{\beta}}$, and setting them to zero yields

$$(-2) X^\top(\mathbf{Y} - X\hat{\boldsymbol{\beta}}) = \mathbf{0} \quad ((p \times 1) - \text{null-vector}).$$

Thus, we get the **normal equations**

$$X^\top X \hat{\boldsymbol{\beta}} = X^\top \mathbf{Y}. \quad (1.4)$$

These are p linear equations for the p unknowns (components of $\hat{\boldsymbol{\beta}}$).

Assuming that the matrix X has full rank p , the $p \times p$ matrix $X^\top X$ is invertible, the least squares estimator is unique and can be represented as

$$\boxed{\hat{\boldsymbol{\beta}} = (X^\top X)^{-1} X^\top \mathbf{Y}.}$$

This formula is useful for theoretical purposes. For numerical computation it is much more stable to use the QR decomposition instead of inverting the matrix $X^\top X$.¹

From the *residuals* $r_i = Y_i - \mathbf{x}_i^\top \hat{\boldsymbol{\beta}}$, the usual estimate for σ^2 is

$$\hat{\sigma}^2 = \frac{1}{n-p} \sum_{i=1}^n r_i^2.$$

Note that the r_i 's are estimates for ε_i 's; hence the estimator is plausible, up to the somewhat unusual factor $1/(n-p)$. It will be shown in section 1.4.1 that due to this factor, $\mathbb{E}[\hat{\sigma}^2] = \sigma^2$.

¹Let $X = QR$ with orthogonal $(n \times p)$ matrix Q and upper (Right) triangular $(p \times p)$ R . Because of $X^\top X = R^\top Q^\top QR = R^\top R$, computing $\hat{\boldsymbol{\beta}}$ only needs subsequent solution of two triangular systems: First solve $R^\top \mathbf{c} = X^\top \mathbf{Y}$ for \mathbf{c} , and then solve $R\hat{\boldsymbol{\beta}} = \mathbf{c}$. Further, when $\text{Cov}(\hat{\boldsymbol{\beta}})$ requires $(X^\top X)^{-1}$, the latter is $R^{-1}(R^{-1})^\top$.

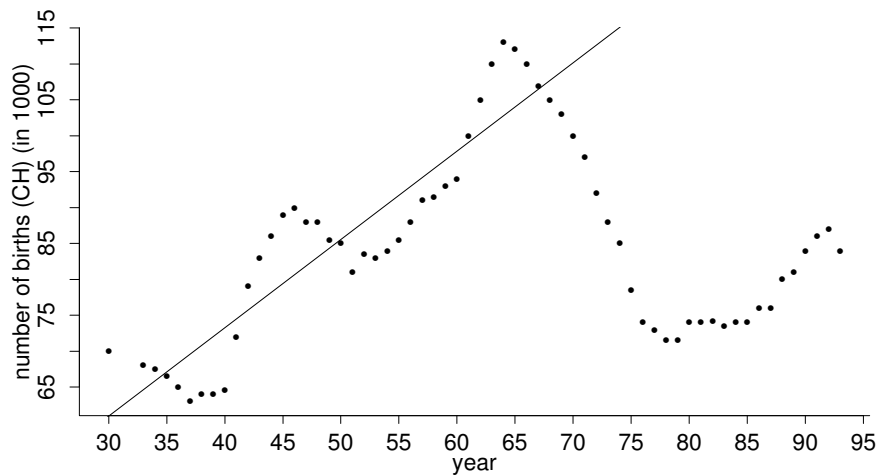


Figure 1.1: The pill kink.

1.3.2 Assumptions for the Linear Model

We emphasize here that we do not make any assumptions on the predictor variables, except that the matrix X has full rank $p < n$. In particular, the predictor variables can be continuous or discrete (e.g. binary).

We need some assumptions so that fitting a linear model by least squares is reasonable and that tests and confidence intervals (see 1.5) are approximately valid.

1. **The linear regression equation is correct.** This means: $\mathbb{E}[\varepsilon_i] = 0$ for all i .
2. **All x_i 's are exact.** This means that we can observe them perfectly.
3. **The variance of the errors is constant (“homoscedasticity”).** This means: $\text{Var}(\varepsilon_i) = \sigma^2$ for all i .
4. **The errors are uncorrelated.** This means: $\text{Cov}(\varepsilon_i, \varepsilon_j) = 0$ for all $i \neq j$.
5. **The errors $\{\varepsilon_i; i = 1, \dots, n\}$ are jointly normally distributed.** This implies that also $\{Y_i; i = 1, \dots, n\}$ are jointly normally distributed.

In case of violations of item 3, we can use weighted least squares instead of least squares. Similarly, if item 4 is violated, we can use generalized least squares. If the normality assumption in 5 does not hold, we can use robust methods instead of least squares. If assumption 2 fails to be true, we need corrections known from “errors in variables” methods. If the crucial assumption in 1 fails, we need other models than the linear model.

The following example shows violations of assumption 1 and 4. The response variable is the annual number of births in Switzerland since 1930, and the predictor variable is the time (year).

We see in Figure 1.1 that the data can be approximately described by a linear relation until the “pill kink” in 1964. We also see that the errors seem to be correlated: they are all positive or negative during periods of 10 – 20 years. Finally, the linear model is not representative after the pill kink in 1964. In general, it is dangerous to use a fitted model for extrapolation where no predictor variables have been observed (for example: if we had fitted the linear model in 1964 for prediction of number of births in the future until 2005).

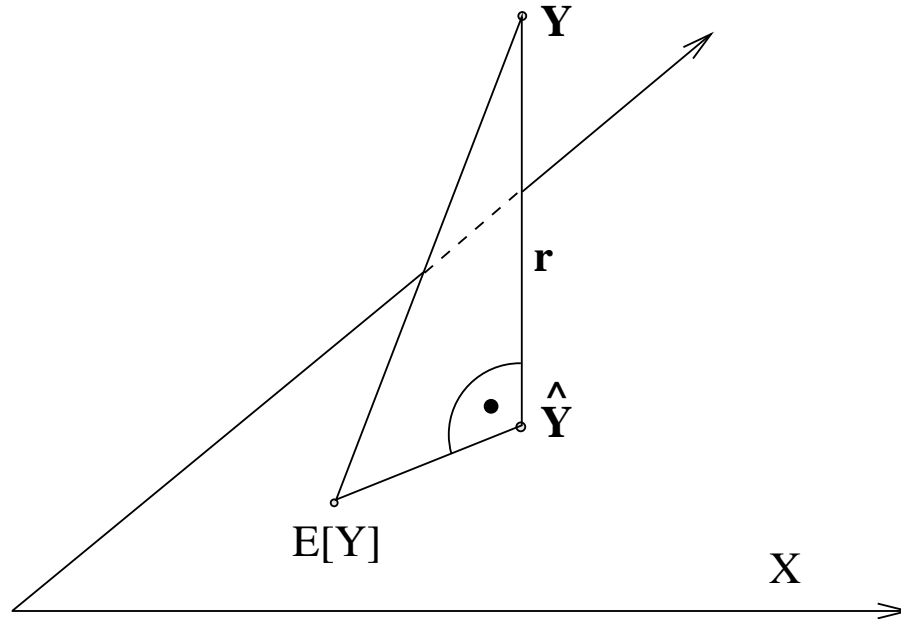


Figure 1.2: The residual vector \mathbf{r} is orthogonal to \mathcal{X} .

1.3.3 Geometrical Interpretation

The response variable \mathbf{Y} is a vector in \mathbb{R}^n . Also, $X\boldsymbol{\beta}$ describes a p -dimensional subspace \mathcal{X} in \mathbb{R}^n (through the origin) when varying $\boldsymbol{\beta} \in \mathbb{R}^p$ (assuming that X has full rank p). The least squares estimator $\hat{\boldsymbol{\beta}}$ is then such that $X\hat{\boldsymbol{\beta}}$ is closest to \mathbf{Y} with respect to the Euclidean distance. But this means geometrically that

$$X\hat{\boldsymbol{\beta}} \text{ is the orthogonal projection of } \mathbf{Y} \text{ onto } \mathcal{X}.$$

We denote the (vector of) fitted values by

$$\hat{\mathbf{Y}} = X\hat{\boldsymbol{\beta}}.$$

They can be viewed as an estimate of $X\boldsymbol{\beta}$.

The (vector of) residuals is defined by

$$\mathbf{r} = \mathbf{Y} - \hat{\mathbf{Y}}.$$

Geometrically, it is evident that the residuals are orthogonal to \mathcal{X} , because $\hat{\mathbf{Y}}$ is the orthogonal projection of \mathbf{Y} onto \mathcal{X} . This means that

$$\mathbf{r}^\top \mathbf{x}^{(j)} = 0 \text{ for all } j = 1, \dots, p,$$

where $\mathbf{x}^{(j)}$ is the j th column of X .

We can formally see why the map

$$\mathbf{Y} \mapsto \hat{\mathbf{Y}}$$

is an orthogonal projection. Since $\hat{\mathbf{Y}} = X\hat{\boldsymbol{\beta}} = X(X^\top X)^{-1}X^\top \mathbf{Y}$, the map can be represented by the matrix

$$P = X(X^\top X)^{-1}X^\top. \quad (1.5)$$

It is evident that P is symmetric ($P^\top = P$) and P is idem-potent ($P^2 = P$). Furthermore

$$\sum_i P_{ii} = \text{tr}(P) = \text{tr}(X(X^\top X)^{-1}X^\top) = \text{tr}((X^\top X)^{-1}X^\top X) = \text{tr}(I_{p \times p}) = p.$$

But these 3 properties characterize that P is an orthogonal projection from \mathbb{R}^n onto a p -dimensional subspace, here \mathcal{X} .

The residuals \mathbf{r} can be represented as

$$\mathbf{r} = (I - P)\mathbf{Y},$$

where $I - P$ is now also an orthogonal projection onto the orthogonal complement of \mathcal{X} , $\mathcal{X}^\perp = \mathbb{R}^n \setminus \mathcal{X}$, which is $(n - p)$ -dimensional. In particular, the residuals are elements of \mathcal{X}^\perp .

1.3.4 Don't do many regressions on single variables!

In general, it is not appropriate to replace multiple regression by many single regressions (on single predictor variables). The following (synthetic) example should help to demonstrate this point.

Consider two predictor variables $x^{(1)}, x^{(2)}$ and a response variable Y with the values

$x^{(1)}$	0	1	2	3	0	1	2	3
$x^{(2)}$	-1	0	1	2	1	2	3	4
Y	1	2	3	4	-1	0	1	2

Multiple regression yields the least squares solution which describes the data points exactly

$$Y_i = \hat{Y}_i = 2x_{i1} - x_{i2} \text{ for all } i \quad (\hat{\sigma}^2 = 0). \quad (1.6)$$

The coefficients 2 and -1, respectively, describe how y is changing when varying either $x^{(1)}$ or $x^{(2)}$ and keeping the other predictor variable constant. In particular, we see that Y decreases when $x^{(2)}$ increases.

On the other hand, if we do a simple regression of Y onto $x^{(2)}$ (while ignoring the values of $x^{(1)}$; and thus, we do not keep them constant), we obtain the least squares estimate

$$\hat{Y}_i = \frac{1}{9}x_{i2} + \frac{4}{3} \text{ for all } i \quad (\hat{\sigma}^2 = 1.72).$$

This least squares regression line describes how Y changes when varying $x^{(2)}$ while ignoring $x^{(1)}$. In particular, \hat{Y} increases when $x^{(2)}$ increases, in contrast to multiple regression!

The reason for this phenomenon is that $x^{(1)}$ and $x^{(2)}$ are strongly correlated: if $x^{(2)}$ increases, then also $x^{(1)}$ increases. Note that in the multiple regression solution, $x^{(1)}$ has a larger coefficient in absolute value than $x^{(2)}$ and hence, an increase in $x^{(1)}$ has a stronger influence for changing y than $x^{(2)}$. The correlation among the predictors in general makes also the interpretation of the regression coefficients more subtle: in the current setting, the coefficient β_1 quantifies the influence of $x^{(1)}$ on Y *after* having subtracted the effect of $x^{(2)}$ on Y , see also section 1.5.

Summarizing:

Simple least squares regressions on single predictor variables yield the multiple regression least squares solution, *only* if the predictor variables are orthogonal.

In general, *multiple* regression is the appropriate tool to include effects of more than one predictor variables simultaneously.

The equivalence in case of orthogonal predictors is easy to see algebraically. Orthogonality of predictors means $X^T X = \text{diag}(\sum_{i=1}^n x_{i1}^2, \dots, \sum_{i=1}^n x_{ip}^2)$ and hence the least squares estimator

$$\hat{\beta}_j = \sum_{i=1}^n x_{ij} Y_i / \sum_{i=1}^n x_{ij}^2 \quad (j = 1, \dots, p),$$

i.e., $\hat{\beta}_j$ depends only on the response variable Y_i and the j th predictor variable x_{ij} .

1.3.5 Computer-Output from R

We show here parts of the computer output (from R) when fitting a linear model to data about quality of asphalt².

```

y = LOGRUT : log("rate of rutting") = log(change of rut depth in inches
           per million wheel passes)
           ["rut" := `Wagenspur', ausgefahrenes Geleise]
x1 = LOGVISC : log(viscosity of asphalt)
x2 = ASPH    : percentage of asphalt in surface course
x3 = BASE    : percentage of asphalt in base course
x4 = RUN     : '0/1' indicator for two sets of runs.
x5 = FINES   : 10* percentage of fines in surface course
x6 = VOIDS   : percentage of voids in surface course

```

The following table shows the least squares estimates $\hat{\beta}_j$ ($j = 1, \dots, 7$), along with uncertainties in the estimates of the regression coefficients and -at the bottom- a test of the global null hypothesis (which is that all regression coefficients are in reality zero).³ $\sqrt{\hat{\sigma}^2}$ and the so-called *degrees of freedom* $n - p$.

```

> url <- "http://stat.ethz.ch/Teaching/Datasets/asphalt.dat"
> d.asphalt <- read.table(url, header = TRUE)
> asphalt1 <-
+   data.frame(LOGRUT = log(d.asphalt[, "RUT"]),
+             LOGVISC= log(d.asphalt[, "VISC"]),
+             d.asphalt[, 2:6]) # the rest
> asphMod <- lm(LOGRUT ~ . , data = asphalt1)

> summary(asphMod)

```

Calling `summary(.)` on the fitted model `asphMod` produces a table which displays the regression coefficient estimates along with their standard errors $\sqrt{\widehat{\text{Var}}(\hat{\beta}_j)} = \sqrt{\hat{\sigma}^2 (X^T X)^{-1}_{jj}}$, the t -test statistics for the null-hypotheses $H_{0,j} : \beta_j = 0$ and their corresponding two-sided P -values with some abbreviation about strength of significance. Moreover, the R^2 and adjusted R^2 are given and finally also the F -test statistic for the null-hypothesis $H_0 : \beta_2 = \dots = \beta_p = 0$ (with the degrees of freedom) and its corresponding P -value.

Call:

```
lm(formula = LOGRUT ~ . , data = asphalt1)
```

²the `untransformed` data is available as <http://stat.ethz.ch/Teaching/Datasets/asphalt.dat>

³The term "residual standard error" is a misnomer with a long tradition, since "standard error" usually means $\sqrt{\widehat{\text{Var}}(\hat{\theta})}$ for an estimated parameter θ .

Residuals:

	Min	1Q	Median	3Q	Max
	-0.48348	-0.14374	-0.01198	0.15523	0.39652

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	-5.781239	2.459179	-2.351	0.027280	*
LOGVISC	-0.513325	0.073056	-7.027	2.9e-07	***
ASPH	1.146898	0.265572	4.319	0.000235	***
BASE	0.232809	0.326528	0.713	0.482731	
RUN	-0.618893	0.294384	-2.102	0.046199	*
FINES	0.004343	0.007881	0.551	0.586700	
VOIDS	0.316648	0.110329	2.870	0.008433	**

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2604 on 24 degrees of freedom

Multiple R-squared: 0.9722, Adjusted R-squared: 0.9653

F-statistic: 140.1 on 6 and 24 DF, p-value: < 2.2e-16

We will look into how the quantities are derived in the following.

1.4 Properties of Least Squares Estimates

As an introductory remark, we point out that the least squares estimates are random variables: for new data from the same data-generating mechanism, the data would look differently every time and hence also the least squares estimates. Figure 1.3 displays three least squares regression lines which are based on three different realizations from the same data-generating model (i.e., three simulations from a model). We see that the estimates are varying: they are random themselves!

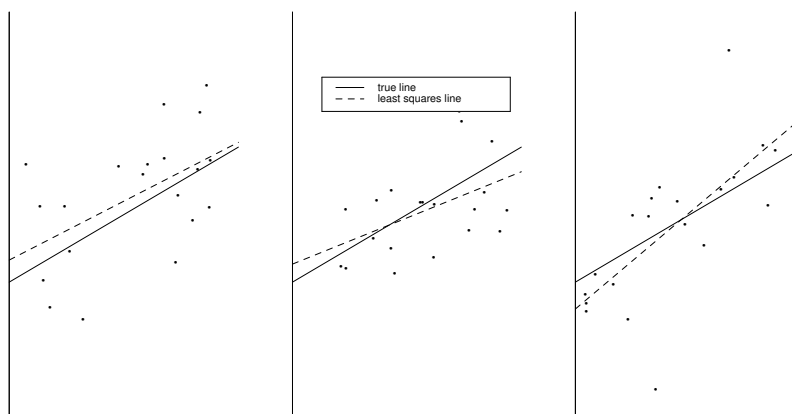


Figure 1.3: Three least squares estimated regression lines for three different data realizations from the same model.

1.4.1 Moments of least squares estimates

We assume here the usual linear model

$$\mathbf{Y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \quad \mathbb{E}[\boldsymbol{\varepsilon}] = \mathbf{0}, \quad \text{Cov}(\boldsymbol{\varepsilon}) = \mathbb{E}[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\top] = \sigma^2 I_{n \times n}. \quad (1.7)$$

This means that assumption 1.-4. from section 1.3.2 are satisfied.

It can then be shown that:

- (i) $\mathbb{E}[\hat{\boldsymbol{\beta}}] = \boldsymbol{\beta}$: that is, $\hat{\boldsymbol{\beta}}$ is **unbiased**
- (ii) $\mathbb{E}[\hat{\mathbf{Y}}] = \mathbb{E}[\mathbf{Y}] = X\boldsymbol{\beta}$ which follows from (i). Moreover, $\mathbb{E}[\mathbf{r}] = \mathbf{0}$.
- (iii) $\text{Cov}(\hat{\boldsymbol{\beta}}) = \sigma^2(X^\top X)^{-1}$
- (iv) $\text{Cov}(\hat{\mathbf{Y}}) = \sigma^2 P$, $\text{Cov}(\mathbf{r}) = \sigma^2(I - P)$

The residuals (which are estimates of the unknown errors ε_i) are also having expectation zero but they are not uncorrelated:

$$\text{Var}(r_i) = \sigma^2(1 - P_{ii}).$$

From this, we obtain

$$\mathbb{E}\left[\sum_{i=1}^n r_i^2\right] = \sum_{i=1}^n \mathbb{E}[r_i^2] = \sum_{i=1}^n \text{Var}(r_i) = \sigma^2 \sum_{i=1}^n (1 - P_{ii}) = \sigma^2(n - \text{tr}(P)) = \sigma^2(n - p).$$

Therefore, $\mathbb{E}[\hat{\sigma}^2] = \mathbb{E}\left[\sum_{i=1}^n r_i^2 / (n - p)\right] = \sigma^2$ is **unbiased**.

1.4.2 Distribution of least squares estimates assuming Gaussian errors

We assume the linear model as in (1.7) but require in addition that $\varepsilon_1, \dots, \varepsilon_n$ i.i.d. $\sim \mathcal{N}(0, \sigma^2)$. It can then be shown that:

- (i) $\hat{\boldsymbol{\beta}} \sim \mathcal{N}_p(\boldsymbol{\beta}, \sigma^2(X^\top X)^{-1})$
- (ii) $\hat{\mathbf{Y}} \sim \mathcal{N}_n(X\boldsymbol{\beta}, \sigma^2 P)$, $\mathbf{r} \sim \mathcal{N}_n(\mathbf{0}, \sigma^2(I - P))$
- (iii) $\hat{\sigma}^2 \sim \frac{\sigma^2}{n-p} \chi_{n-p}^2$.

The normality assumptions of the errors ε_i is often not (approximately) fulfilled in practice. We can then rely on the central limit theorem which implies that for large sample size n , the properties (i)-(iii) above are still approximately true. This is the usual justification in practice to use these properties for constructing confidence intervals and tests for the linear model parameters. However, it is often much better to use **robust methods** in case of non-Gaussian errors which we are not discussing here.

1.5 Tests and Confidence Regions

1.5.1 Testing regression coefficients

We assume the linear model as in (1.7) with $\varepsilon_1, \dots, \varepsilon_n$ i.i.d. $\sim \mathcal{N}(0, \sigma^2)$ (or with ε_i 's i.i.d. and "large" sample size n). As we have seen above, the parameter estimates $\hat{\boldsymbol{\beta}}$ are normally distributed.

If we are interested whether the j th predictor variable is relevant, we can test the null-hypothesis $H_{0,j} : \beta_j = 0$ against the alternative $H_{A,j} : \beta_j \neq 0$. We can then easily derive from the normal distribution of $\hat{\beta}_j$ that

$$\frac{\hat{\beta}_j}{\sqrt{\sigma^2(X^\top X)_{jj}^{-1}}} \sim \mathcal{N}(0, 1) \text{ under the null-hypothesis } H_{0,j}.$$

Since σ^2 is unknown, this quantity is not useful, but if we substitute it with the estimate $\hat{\sigma}^2$ we obtain the so-called t -test statistic

$$T_j = \frac{\hat{\beta}_j}{\sqrt{\hat{\sigma}^2(X^\top X)_{jj}^{-1}}} \sim t_{n-p} \text{ under the null-hypothesis } H_{0,j}, \quad (1.8)$$

which has a slightly different distribution than standard Normal $\mathcal{N}(0, 1)$. The corresponding test is then called the t -test. In practice, we can thus quantify the relevance of individual predictor variables by looking at the size of the test-statistics T_j ($j = 1, \dots, p$) or at the corresponding P -values which may be more informative.

The problem by looking at *individual* tests $H_{0,j}$ is (besides the multiple testing problem in general) that it can happen that all individual tests do not reject the null-hypotheses (say at the 5% significance level) although it is true that some predictor variables have a significant effect. This “paradox” can occur because of correlation among the predictor variables.

An individual t -test for $H_{0,j}$ should be interpreted as quantifying the effect of the j th predictor variable after having subtracted the linear effect of all other predictor variables on Y .

Similarly to the t -tests as in (1.8), one can derive confidence intervals for the unknown parameters β_j :

$$\hat{\beta}_j \pm \sqrt{\hat{\sigma}^2(X^\top X)_{jj}^{-1}} \cdot t_{n-p; 1-\alpha/2}$$

is a two-sided confidence interval which covers the true β_j with probability $1 - \alpha$; here, $t_{n-p; 1-\alpha/2}$ denotes the $1 - \alpha/2$ quantile of a t_{n-p} distribution.

1.5.2 Preview: computational alternative via bootstrap

The above approach to compute confidence intervals and uncertainties of estimates is developed for linear models. There is a simple alternative via the so-called bootstrap, which we will look at more closely later. The rough idea is to subsample the observations and compute the estimator on each subsampled estimate. The variance of the estimators over these so-called bootstrap samples is then an approximation to the true variance of the estimator.

```
> n <- nrow(asphalt1)
> p <- length(coefficients(asphMod)) # length of "beta"
> nBoot <- 1000 # number of bootstrap samples
> coefficientMatrix <- matrix(nrow=nBoot, ncol=p)
> for (iB in 1:nBoot) {
+   bootInd <- sample(1:n, n, replace=TRUE) ## resample from observations
+   lmodB <- lm(LOGRUT ~ . , data = asphalt1[bootInd, ])
+   coefficientMatrix[iB, ] <- coef(lmodB)
```

```

+ }
> ## standard deviation of coefficients across bootstrap samples
> sdCoeff <- apply(coefficientMatrix, 2, sd)
> round(sdCoeff*1000)/1000

[1] 3.166 0.077 0.341 0.289 0.327 0.007 0.127

```

The original output was

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	-5.781239	2.459179	-2.351	0.027280	*
LOGVISC	-0.513325	0.073056	-7.027	2.9e-07	***
ASPH	1.146898	0.265572	4.319	0.000235	***
BASE	0.232809	0.326528	0.713	0.482731	
RUN	-0.618893	0.294384	-2.102	0.046199	*
FINES	0.004343	0.007881	0.551	0.586700	
VOIDS	0.316648	0.110329	2.870	0.008433	**

and the bootstrap approximation to the standard deviation is hence an approximate estimate of the “Std. Error” column. This is not necessary for the linear model (since we know the exact answer under the made assumptions) but the bootstrap is a versatile tool which can be employed in more general settings.

1.5.3 Test of global null hypothesis

To test whether there exists *any* effect from the predictor variables, we can look at the simultaneous null-hypothesis $H_0 : \beta_2 = \dots = \beta_p = 0$ versus the alternative $H_A : \beta_j \neq 0$ for at least one $j \in \{2, \dots, p\}$; we assume here that the first predictor variable is the constant $X_{i,1} \equiv 1$ (there are $p - 1$ (non-trivial) predictor variables). Such a test can be developed with an analysis of variance (*anova*) decomposition which takes a simple form for this special case:

$$\|\mathbf{Y} - \bar{\mathbf{Y}}\|^2 = \|\hat{\mathbf{Y}} - \bar{\mathbf{Y}}\|^2 + \|\mathbf{Y} - \hat{\mathbf{Y}}\|^2$$

which decomposes the total squared error $\mathbf{Y} - \bar{\mathbf{Y}}$ around the mean $\bar{\mathbf{Y}} = n^{-1} \sum_{i=1}^n Y_i \cdot \mathbf{1}$ as a sum of the squared error due to the regression $\hat{\mathbf{Y}} - \bar{\mathbf{Y}}$ (the amount that the fitted values vary around the global arithmetic mean) and the squared residual error $\mathbf{r} = \mathbf{Y} - \hat{\mathbf{Y}}$. (The equality can be seen most easily from a geometrical point of view: the residuals \mathbf{r} are orthogonal to \mathcal{X} and hence to $\hat{\mathbf{Y}} - \bar{\mathbf{Y}}$). Such a decomposition is usually summarized by an ANOVA table (**AN**alysis **Of** **VA**riance).

	sum of squares	degrees of freedom	mean square	\mathbf{E} [mean square]
regression	$\ \hat{\mathbf{Y}} - \bar{\mathbf{Y}}\ ^2$	$p - 1$	$\ \hat{\mathbf{Y}} - \bar{\mathbf{Y}}\ ^2 / (p - 1)$	$\sigma^2 + \frac{\ \mathbf{E}[\mathbf{Y}] - \mathbf{E}[\bar{\mathbf{Y}}]\ ^2}{p - 1}$
error	$\ \mathbf{Y} - \hat{\mathbf{Y}}\ ^2$	$n - p$	$\ \mathbf{Y} - \hat{\mathbf{Y}}\ ^2 / (n - p)$	σ^2
total around global mean	$\ \mathbf{Y} - \bar{\mathbf{Y}}\ ^2$	$n - 1$	—	—

In case of the global null-hypothesis, there is no effect of any predictor variable and hence $\mathbf{E}[\mathbf{Y}] \equiv \text{const.} = \mathbf{E}[\bar{\mathbf{Y}}]$; therefore, the expected mean square equals σ^2 under H_0 . The idea is now to divide the mean square by the estimate $\hat{\sigma}^2$ to obtain a scale-free quantity: this leads to the so-called *F*-statistic

$$F = \frac{\|\hat{\mathbf{Y}} - \bar{\mathbf{Y}}\|^2 / (p - 1)}{\|\mathbf{Y} - \hat{\mathbf{Y}}\|^2 / (n - p)} \sim F_{p-1, n-p} \text{ under the global null-hypothesis } H_0.$$

This test is called the F -test (it is one among several other F -tests in regression).

Besides performing a global F -test to quantify the statistical significance of the predictor variables, we often want to describe the goodness of fit of the linear model for explaining the data. A meaningful quantity is the coefficient of determination, abbreviated by R^2 ,

$$R^2 = \frac{\|\hat{\mathbf{Y}} - \bar{\mathbf{Y}}\|^2}{\|\mathbf{Y} - \bar{\mathbf{Y}}\|^2}.$$

This is the proportion of the total variation of \mathbf{Y} around $\bar{\mathbf{Y}}$ which is explained by the regression (see the ANOVA decomposition and table above).

1.5.4 Permutation test as computational alternative

Instead of the approach above of testing the global null hypothesis via an F -test, there is a simple computational alternative for testing the global null. As statistic, we check again the mean squared residual:

```
> MSEfit <- mean( (fitted(asphMod) - asphalt1[, "LOGRUT"])^2 )
> print(MSEfit)
```

```
[1] 0.05250087
```

If the global null hypothesis were true (and the other linear model assumptions are true), then the outcome variable is independent of all predictor variables. If we condition on the observed predictor variable matrix X and the observed outcomes Y , then any permutation of the outcomes Y is equally likely under the global null. We can thus get in a computational approach the distribution of the mean squared error under the global null by permuting the outcome several times and observing the distribution of the mean squared error under this permutation distribution. The p-value of the global null hypothesis is then approximated as the number of permutations for which we see a lower mean squared error in permutations compared to the real observations.

```
> nPerm <- 10^6 # number of permutations
> mse <- numeric(nPerm) # Mean Squared Error for permutations

> for (p in 1:nPerm) { # for p-th permutation :
+   permutedData <- asphalt1
+   permutedData[, "LOGRUT"] <- permY <- sample(asphalt1[, "LOGRUT"])
+   lmPerm <- lm(LOGRUT ~ . , data = permutedData)
+   mse[p] <- mean( (fitted(lmPerm) - permY)^2 )
+ }

> mean(mse <= MSEfit) # the p-value =   #{mse <= *} / #total
```

```
[1] 0
```

Here, none of the 10^6 permutations yielded a lower MSE than the actually observed one. The p-value would be the probability of mean squared error being lower under a random permutation and we have here just a million random permutations so we would typically say that the p-value is smaller than say 10^{-6} (or 10^{-5} to be on the safe side).

1.6 Analysis of residuals and checking of model assumptions

The residuals $r_i = Y_i - \hat{Y}_i$ can serve as an approximation of the unobservable error term ε_i and for checking whether the linear model is appropriate.

1.6.1 The Tukey-Anscombe Plot

The Tukey-Anscombe is a graphical tool: we plot the residuals r_i (on the y -axis) versus the fitted values \hat{Y}_i (on the x -axis). A reason to plot against the fitted values \hat{Y}_i is that the sample correlation between r_i and \hat{Y}_i is always zero.

In the ideal case, the points in the Tukey-Anscombe plot “fluctuate randomly” around the horizontal line through zero: see also Figure 1.4. An often encountered deviation is non-constant variability of the residuals, i.e., an indication that the variance of ε_i increases with the response variable Y_i : this is shown in Figure 1.5 a)–c). If the Tukey-Anscombe plot shows a trend, there is some evidence that the linear model assumption is not correct (the expectation of the error is not zero which indicates a systematic error): Figure 1.5d) is a typical example.

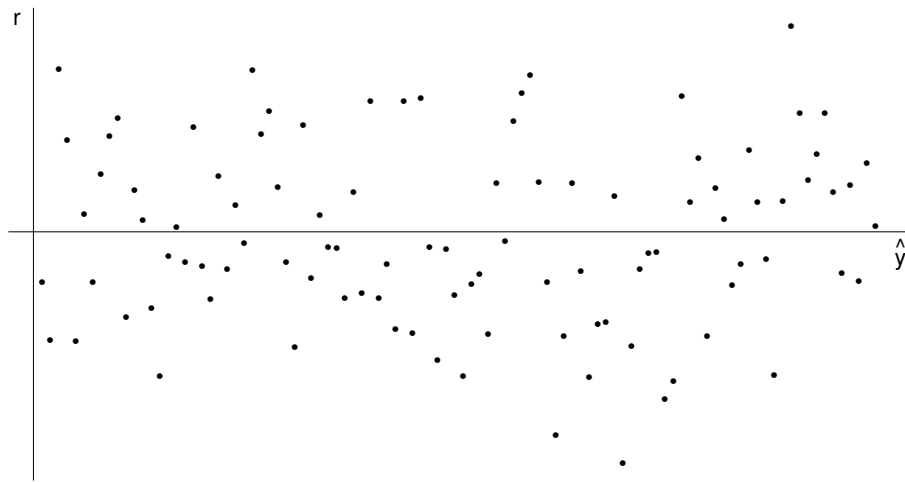


Figure 1.4: Ideal Tukey-Anscombe plot: no violations of model assumptions.

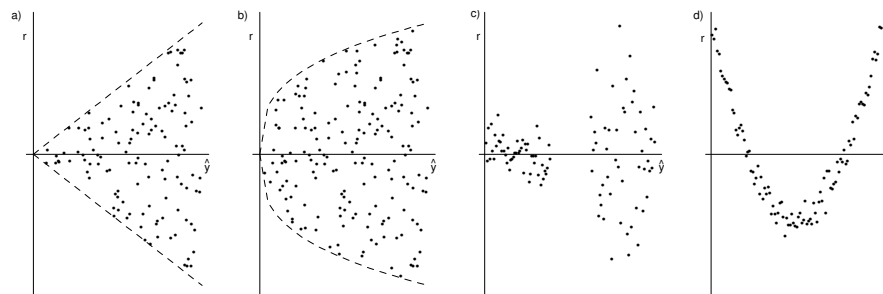


Figure 1.5: a) linear increase of standard deviation, b) nonlinear increase of standard deviation, c) 2 groups with different variances, d) missing quadratic term in the model.

In case where the Tukey-Anscombe plot exhibits a systematic relation of the variability on the fitted values \hat{Y}_i , we should either transform the response variable or perform a weighted regression (see Section 1.6.4). If the standard deviation grows linearly with the fitted values (as in Figure 1.5a)), the log-transform $Y \mapsto \log(Y)$ stabilizes the variance; if

the standard deviation grows as the square root with the values \hat{Y}_i (as in Figure 1.5b)), the square root transformation $Y \mapsto \sqrt{Y}$ stabilizes the variance.

1.6.2 The Normal Plot

Assumptions for the distribution of random variables can be graphically checked with the QQ (quantile-quantile) plot. In the special case of checking for the normal distribution, the QQ plot is also referred to as a normal plot.

In the linear model application, we plot the empirical quantiles of the residuals (on the y axis) versus the theoretical quantiles of a $\mathcal{N}(0,1)$ distribution (on the x axis). If the residuals were normally distributed with expectation μ and variance σ^2 , the normal plot would exhibit an approximate straight line with intercept μ and slope σ . Figures 1.6 and 1.7 show some normal plots with exactly normally and non-normally distributed observations.

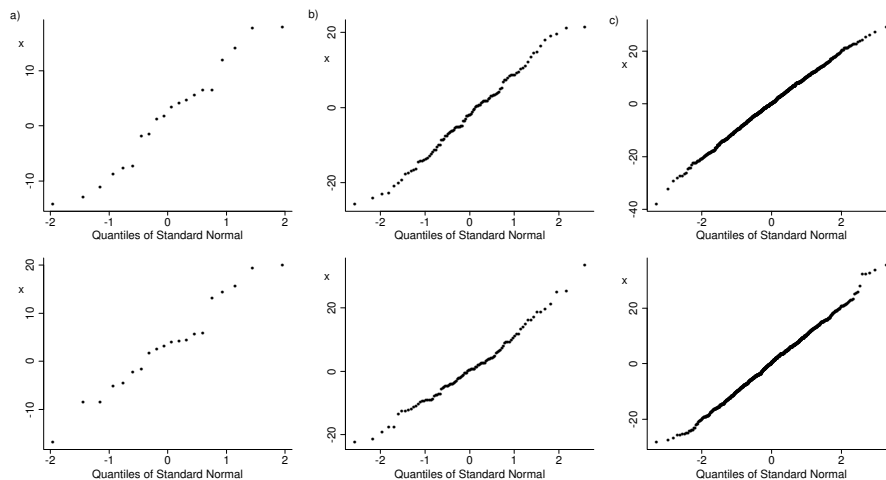


Figure 1.6: QQ-plots for i.i.d. normally distributed random variables. Two plots for each sample size n equal to a) 20, b) 100 and c) 1000.

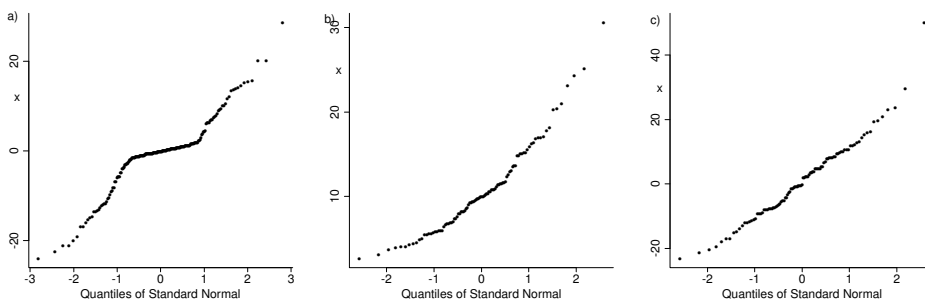


Figure 1.7: QQ-plots for a) long-tailed distribution, b) skewed distribution, c) dataset with outlier.

1.6.3 Plot for detecting serial correlation

For checking independence of the errors we plot the residuals r_i versus the observation number i (or if available, the time t_i of recording the i th observation). If the residuals vary randomly around the zero line, there are no indications for serial correlations among

the errors ε_i . On the other hand, if neighboring (with respect to the x -axis) residuals look similar, the independence assumption for the errors seems violated.

1.6.4 Generalized least squares and weighted regression

In a more general situation, the errors are correlated with known covariance matrix,

$$\mathbf{Y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \quad \text{with } \boldsymbol{\varepsilon} \sim \mathcal{N}_n(\mathbf{0}, \boldsymbol{\Sigma}).$$

When $\boldsymbol{\Sigma}$ is known (and also in the case where $\boldsymbol{\Sigma} = \sigma^2 G$ with unknown σ^2), this case can be transformed to the i.i.d. one, using a “square root” C such that $\boldsymbol{\Sigma} = CC^\top$ (defined, e.g., via Cholesky factorization $\boldsymbol{\Sigma} = LL^\top$, L is uniquely determined lower triangular): If $\tilde{\mathbf{Y}} := C^{-1}\mathbf{Y}$ and $\tilde{X} := C^{-1}X$, we have $\tilde{\mathbf{Y}} = \tilde{X}\boldsymbol{\beta} + \tilde{\boldsymbol{\varepsilon}}$, where $\tilde{\boldsymbol{\varepsilon}} \sim \mathcal{N}(\mathbf{0}, I)$. This leads to the *generalized least squares* solution $\hat{\boldsymbol{\beta}} = (X^\top \boldsymbol{\Sigma}^{-1} X)^{-1} X^\top \boldsymbol{\Sigma}^{-1} \mathbf{Y}$ with $\text{Cov}(\hat{\boldsymbol{\beta}}) = (X^\top \boldsymbol{\Sigma}^{-1} X)^{-1}$.

A special case where $\boldsymbol{\Sigma}$ is *diagonal*, $\boldsymbol{\Sigma} = \sigma^2 \text{diag}(z_1, z_2, \dots, z_n)$ (with trivial inverse) is the *weighted* least squares problem $\min_{\boldsymbol{\beta}} \sum_{i=1}^n w_i (Y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2$, with weights $w_i \equiv 1/z_i$.

1.7 Model Selection

We assume the linear model

$$Y_i = \sum_{j=1}^p \beta_j x_{ij} + \varepsilon_i \quad (i = 1, \dots, n),$$

where $\varepsilon_1, \dots, \varepsilon_n$ i.i.d., $\mathbb{E}[\varepsilon_i] = 0$, $\text{Var}(\varepsilon_i) = \sigma^2$.

Problem: Which of the predictor variables should be used in the linear model?

It may be that not all of the p predictor variables are relevant. In addition, every coefficient has to be estimated and thus is afflicted with variability: the individual variabilities for each coefficient sum up and **the variability of the estimated hyper-plane increases the more predictors are entered into the model, whether they are relevant or not**. The aim is often to look for the **optimal** or **best** - not the true - model.

What we just explained in words can be formalized a bit more. Suppose we are looking for optimizing the prediction

$$\sum_{r=1}^q \hat{\beta}_{j_r} x_{ij_r} =: \hat{m}_q(\mathbf{x}_i) = \hat{m}_{\{j_1, \dots, j_q\}}(\mathbf{x}_i)$$

which includes q predictor variables with indices $j_1, \dots, j_q \in \{1, \dots, p\}$. The average mean squared error of this prediction is

$$\begin{aligned} \text{aMSE} &= n^{-1} \sum_{i=1}^n \mathbb{E}[(m(\mathbf{x}_i) - \hat{m}_q(\mathbf{x}_i))^2] \\ &= n^{-1} \sum_{i=1}^n (\mathbb{E}[\hat{m}_q(\mathbf{x}_i)] - m(\mathbf{x}_i))^2 + n^{-1} \underbrace{\sum_{i=1}^n \text{Var}(\hat{m}_q(\mathbf{x}_i))}_{=\frac{q}{n}\sigma^2}, \end{aligned} \quad (1.9)$$

where $m(\cdot)$ denotes the regression function in the full model (“unknown true model”) with all the predictor variables. It is plausible that the systematic error (squared bias) $n^{-1} \sum_{i=1}^n (\mathbb{E}[\hat{m}_q(\mathbf{x}_i)] - m(\mathbf{x}_i))^2$ decreases as the number of predictor variables q increases

(i.e., with respect to bias, we have nothing to lose by using as many predictors as we can), but the variance term increases linearly in the number of predictors q (the variance term equals $q/n \cdot \sigma^2$ which is not too difficult to derive). This is the so-called **bias-variance trade-off** which is present in very many other situations and applications in statistics. Finding the best model thus means to optimize the bias-variance trade-off: this is sometimes also referred to as “regularization” (for avoiding a too complex model).

1.7.1 Penalized regression – Mallows C_p statistic

The mean squared error in (1.9) is unknown: we do not know the magnitude of the bias term but fortunately, we can estimate the mean squared error.

Denote by $SSE(\mathcal{M}) = \|Y - X_{\mathcal{M}}\hat{\beta}_{\mathcal{M}}\|^2$ the residual sum of squares in a model \mathcal{M} , where here \mathcal{M} corresponds to a subset of all available predictor variables $\{1, \dots, p\}$. The value of $SSE(\mathcal{M})$ is not a good measure to estimate the mean squared error in (1.9). For example, $SSE(\mathcal{M})$ becomes smaller the bigger the model \mathcal{M} and the biggest model under consideration has the lowest SSE (which generally contradicts the equation in (1.9)).

One option is to use a regularized residual sum of squares. Instead of estimating the regression coefficients in least squares fashion as

$$\hat{\beta} = \operatorname{argmin}_{\beta} \|Y - X\beta\|^2,$$

we add a term penalizing the number of non-zero entries in the regression coefficient vector (as zero entries mean the corresponding variables are effectively dropped from the regression):

$$\hat{\beta}(\lambda) = \operatorname{argmin}_{\beta} \|Y - X\beta\|^2 + \lambda \|\beta\|_0, \quad (1.10)$$

where $\|\beta\|_0 = |\{k : \beta_k \neq 0\}|$, the number of non-zero coefficients. The larger the value of λ , the fewer variables will be selected. For $\lambda = 0$, we recover least squares regression.

Popular values of λ are based on Akaike’s information criterion (AIC), which is *equivalent* to Mallows C_p for linear Gaussian models, and amounts to a choice $\lambda = 2\hat{\sigma}^2$. Another popular choice is the Bayesian information criterion (BIC), which replaces the factor 2 with $\log(n)$. In practice, however, the regularization λ is often chosen via cross-validation, see Ch. 4.

Searching for the best model

If the full model has p predictor variables, there are $2^p - 1$ sub-models (every predictor can be in or out but we exclude the sub-model \mathcal{M} which corresponds to the empty set).

Therefore, an exhaustive search for the sub-model \mathcal{M} minimizing the regularized residual sum of squares is only feasible if p is less than say 16 ($2^{16} - 1 = 65'535$ which is already fairly large). If p is “large”, we can proceed with *stepwise* algorithms.

Forward selection:

1. Start with the smallest model \mathcal{M}_0 (location model) as the current model.
2. Include the predictor variable to the current model which reduces the residual sum of squares most.
3. Continue step 2. until all predictor variables have been chosen or until a large number of predictor variables has been selected. This produces a sequence of sub-models $\mathcal{M}_0 \subseteq \mathcal{M}_1 \subseteq \mathcal{M}_2 \subseteq \dots$.
4. Choose the model in the sequence $\mathcal{M}_0 \subseteq \mathcal{M}_1 \subseteq \mathcal{M}_2 \subseteq \dots$ which has smallest regularized sum of squares (for chosen value of λ).

Backward selection:

1. Start with the full model \mathcal{M}_0 as the current model.
2. Exclude the predictor variable from the current model which increases the residual sum of squares the least.
3. Continue step 2. until all predictor variables have been deleted (or a large number of predictor variables). This produces a sequence of sub-models $\mathcal{M}_0 \supseteq \mathcal{M}_1 \supseteq \mathcal{M}_2 \supseteq \dots$
4. Choose the model in the sequence $\mathcal{M}_0 \supseteq \mathcal{M}_1 \supseteq \mathcal{M}_2 \supseteq \dots$ which has smallest regularized sum of squares (for chosen value of λ).

Considerations: Backward selection is typically a bit better than forward selection but it is computationally more expensive. Also, in case where $p \geq n$, we don't want to fit the full model and forward selection is an appropriate way to proceed, but see Section 7.7.

Note that the data is heavily used for selecting the final model and therefore, the p-values and similar inferences are *not* valid anymore and effects typically look too significant.

R's `step()` function for stepwise model selection by default uses a *combination* of forward and backward selection called "*both*".

Chapter 2

Nonparametric Density Estimation

2.1 Introduction

For a moment, we will go back to simple data structures: we have observations which are realizations of univariate random variables,

$$X_1, \dots, X_n \text{ i.i.d. } \sim F,$$

where F denotes an unknown cumulative distribution function. The goal is to estimate the distribution F . In particular, we are interested in estimating the density $f = F'$, assuming that it exists.

Instead of assuming a parametric model for the distribution (e.g. Normal distribution with unknown expectation and variance), we rather want to be “as general as possible”: that is, we only assume that the density exists and is suitably smooth (e.g. differentiable). It is then possible to estimate the unknown density **function** $f(\cdot)$. Mathematically, a function is an **infinite-dimensional** object. Density estimation will become a “basic principle” how to do estimation for infinite-dimensional objects. We will make use of such a principle in many other settings such as nonparametric regression with one predictor variable (Chapter 3) and flexible regression and classification methods with many predictor variables (Chapter 7).

2.2 Estimation of a density

We consider the data which records the duration of eruptions of “Old Faithful”, a famous geysir in Yellowstone National Park (Wyoming, USA). You can see pictures or watch it via webcam on <http://www.nps.gov/yell/>¹.

2.2.1 Histogram

The histogram is the oldest and most popular density estimator. We need to specify an “*origin*” x_0 and the *class width* h for the specifications of the intervals

$$I_j = (x_0 + j \cdot h, x_0 + (j + 1)h] \quad (j = \dots, -1, 0, 1, \dots)$$

for which the histogram counts the number of observations falling into each I_j : we then plot the histogram such that the area of each bar is proportional to the number of observations falling into the corresponding class (interval I_j).

¹or a sample from 2007 on <https://www.youtube.com/watch?v=tShhZvvIM84>.

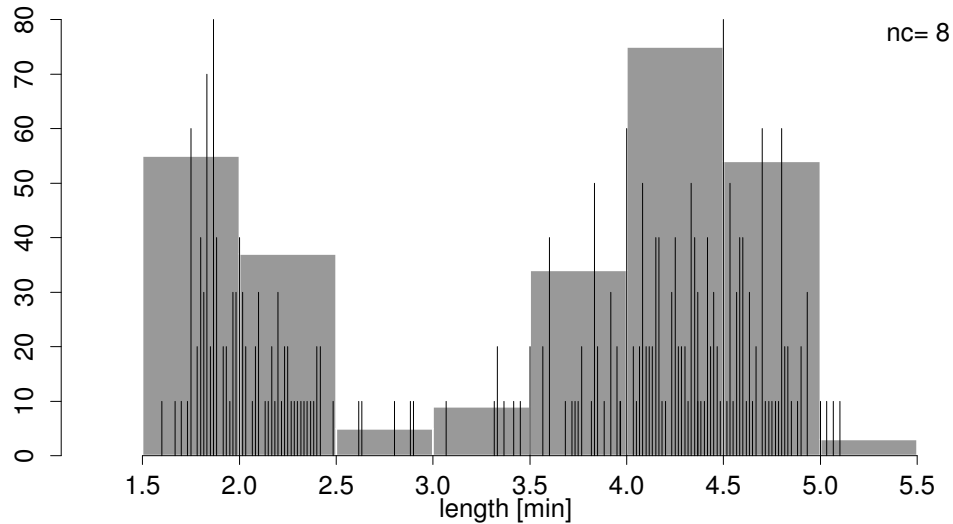
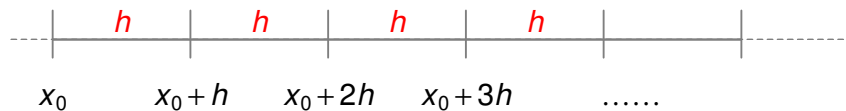


Figure 2.1: Histograms (different class widths) for durations of eruptions of “Old Faithful” geysir in Yellowstone Park ($n = 272$, `data(faithful)`).



The choice of the “origin” x_0 is highly arbitrary, whereas the role of the class width is immediately clear for the user. The form of the histogram depends very much on these two tuning parameters.

2.2.2 Kernel estimator

The naive estimator

Similar to the histogram, we can compute the relative frequency of observations falling into a small region. The density function $f(\cdot)$ at a point x can be represented as

$$f(x) = \lim_{h \rightarrow 0} \frac{1}{2h} \mathbb{P}[x - h < X \leq x + h]. \quad (2.1)$$

The naive estimator is then constructed without taking the limit in (2.1) and by replacing probabilities with relative frequencies:

$$\hat{f}(x) = \frac{1}{2hn} \#\{i; X_i \in (x - h, x + h]\}. \quad (2.2)$$

This naive estimator is only piecewise constant since every X_i is either in or out of the interval $(x - h, x + h]$. As for histograms, we also need to specify the so-called bandwidth h ; but in contrast to the histogram, we do not need to specify an origin x_0 .

An alternative representation of the naive estimator (2.2) is as follows. Define the weight function

$$w(x) = \begin{cases} 1/2 & \text{if } |x| \leq 1, \\ 0 & \text{otherwise.} \end{cases}$$

Then,

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^n w\left(\frac{x - X_i}{h}\right).$$

If we choose instead of the rectangle weight function $w(\cdot)$ a general, typically more smooth kernel function $K(\cdot)$, we have the definition of the kernel density estimator

$$\begin{aligned} \hat{f}(x) &= \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - X_i}{h}\right), \\ K(x) &\geq 0, \int_{-\infty}^{\infty} K(x)dx = 1, K(x) = K(-x). \end{aligned} \quad (2.3)$$

The estimator depends on the *bandwidth* $h > 0$ which acts as a tuning parameter. For large bandwidth h , the estimate $\hat{f}(x)$ tends to be very slowly varying as a function of x , while small bandwidths will produce a more wiggly function estimate. The positivity of the kernel function $K(\cdot)$ guarantees a positive density estimate $\hat{f}(\cdot)$ and the normalization $\int K(x)dx = 1$ implies that $\int \hat{f}(x)dx = 1$ which is necessary for $\hat{f}(\cdot)$ to be a density. Typically, the kernel function $K(\cdot)$ is chosen as a probability density which is symmetric around 0.

The smoothness of $\hat{f}(\cdot)$ is inherited from the smoothness of the kernel: if the r th derivative $K^{(r)}(x)$ exists for all x , then $\hat{f}^{(r)}(x)$ exists as well for all x (easy to verify using the chain rule for differentiation).

Popular kernels are the Gaussian kernel

$$K(x) = \varphi(x) = (2\pi)^{-\frac{1}{2}} e^{-x^2/2} \text{ (the density of the } \mathcal{N}(0, 1) \text{ distribution)}$$

or a kernel with finite support such as $K(x) = \frac{\pi}{4} \cos(\frac{\pi}{2}x) \mathbb{1}\{|x| \leq 1\}$. The Epanechnikov kernel, which is optimal with respect to mean squared error, is

$$K(x) = \frac{3}{4} (1 - |x|^2) \mathbb{1}\{|x| \leq 1\}.$$

But far more important than the kernel is the bandwidth h , see figure 2.2: its role and how to choose it are discussed below.

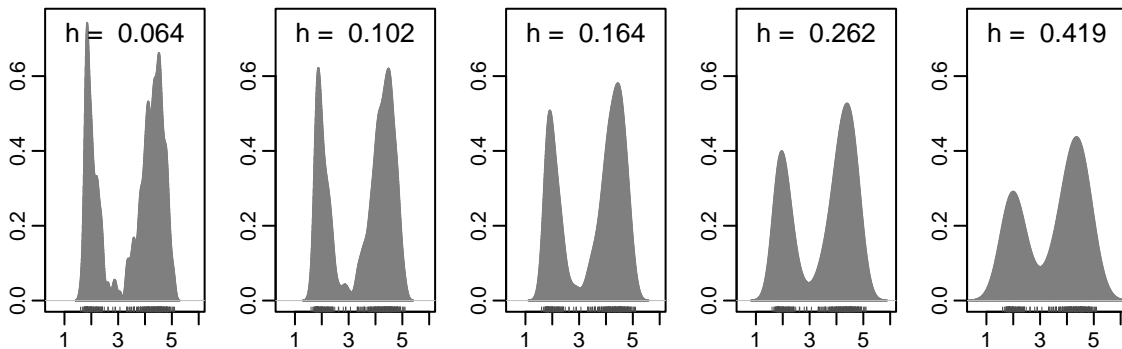


Figure 2.2: kernel density estimates of the “Old Faithful” eruption lengths; Gaussian kernel and bandwidths $h = 0.04 \cdot 1.6^{1,2,\dots,5}$.

2.3 The role of the bandwidth

The bandwidth h is often also called the “smoothing parameter”: a moment of thought will reveal that for $h \rightarrow 0$, we will have “ δ -spikes” at every observation X_i , whereas $\hat{f}(\cdot) = \hat{f}_h(\cdot)$ becomes smoother as h is increasing.

2.3.1 Variable bandwidths: k nearest neighbors

Instead of using a global bandwidth, we can use locally changing bandwidths $h(x)$.

The general idea is to use a large bandwidth for regions where the data is sparse. The k -nearest neighbor idea is to choose

$$h(x) = \text{Euclidean distance of } x \text{ to the } k\text{th nearest observation,}$$

where k is regulating the magnitude of the bandwidth. Note that generally, $\hat{f}_{h(x)}(\cdot)$ will not be a density anymore since the integral is not necessarily equal to one.

2.3.2 The bias-variance trade-off

We can formalize the behavior of $\hat{f}(\cdot)$ when varying the bandwidth h in terms of bias and variance of the estimator. It is important to understand heuristically that

the (absolute value of the) bias of \hat{f} increases and the variance of \hat{f} decreases as h increases.

Therefore, if we want to minimize the mean squared error $\text{MSE}(x)$ at a point x ,

$$\text{MSE}(x) = \mathbb{E} \left[\left(\hat{f}(x) - f(x) \right)^2 \right] = \left(\mathbb{E}[\hat{f}(x)] - f(x) \right)^2 + \text{Var}(\hat{f}(x)),$$

we are confronted with a **bias-variance trade-off**. As a consequence, this allows - at least conceptually - to optimize the bandwidth parameter (namely to minimize the mean squared error) in a well-defined, coherent way. Instead of optimizing the mean squared error at a point x , one may want to optimize the integrated mean squared error (IMSE)

$$\text{IMSE} = \int \text{MSE}(x) dx$$

which yields an integrated decomposition of squared bias and variance (integration is over the support of X). Since the integrand is non-negative, the order of integration (over the support of X and over the probability space of X) can be reversed, denoted as MISE (mean integrated squared error) and written as

$$\text{MISE} = \mathbb{E} \left[\int \left(\hat{f}(x) - f(x) \right)^2 dx \right] = \mathbb{E}[\text{ISE}], \quad (2.4)$$

where $\text{ISE} = \int \left(\hat{f}(x) - f(x) \right)^2 dx$.

2.3.3 Asymptotic bias and variance

It is straightforward (using definitions) to give an expression for the exact bias and variance:

$$\begin{aligned} \mathbb{E}[\hat{f}(x)] &= \int \frac{1}{h} K \left(\frac{x-y}{h} \right) f(y) dy \\ \text{Var}(\hat{f}(x)) &= \frac{1}{nh^2} \text{Var} \left(K \left(\frac{x-X_i}{h} \right) \right) = \frac{1}{nh^2} \mathbb{E} \left[K \left(\frac{x-X_i}{h} \right)^2 \right] - \frac{1}{nh^2} \mathbb{E} \left[K \left(\frac{x-X_i}{h} \right) \right]^2 \\ &= n^{-1} \int \frac{1}{h^2} K \left(\frac{x-y}{h} \right)^2 f(y) dy - n^{-1} \left(\int \frac{1}{h} K \left(\frac{x-y}{h} \right) f(y) dy \right)^2. \end{aligned} \quad (2.5)$$

For the bias we therefore get (by a change of variable and $K(-z) = K(z)$)

$$\begin{aligned} \text{Bias}(x) &= \int \frac{1}{h} K\left(\frac{x-y}{h}\right) f(y) dy - f(x) \\ &\stackrel{\underbrace{z=(y-x)/h}}{=} \int K(z) f(x+hz) dz - f(x) = \int K(z) (f(x+hz) - f(x)) dz, \end{aligned} \quad (2.6)$$

To approximate this expression in general, we invoke an asymptotic argument. We assume that $h \rightarrow 0$ as sample size $n \rightarrow \infty$, that is:

$$\boxed{h = h_n \rightarrow 0 \text{ with } nh_n \rightarrow \infty.}$$

This will imply that the bias goes to zero since $h_n \rightarrow 0$; the second condition requires that h_n is going to zero more slowly than $1/n$ which turns out to imply that also the variance of the estimator will go to zero as $n \rightarrow \infty$. To see this, we use a Taylor expansion of f , assuming that f is sufficiently smooth:

$$f(x+hz) = f(x) + hzf'(x) + \frac{1}{2}h^2z^2f''(x) + \dots$$

Plugging this into (2.6) yields

$$\begin{aligned} \text{Bias}(x) &= hf'(x) \underbrace{\int zK(z)dz}_{=0} + \frac{1}{2}h^2f''(x) \int z^2K(z)dz + \dots \\ &= \frac{1}{2}h^2f''(x) \int z^2K(z)dz + \text{higher order terms in } h. \end{aligned}$$

For the variance, we get from (2.5)

$$\begin{aligned} \text{Var}(\hat{f}(x)) &= n^{-1} \int \frac{1}{h^2} K\left(\frac{x-y}{h}\right)^2 f(y) dy - n^{-1} (f(x) + \text{Bias}(x))^2 \\ &= n^{-1}h^{-1} \int f(x-hz)K(z)^2 dz - \underbrace{n^{-1}(f(x) + \text{Bias}(x))^2}_{=O(n^{-1})} \\ &= n^{-1}h^{-1} \int f(x-hz)K(z)^2 dz + O(n^{-1}) = n^{-1}h^{-1}f(x) \int K(z)^2 dz + o(n^{-1}h^{-1}), \end{aligned}$$

assuming that f is smooth and hence $f(x-hz) \rightarrow f(x)$ as $h_n \rightarrow 0$.

In summary: for $h = h_n \rightarrow 0$, $h_n n \rightarrow \infty$ as $n \rightarrow \infty$

$$\boxed{\begin{aligned} \text{Bias}(x) &= h^2 f''(x) \int z^2 K(z) dz / 2 + o(h^2) & (n \rightarrow \infty) \\ \text{Var}(\hat{f}(x)) &= (nh)^{-1} f(x) \int K(z)^2 dz + o((nh)^{-1}) & (n \rightarrow \infty) \end{aligned}}$$

The optimal bandwidth $h = h_n = h_{\text{opt}}(x)$ which minimizes the leading term in the asymptotic $\text{MSE}(x)$ can be calculated straightforwardly by solving $\frac{\partial}{\partial h} \text{MSE}(x) = 0$,

$$h_{\text{opt}}(x) = n^{-1/5} \left(\frac{f(x) \int K^2(z) dz}{(f''(x))^2 \int z^2 K(z) dz} \right)^{1/5}. \quad (2.7)$$

Since it's not straightforward to estimate and use a *local* bandwidth $h(x)$, one rather considers minimizing the MISE, i.e., $\int \text{MSE}(x) dx$ which is *asymptotically*

$$\text{asympt. MISE} = \int \text{Bias}(x)^2 + \text{Var}(\hat{f}(x)) dx = \frac{1}{4} h^4 R(f'') \sigma_K^4 + R(K)/(nh), \quad (2.8)$$

where $R(g) = \int g^2(x) dx$, $\sigma_K^2 = \int x^2 K(x) dx$, and the “global” asymptotically optimal bandwidth becomes

$$h_{\text{opt}} = n^{-1/5} (R(K)/\sigma_K^4 \times 1/R(f''))^{1/5}. \quad (2.9)$$

By replacing h with h_{opt} , e.g., in (2.8), we see that both variance and bias terms are of order $O(n^{-4/5})$, the optimal rate for the MISE and $\text{MSE}(x)$. From section 2.4.1, this rate is also optimal for a much larger class of density estimators.

2.3.4 Estimating the bandwidth

As seen from (2.9), the asymptotically best bandwidth depends on $R(f'') = \int f''^2(x) dx$ which is unknown (whereas $R(K)$ and σ_K^2 are known). It is possible to estimate the f'' again by a kernel estimator with an “initial” bandwidth h_{init} (sometimes called a pilot bandwidth) yielding $\hat{f}_{h_{\text{init}}}''$. Plugging this estimate into (2.9) yields an estimated bandwidth \hat{h} for the density estimator $\hat{f}(\cdot)$ (the original problem): of course, \hat{h} depends on the initial bandwidth h_{init} , but choosing h_{init} in an ad-hoc way is less critical for the density estimator than choosing the bandwidth h itself. Furthermore, methods have been devised to determine h_{init} and h simultaneously (e.g., “Sheather-Jones”, in R using `density(*, bw="SJ")`).

Estimating local bandwidths

Note that the $h_{\text{opt}}(x)$ bandwidth selection in (2.7) is more problematical mainly because $\hat{f}_{h_{\text{opt}}(x)}(x)$ will not integrate to one without further normalization. On the other hand, it can be important to use *locally varying* bandwidths instead of a single global one in a kernel estimator at the expense of being more difficult. The plug-in procedure outlined above can be applied *locally*, i.e., conceptually for each x and hence describes how to estimate local bandwidths from data and how to implement a kernel estimator with locally varying bandwidths. In the related area of nonparametric regression, in section 3.2.1, we will show an example about locally changing bandwidths which are estimated based on an iterative version of the (local) plug-in idea above.

Other density estimators

There are quite a few other approaches to density estimation than the kernel estimators above (whereas in practice, the *fixed* bandwidth kernel estimators are used predominantly because of their simplicity). An important approach in particular aims to estimate the *log density* $\log f(x)$ (setting $\hat{f} = \exp(\widehat{\log f})$) which has no positivity constraints and whose “normal limit” is a simple quadratic. One good implementation is in Kooperberg’s R package `logspline`, where spline knots are placed in a stepwise algorithm minimizing approximate BIC (or AIC). This is can be seen as another version of locally varying bandwidths.

2.4 Higher dimensions

Quite many applications involve multivariate data. For simplicity, consider data which are i.i.d. realizations of d -dimensional random variables

$$\mathbf{X}_1, \dots, \mathbf{X}_n \text{ i.i.d. } \sim f(x_1, \dots, x_d) dx_1 \cdots dx_d$$

where $f(\cdot)$ denotes the multivariate density.

The multivariate kernel density estimator is, in its simplest form, defined as

$$\hat{f}(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^n K\left(\frac{\mathbf{x} - \mathbf{X}_i}{h}\right),$$

where the kernel $K(\cdot)$ is now a function, defined for d -dimensional \mathbf{x} , satisfying

$$K(\mathbf{u}) \geq 0, \quad \int_{\mathbb{R}^d} K(\mathbf{u}) d\mathbf{u} = 1, \quad \int_{\mathbb{R}^d} \mathbf{u}K(\mathbf{u}) d\mathbf{u} = \mathbf{0}, \quad \int_{\mathbb{R}^d} \mathbf{u}\mathbf{u}^\top K(\mathbf{u}) d\mathbf{u} = I_d.$$

Usually, the kernel $K(\cdot)$ is chosen as a product of a kernel K_{univ} for univariate density estimation

$$K(\mathbf{u}) = \prod_{j=1}^d K_{univ}(u_j).$$

If one additionally desires the multivariate kernel $K(\mathbf{u})$ to be *radially symmetric*, it can be shown that K must be the multivariate normal (Gaussian) density, $K(\mathbf{u}) = c_d \exp(-\frac{1}{2}\mathbf{u}^\top \mathbf{u})$.

2.4.1 The curse of dimensionality

In practice, multivariate kernel density estimation is often restricted to dimension $d = 2$. The reason is, that a higher dimensional space (with d of medium size or large) will be only very sparsely populated by data points. Or in other words, there will be only very few neighboring data points to any value \mathbf{x} in a higher dimensional space, unless the sample size is extremely large. This phenomenon is also called the *curse of dimensionality*.

An implication of the curse of dimensionality is the following lower bound for the best mean squared error of nonparametric density estimators (assuming that the underlying density is twice differentiable): it has been shown that the best possible MSE rate is

$$O(n^{-4/(4+d)}).$$

The following table evaluates $n^{-4/(4+d)}$ for various n and d :

$n^{-4/(4+d)}$	$d = 1$	$d = 2$	$d = 3$	$d = 5$	$d = 10$
$n = 100$	0.025	0.046	0.072	0.129	0.268
$n = 1000$	0.004	0.010	0.019	0.046	0.139
$n = 100'000$	$1.0 \cdot 10^{-4}$	$4.6 \cdot 10^{-4}$	$13.9 \cdot 10^{-4}$	0.006	0.037

Thus, to improve the MSE by a factor 0.1, we need to increase sample size by a factor 17 in $d = 1$, but need an increase in sample size by a factor of about 3160 in dimension $d = 10$, and more than an increase in sample size by 67 million for $d = 100$.

What might still make density estimation possible in higher dimensions is if the data lives on a low-dimensional manifold and we are able to identify that manifold in some suitable way.

Chapter 3

Nonparametric Regression

3.1 Introduction

We consider here nonparametric regression with one predictor variable. Practically relevant generalizations to more than one or two predictor variables are not so easy due to the curse of dimensionality mentioned in section 2.4.1 and often require different approaches, as will be discussed later in Chapter 7.

Figure 3.1 shows (several identical) scatter plots of (x_i, Y_i) ($i = 1, \dots, n$). We can model such data as

$$Y_i = m(x_i) + \varepsilon_i, \quad (3.1)$$

where $\varepsilon_1, \dots, \varepsilon_n$ i.i.d. with $\mathbb{E}[\varepsilon_i] = 0$ and $m : \mathbb{R} \rightarrow \mathbb{R}$ is an “arbitrary” function. The function $m(\cdot)$ is called the nonparametric regression function and it satisfies $m(x) = \mathbb{E}[Y|x]$. The restriction we make for $m(\cdot)$ is that it fulfills some kind of smoothness conditions. The regression function in Figure 3.1 does not appear to be linear in x and linear regression is not a good model. The flexibility to allow for an “arbitrary” regression function is very desirable; but of course, such flexibility has its price, namely an inferior estimation accuracy than for linear regression.

3.2 The kernel regression estimator

We can view the regression function in (3.1) as

$$m(x) = \mathbb{E}[Y|X = x],$$

(assuming that X is random and $X_i = x_i$ are realized values of the random variables). We can express this conditional expectation as

$$\int_{\mathbb{R}} y f_{Y|X}(y|x) dy = \frac{\int_{\mathbb{R}} y f_{X,Y}(x, y) dy}{f_X(x)},$$

where $f_{Y|X}$, $f_{X,Y}$, f_X denote the conditional, joint and marginal densities. We can now plug in the univariate and bivariate kernel density (all with the same univariate kernel K) estimates

$$\hat{f}_X(x) = \frac{\sum_{i=1}^n K\left(\frac{x-x_i}{h}\right)}{nh}, \quad \hat{f}_{X,Y}(x, y) = \frac{\sum_{i=1}^n K\left(\frac{x-x_i}{h}\right) K\left(\frac{y-Y_i}{h}\right)}{nh^2}$$

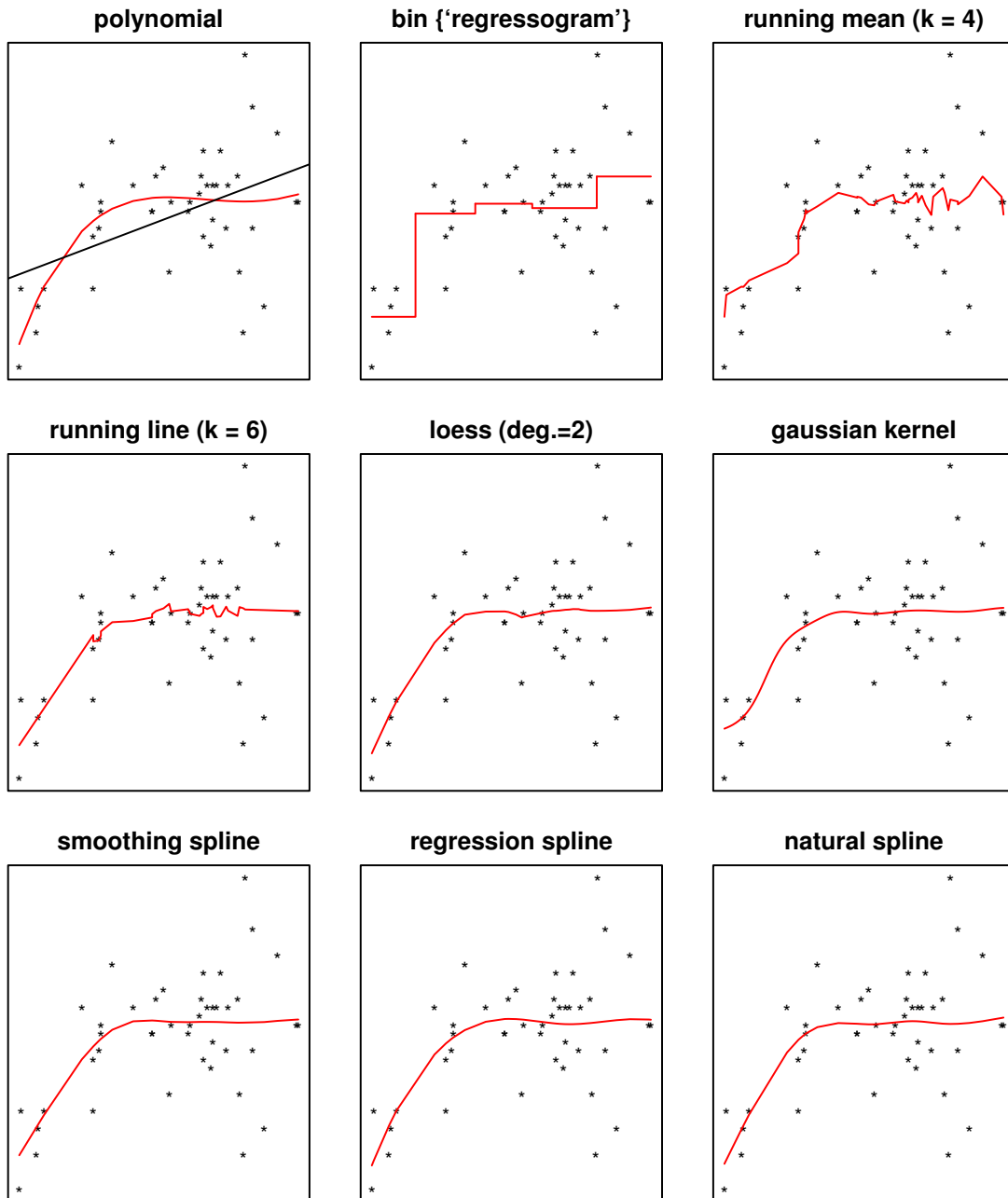


Figure 3.1: Various regression estimators in model $Y_i = m(x_i) + \varepsilon_i$ ($i = 1, \dots, 43$) with response Y a log-concentration of a serum (in connection of Diabetes) and predictor variable x the age in months of children. See Hastie and Tibshirani (1990, p.10). Except for the linear regression fit (top left panel), all other estimators have about 5 degrees of freedom.

into the formula above which yields the so-called Nadaraya-Watson kernel estimator

$$\hat{m}(x) = \frac{\sum_{i=1}^n K((x - x_i)/h) Y_i}{\sum_{i=1}^n K((x - x_i)/h)} = \frac{\sum_{i=1}^n \omega_i Y_i}{\sum_i \omega_i}, \quad (3.2)$$

i.e., a weighted mean of the Y_i where $\omega_i = \omega_i(x)$ is a kernel centered at x_i . An interesting interpretation of the kernel regression estimator in (3.2) is

$$\hat{m}(x) = \arg \min_{m_x \in \mathbb{R}} \sum_{i=1}^n K\left(\frac{x-x_i}{h}\right) (Y_i - m_x)^2. \quad (3.3)$$

This can be easily verified by solving $\frac{d}{dm_x} \sum_{i=1}^n K((x-x_i)/h)(Y_i - m_x)^2 = 0$. Thus, for every fixed x , we are searching for the best local constant m_x such that the localized sum of squares is minimized; localization is here described by the kernel and gives a large weight to those observations (x_i, Y_i) where x_i is close to the point x of interest.

3.2.1 The role of the bandwidth

Analogously as in section 2.3, the bandwidth h controls the bias-variance trade-off: a large bandwidth h implies high bias but small variance, resulting in a slowly varying curve, and vice-versa. We are not showing the computations for $\text{MSE}(x)$, just note that they not only depend on (derivatives of) $m(x)$, but also on $f_X(x)$.

Local bandwidth selection

Similarly as in (2.7), also using $\int uK(u)du = 0$, (assuming *equidistant* x_i , i.e., constant $x_{i+1} - x_i$) there is a formula of the asymptotically best local bandwidth $h_{\text{opt}}(x)$ which depends on $m''(\cdot)$ and the error variance σ_ε^2 :

$$h_{\text{opt}}(x) = n^{-1/5} \left(\frac{\sigma_\varepsilon^2 \int K^2(z) dz}{\{m''(x) \int z^2 K(z) dz\}^2} \right)^{1/5}. \quad (3.4)$$

The locally optimal bandwidth $h_{\text{opt}}(x)$ can then be estimated in an iterative way using the plug-in principle. Roughly speaking, start with an initial bandwidth h_0 to estimate $m''(\cdot)$ (by using an inflated version $n^{1/10}h_0$) and σ_ε^2 ; these estimates can then be used to get a first estimate of $h_{\text{opt}}(x)$. Now use this first bandwidth estimate as the current bandwidth h_1 to estimate again $m''(\cdot)$ (by using the inflated version $n^{1/10}h_1$) and σ_ε^2 , and then obtain new estimates for $h_{\text{opt}}(x)$; and so on, see Brockmann et al. (1993).

Such a procedure has been implemented in R with the function `lokerns` in the package `lokern`. The dataset `cars` contains the distance for stopping as a function of speed of a car. A nonparametric function estimate with locally varying bandwidth can then be obtained as follows:

```
library(lokern); lofit <- lokerns(cars$ speed, cars$ dist)
```

3.2.2 Inference for the underlying regression curve

We consider here the properties, in particular the variability, of the kernel estimator $\hat{m}(x_i)$ at an observed design point x_i .

¹The equidistant case is a simple special case (with $f(x) \equiv 1$) of the general “random design case”, $X_i \sim f(x) dx$, where the design density $f(\cdot)$ plays an important role: The bias, $\mathbb{E}[\hat{m}(x)|X_1, \dots, X_n] - m(x)$, is asymptotically, $B(x, h) = h^2(m''(x) + 2m'(x)f'(x)/f(x)) \cdot \sigma_K^2/2 + o(h^2)$, and the (conditional) variance $V(x, h) = \frac{1}{nh}R(K)/f(x) \cdot \sigma_\varepsilon^2 + o(\frac{1}{nh})$, with the same notation as in (2.8) and (2.9) for kernel *density* estimation, also for the heteroscedastic case where $\sigma_\varepsilon^2 = \sigma_\varepsilon^2(x)$.

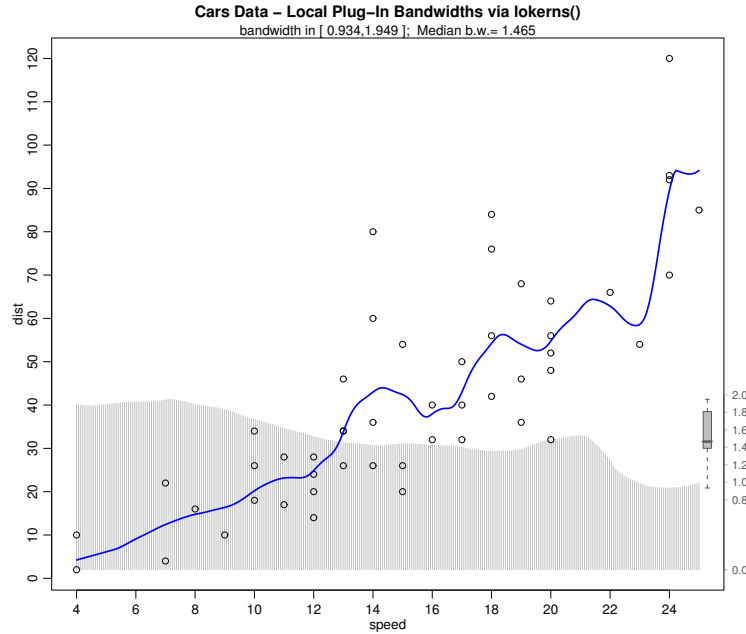


Figure 3.2: Nonparametric function estimate and locally varying bandwidths for distance of stopping as a function of speed of cars.

The hat matrix

It is useful to represent the kernel estimator evaluated at the design points $\hat{m}(x_1), \dots, \hat{m}(x_n)$ as a *linear operator* (on \mathbb{R}^n , i.e., a matrix):

$$\mathcal{S}: \quad \mathbb{R}^n \rightarrow \mathbb{R}^n, \\ (Y_1, \dots, Y_n)^\top \mapsto (\hat{m}(x_1), \dots, \hat{m}(x_n))^\top =: \hat{\mathbf{m}}(x.) = \hat{\mathbf{Y}},$$

i.e., $\hat{\mathbf{Y}} = \mathcal{S}\mathbf{Y}$ where \mathcal{S} is the matrix representing the linear operator above. The kernel estimator in (3.2) is of the form

$$\hat{m}(x) = \sum_{i=1}^n w_i(x) Y_i, \quad w_i(x) = \frac{K((x - x_i)/h)}{\sum_{j=1}^n K((x - x_j)/h)}.$$

Therefore, the matrix \mathcal{S} which represents the operator above is

$$[\mathcal{S}]_{r,s} = w_s(x_r), \quad r, s \in \{1, \dots, n\},$$

since $\mathcal{S}\{(Y_1, \dots, Y_n)^\top\} = (\hat{m}(x_1), \dots, \hat{m}(x_n))^\top$. The “smoother” matrix \mathcal{S} is also called the “*hat matrix*”, since it yields the vector of fitted values (at the observed design points x_i). Note that many other nonparametric regression methods (including those in the next two sections) can be seen to be linear in \mathbf{Y} and hence be written as $\hat{\mathbf{Y}} = \mathcal{S}\mathbf{Y}$ where \mathcal{S} depends on the x -design (x_1, x_2, \dots, x_n) and typically a smoothing parameter, say h . Algorithmically, for $\hat{\mathbf{Y}} = s(\mathbf{x}, \mathbf{Y}, h)$, the hat matrix can easily be computed columnwise as $\mathcal{S}_{\cdot,j} = s(\mathbf{x}, \mathbf{e}_j, h)$ where \mathbf{e}_j is the unit vector with $(\mathbf{e}_j)_i = \delta_{i,j} := \mathbb{1}\{i = j\}$.

Because of the elementary formula $\text{Cov}(\mathbf{A}\mathbf{X}) = \mathbf{A} \text{Cov}(\mathbf{X}) \mathbf{A}^\top$ (for a non-random matrix \mathbf{A} and random vector \mathbf{X}), we get the covariance matrix

$$\text{Cov}(\hat{\mathbf{m}}(x.)) = \sigma_\varepsilon^2 \mathcal{S} \mathcal{S}^\top, \quad (3.5)$$

i.e., $\text{Cov}(\hat{m}(x_i), \hat{m}(x_j)) = \sigma_\varepsilon^2 (\mathcal{S} \mathcal{S}^\top)_{ij}$, and $\text{Var}(\hat{m}(x_i)) = \sigma_\varepsilon^2 (\mathcal{S} \mathcal{S}^\top)_{ii}$.

Degrees of freedom

One way to assign degrees of freedom for regression estimators with a linear hat-operator \mathcal{S} is given by the trace,

$$df = \mathbf{tr}(\mathcal{S}). \quad (3.6)$$

This definition coincides with the notion we have seen in the linear model: there, (1.5), the fitted values $\hat{Y}_1, \dots, \hat{Y}_n$ can be represented by the projection $P = X(X^\top X)^{-1}X^\top$, which is the hat matrix, and $\mathbf{tr}(P) = \mathbf{tr}((X^\top X)^{-1}X^\top X) = \mathbf{tr}(I_p) = p$ equals the number of parameters in the model. Thus, the definition of degrees of freedom above can be viewed as a general concept for the number of parameters in a model fit with linear hat matrix.

Estimation of the error variance

Formula (3.5) requires knowledge of σ_ε^2 . A plausible estimate is via the residual sum of squares,

$$\hat{\sigma}_\varepsilon^2 = \frac{\sum_{i=1}^n (Y_i - \hat{m}(x_i))^2}{n - df}.$$

We then get an estimate for the standard error of the kernel regression estimator at the design points via (3.5):

$$\widehat{s.e.}(\hat{m}(x_i)) = \sqrt{\widehat{\text{Var}}(\hat{m}(x_i))} = \hat{\sigma}_\varepsilon \sqrt{(\mathcal{S}\mathcal{S}^\top)_{ii}}. \quad (3.7)$$

The estimated standard errors above are useful since under regularity conditions, $\hat{m}(x_i)$ is asymptotically normal distributed:

$$\hat{m}(x_i) \approx \mathcal{N}(\mathbb{E}[\hat{m}(x_i)], \text{Var}(\hat{m}(x_i))),$$

so that

$$I = \hat{m}(x_i) \pm 1.96 \cdot \widehat{s.e.}(\hat{m}(x_i))$$

yields approximate pointwise confidence intervals for $\mathbb{E}[\hat{m}(x_i)]$. Some functions in R (e.g. the function `gam` from package `mgcv`, see Chapter 7) supply such pointwise confidence intervals. Unfortunately, it is only a confidence interval for the expected value $\mathbb{E}[\hat{m}(x_i)]$ and not for the true underlying function $m(x_i)$. Correction of this interval is possible by subtracting a bias estimate: i.e., instead of the interval I above, we can use $I - \widehat{\text{bias}}$, where $\widehat{\text{bias}}$ is an estimate of the bias (which is not so easy to construct; see also section 2.3).

3.3 Local polynomial nonparametric regression estimator

As a starting point, consider the kernel estimator which can be represented as a locally constant function as in (3.3). This can now be extended to functions which are locally polynomial. We aim to find local regression parameters $\beta(x)$, defined as

$$\hat{\beta}(x) = \arg \min_{\beta \in \mathbb{R}^p} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right) (Y_i - \beta_1 - \beta_2(x_i - x) - \dots - \beta_p(x_i - x)^{p-1})^2.$$

An even number p turns out to be better: in practice, we often choose $p = 2$ or $p = 4$. The estimated local regression parameter $\hat{\beta}(x)$ describes a local polynomial regression fit, localized and centered at x . The function estimator is then given by evaluating this local

regression fit $\sum_{j=1}^p \hat{\beta}_j(x)(u-x)^{j-1}$ at $u = x$: due to the centering, only the local intercept remains and the local polynomial function estimator becomes

$$\hat{m}(x) = \hat{\beta}_1(x).$$

Note that due to (local) correlation among the $(x_i - x)^j$'s, $\hat{\beta}_1(x)$ is not the same as a local constant fit from (3.3).

The local polynomial estimator is often better at the edges than the locally constant Nadaraya-Watson kernel estimator. Another interesting property is that the method also immediately yields estimates for the derivatives of the function: when differentiating the local regression fit $\sum_{j=1}^p \hat{\beta}_j(x)(u-x)^{j-1}$ with respect to u and evaluating it at x , we obtain

$$\hat{m}^{(r)}(x) = r! \hat{\beta}_{r+1}(x) \quad (r = 0, 1, \dots, p-1).$$

3.4 Smoothing splines and penalized regression

Function estimation could also be done by using higher order global polynomials, which is often not advisable, or by using splines which can be specified by choosing a set of knots. The latter is a more locally oriented approach and is called “regression splines”. Here, we discuss a method based on splines *without* having to specify where to select the knots of the spline.

3.4.1 Penalized sum of squares

Consider the following problem: among all functions m with continuous second derivatives, find the one which minimizes the penalized residual sum of squares

$$\sum_{i=1}^n (Y_i - m(x_i))^2 + \lambda \int m''(z)^2 dz, \quad (3.8)$$

where $\lambda \geq 0$ is a smoothing parameter. The first term measures closeness to the data and the second term penalizes curvature (“roughness”) of the function. The two extreme cases are:

- $\lambda = 0$: m is any function interpolating the data (but for $\lambda \rightarrow 0$, in the limit, $m_\lambda \rightarrow$ the well defined interpolating natural cubic spline).
- $\lambda = \infty$: the least squares fit for linear regression which fulfills $m''(x) \equiv 0$.

Thus, a large λ corresponds to a smooth function.

3.4.2 The smoothing spline solution

Remarkably, the minimizer of (3.8) is *finite*-dimensional, although the criterion to be minimized is over a Sobolev space of functions (function space for which the integral $\int m''^2$ is defined), an infinite-dimensional space. Let us assume for now that the data has x values sorted and unique, $x_1 < x_2 < \dots < x_n$.

The solution $\hat{m}_\lambda(\cdot)$ (i.e., the unique minimizer of (3.8)) is a natural **cubic spline** with knots at the predictors x_i : that is, \hat{m} is a piecewise cubic polynomial in each interval $[x_i, x_{i+1})$ such that $\hat{m}_\lambda^{(k)}$ ($k = 0, 1, 2$) is continuous everywhere and (“natural”) $\hat{m}''(x_1) = \hat{m}''(x_n) = 0$. For the $n - 1$ cubic polynomials, we’d need $(n - 1) \cdot 4$ coefficients. Since there are $(n - 2) \cdot 3$ continuity conditions (at every “inner knot”, $i = 2, \dots, n - 1$) plus the

2 “natural” conditions, this leaves $4(n - 1) - [3(n - 2) + 2] = n$ free parameters (the β_j ’s below). Knowing that the solution is a cubic spline, it can be obtained by linear algebra. The trick is to represent

$$m_\lambda(x) = \sum_{j=1}^n \beta_j B_j(x), \quad (3.9)$$

where the $B_j(\cdot)$ ’s are basis functions for natural splines. The unknown coefficients can then be estimated from least squares in linear regression under side constraints. The criterion in (3.8) for \hat{m}_λ as in (3.9) then becomes

$$\|\mathbf{Y} - B\boldsymbol{\beta}\|^2 + \lambda\boldsymbol{\beta}^\top\Omega\boldsymbol{\beta},$$

where the design matrix B has j th column $(B_j(x_1), \dots, B_j(x_n))^\top$ and $\Omega_{jk} = \int B_j''(z)B_k''(z)dz$. The solution can then be derived in a straightforward way,

$$\hat{\boldsymbol{\beta}}_{n \times 1} = (B^\top B + \lambda\Omega)^{-1}B^\top \mathbf{Y}. \quad (3.10)$$

This can be computed efficiently using fast linear algebra, particularly when B is a banded matrix.

The fitted values are then $\hat{Y}_i = \hat{m}_\lambda(x_i)$ ($i = 1, \dots, n$), $\hat{m}_\lambda(x) = \sum_{j=1}^n \hat{\beta}_j B_j(x)$, and

$$\hat{\mathbf{Y}} = (\hat{Y}_1, \dots, \hat{Y}_n)^\top = \mathcal{S}_\lambda \mathbf{Y}, \quad \mathcal{S}_\lambda = B(B^\top B + \lambda\Omega)^{-1}B^\top. \quad (3.11)$$

The hat matrix $\mathcal{S}_\lambda = \mathcal{S}_\lambda^\top$ is here symmetric which implies elegant mathematical properties (real-valued eigen-decomposition).

3.4.3 Shrinking towards zero

At first sight, the smoothing spline solution in (3.9) looks heavily over-parameterized since we have to fit n unknown coefficients β_1, \dots, β_n . However, the solution in (3.10) is not the least squares estimator but rather a Ridge-type version: the matrix $\lambda\Omega$ serves as a Ridge or shrinkage matrix so that the estimates $\hat{\boldsymbol{\beta}}$ are shrunken towards zero: i.e., for large λ , the expression $(B^\top B + \lambda\Omega)^{-1}$ becomes small. Thus, since all the coefficients are shrunken towards zero, we gain on the variance part of each $\hat{\beta}_j$ by the square of the shrinkage factor, and the overall smoothing spline fit will be appropriate if λ is chosen suitably.

Note that λ can be chosen on the scale of equivalent degrees of freedom (df): $df = \text{tr}(\mathcal{S}_\lambda)$. This provides an intuitive way to specify a smoothing spline: e.g. a smoothing spline with $df=5$ is as complex as a global polynomial of degree 4 (which has 5 parameters including the intercept), see also Figure 3.1.

3.4.4 Relation to equivalent kernels

It is interesting to note that there is a relationship between the smoothing spline estimate and a particular kernel estimator. The smoothing spline estimate $\hat{m}_\lambda(x)$ is approximately

$$\begin{aligned} \hat{m}_\lambda(x) &\approx \sum_{i=1}^n w_i(x) Y_i, \\ w_i(x) &= \frac{1}{nh(x)f_X(x)} K\left(\frac{x - x_i}{h(x)}\right), \\ h(x) &= \lambda^{1/4} n^{-1/4} f_X(x)^{-1/4}, \\ K(u) &= \frac{1}{2} \exp\left(-\frac{|u|}{\sqrt{2}}\right) \sin\left(\frac{|u|}{\sqrt{2}} + \frac{\pi}{4}\right). \end{aligned}$$

See for example Green and Silverman (1994, Ch. 3.7).

The important fact is here that the bandwidth of the equivalent kernel estimator has a *local bandwidth*, depending on the density of the predictor variable x . In regions where the density of the predictor is low (observations are sparse), the bandwidth automatically adapts and becomes large: intuitively, this is the right behavior because we should use strong smoothing in regions where only few observations are available.

An example of a smoothing spline fit for real data is displayed in Figure 3.1. Finally, we illustrate on an artificial dataset the advantage of smoothing splines to adapt to the density of the predictor variables. Figure 3.3 shows the performance of smoothing splines in comparison with the Nadaraya-Watson Gaussian kernel estimator. The data has the following structure:

- the density of the predictor is high for positive values and low for negative values
- the true function is strongly oscillating where the predictor density is high and slowly oscillating where the predictors are sparse

The smoothing spline fit (using the GCV criterion for selecting the degrees of freedom, see section 4.5) yields a very good fit: it captures the strong oscillations because there are many data points with positive values of the predictors. On the other hand, the kernel estimator has been tuned such that it captures the strong oscillations, using a small bandwidth h (this was done by knowing the true underlying function – which is not feasible in practice): but the small bandwidth h then causes a much too rough and poor estimate for negative predictor values, although the underlying true function is smooth.

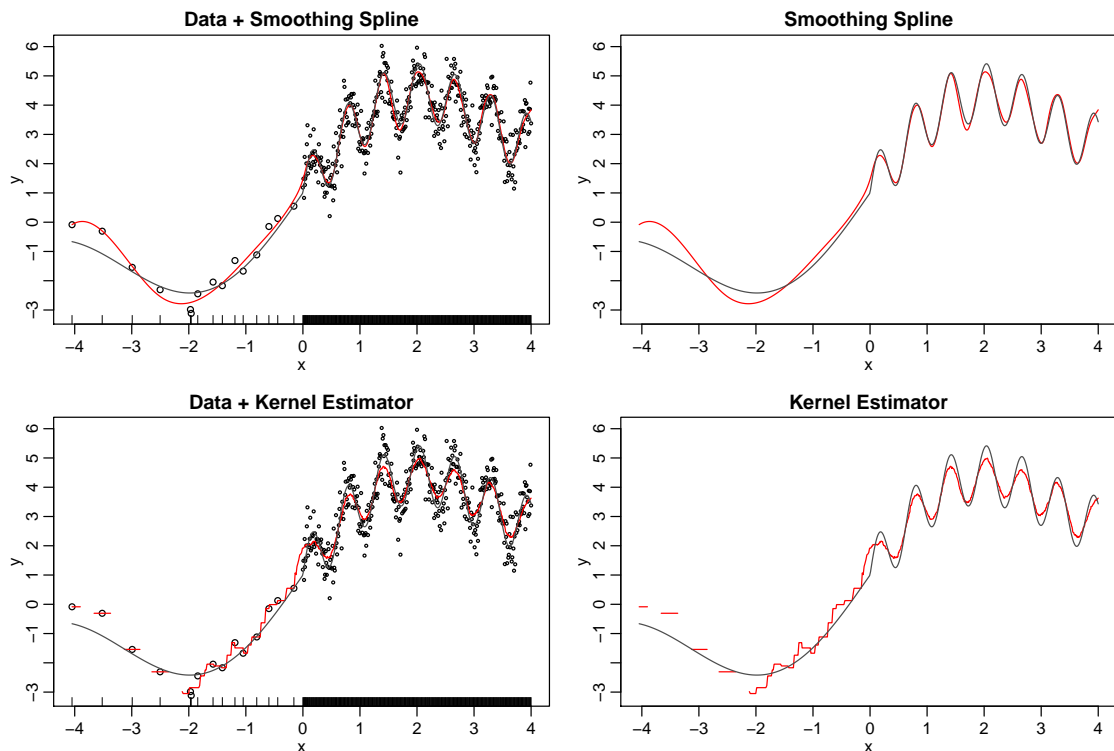


Figure 3.3: Curve estimation for synthetic dataset ($n = 417$). Left panels: scatterplot, overlaid with curve estimates (red) and true curve (gray); Right panels: curve estimates (red) and true curve (gray).

Top: Smoothing spline with GCV-selected df ($= 9.23$); Bottom: Nadaraya-Watson kernel estimator with bandwidth chosen for good estimation of the strong oscillations of the true function (giving $df = 25.6$).

Chapter 4

Cross-Validation

4.1 Introduction

The generalization performance (which is a terminology from the machine learning community) of a learning method describes its prediction capacity on new test data. Or in other words, the generalization performance measures the predictive power of a learning method on new, out-sample data. Having an estimate of this predictive power is very important for comparing among different methods or for tuning an estimator or algorithm to optimize predictive power.

Cross-Validation is one very general method for estimating such generalization or out-sample performance.

4.2 Training and Test Set

Consider the situation where we have data

$$(X_1, Y_1), \dots, (X_n, Y_n) \text{ i.i.d. } \sim P$$

where P denotes the unknown distribution. (In case where the x_i is non-random, we can also drop the i.i.d. assumption for the pairs (x_i, Y_i) .)

We typically have a target in mind, for example the regression function $m(\cdot)$. The estimated regression function $\hat{m}(\cdot)$ is then constructed by using some estimator or algorithm, based on the data $(X_1, Y_1), \dots, (X_n, Y_n)$. This data is also called the **training data**, since it is used to train the estimator or learning algorithm.

We then would like to evaluate the accuracy of the estimated target which is based on the training data. A principal problem thereby is that if we use the training data again to measure the predictive power of our estimated target (e.g. of $\hat{m}(\cdot)$), the results will be overly optimistic. For example, when using the residual sum of squares in regression,

$$\sum_{i=1}^n (Y_i - \hat{m}(X_i))^2$$

becomes smaller the more “complex” (more degrees of freedom) the model for $\hat{m}(\cdot)$ involves. Obviously, a very complex model will not be good. We have seen already in Section 3.4 that penalizing the residual sum of squares can be useful to cope with the overfitting problem.

Alternatively, we could look how well the estimated target, e.g. $\hat{m}(\cdot)$, does on **new test data**

$$(X_{new,1}, Y_{new,1}), \dots, (X_{new,\ell}, Y_{new,\ell}) \text{ i.i.d. } \sim P,$$

which is assumed to be independent from the training data but having the same distribution P . In case of regression, we would like to evaluate

$$\ell^{-1} \sum_{i=1}^{\ell} (Y_{new,i} - \hat{m}(X_{new,i}))^2.$$

More generally, when having an estimated target \hat{m} and a loss function ρ , we would like to evaluate

$$\ell^{-1} \sum_{i=1}^{\ell} \rho(Y_{new,i}, \hat{m}(X_{new,i})),$$

where $\hat{m}(\cdot)$ is constructed from training data only. If ℓ is large, this evaluation on the test set approximates the theoretical test set error

$$\mathbb{E}_{(X_{new}, Y_{new})} [\rho(Y_{new}, \underbrace{\hat{m}}_{\substack{\text{from training} \\ \text{data only}}}(X_{new}))]$$

which is still a function of the training data. The expected value of this (with respect to the training data) is the generalization error,

$$\mathbb{E}_{\text{training}} \mathbb{E}_{(X_{new}, Y_{new})} [\rho(Y_{new}, \underbrace{\hat{m}}_{\substack{\text{training data}}}(X_{new}))] = \mathbb{E}[\rho(Y_{new}, \hat{m}(X_{new}))], \quad (4.1)$$

where the latter \mathbb{E} is over the training data $(X_1, Y_1), \dots, (X_n, Y_n)$ as well as the test data (X_{new}, Y_{new}) . This generalization error avoids the fact that it would get smaller the more complex the model for $\hat{m}(\cdot)$. In particular, it will increase as \hat{m} is overfitting. Hence the generalization error in (4.1) is a very useful quantity to regularize and tune an estimator or algorithm for \hat{m} .

4.3 Constructing training-, test-data and cross-validation

In reality, we just have the data, no “future observations”, when we want to run and tune our estimator or algorithm. But we can always “artificially” construct (smaller) training- and test-data by splitting our sample.

4.3.1 “Validation Set” or “Sample Splitting Once” Approach

The simple most approach (randomly) selects a part (say half or a quarter) of the sample as “validation set” (or “hold-out” set) and uses the remaining data for “training”, i.e., fitting. Put differently, the sample is split *only once* into training and testing.

This approach has been chosen frequently in machine learning contexts, possibly because it is computationally the fastest. Its drawbacks are its arbitrariness and the high variance and bias as an estimate of the true generalization error.

The following cross-validation versions are all more stable than this single sample splitting.

4.3.2 Leave-one-out cross-validation

For leave-one-out cross-validation (l.o.o. CV), we use the i th sample point as test data (test sample size = 1) and the remaining $n - 1$ sample points as training data.

Denote in general the estimator or algorithm by $\hat{\theta}_n$ which is based on the n sample points. In CV, when deleting the i th sample, the estimator is based on the sample without the i th observation, and we denote this by

$$\hat{\theta}_{n-1}^{(-i)}, \quad i = 1, \dots, n.$$

We can then evaluate this estimate on the i th observation (the test sample), for every $i = 1, \dots, n$. To make this more explicit, we suppose that the estimator $\hat{\theta}$ is a curve estimator \hat{m} , and performance is evaluated in terms of a loss function ρ , e.g. $\rho(u, v) = (u - v)^2$ as above¹ and in Chapter 3. The cross-validated performance is then

$$n^{-1} \sum_{i=1}^n \rho \left(Y_i, \hat{m}_{n-1}^{(-i)}(X_i) \right) \quad (4.2)$$

which is an estimate of the test set error, or generalization error, in (4.1).

Note that the CV-method is very general: it can be used for any loss function ρ and in many problems which can be different than linear or nonparametric regression.

L.o.o. CV in general requires that the estimator $\hat{\theta}$ is fitted n times, namely for all training sets where the i th observation has been deleted ($i = 1, \dots, n$).

4.3.3 K -fold Cross-Validation

A computationally cheaper generalization of leave-one-out CV is the so-called K -fold CV (for $K \ll n$). The data set is randomly partitioned into K equally sized (as equal as possible) subsets \mathcal{B}_k of $\{1, \dots, n\}$ such that $\cup_{k=1}^K \mathcal{B}_k = \{1, \dots, n\}$ and $\mathcal{B}_j \cap \mathcal{B}_k = \emptyset$ ($j \neq k$). We can now set aside a k th test data set including all sample points whose indices are elements of \mathcal{B}_k .

K -fold cross-validation then uses the sample points with indices not in \mathcal{B}_k as training set to construct an estimator

$$\hat{\theta}_{n-|\mathcal{B}_k|}^{(-\mathcal{B}_k)}.$$

The analogue of the evaluation formula in (4.2) is then, for regression with $\hat{m}(\cdot)$,

$$K^{-1} \sum_{k=1}^K |\mathcal{B}_k|^{-1} \sum_{i \in \mathcal{B}_k} \rho \left(Y_i, \hat{m}_{n-|\mathcal{B}_k|}^{(-\mathcal{B}_k)}(X_i) \right).$$

K -fold CV thus only needs to run the estimation algorithm K times. Leave-one-out CV is the same as n -fold CV. In practice, often $K = 5$ or $K = 10$ are used.

4.3.4 Many random divisions into test- and training-data

K -fold CV has the disadvantage that it depends on the **one** realized random partition into subsets $\mathcal{B}_1, \dots, \mathcal{B}_K$. In particular, if the data (pairs) are assumed to be i.i.d., the indexing of the data (pairs) should not have an influence on validating a performance: note that leave-one-out CV is indeed independent of indexing the data.

In principle, we can generalize leave-one-out CV to leave- d -out CV. Leave a set \mathcal{C} comprising d observations out (set them aside as test data) and use the remaining $n - d$ data points for training the statistical model or algorithm:

$$\hat{\theta}_{n-d}^{(-\mathcal{C})}, \quad \text{for all possible subsets } \mathcal{C}_k, \quad k = 1, 2, \dots, \binom{n}{d}.$$

¹e.g., the L_1 penalty $\rho(u, v) = |u - v|$ is much more robust against outliers

We can then evaluate this estimate on observations from the test set \mathcal{C}_i (the test sample), for every i . The analogue of (4.2), for regression with $\hat{m}(\cdot)$, is then

$$\binom{n}{d}^{-1} \sum_{k=1}^{\binom{n}{d}} d^{-1} \sum_{i \in \mathcal{C}_k} \rho \left(Y_i, \hat{m}_{n-d}^{(-\mathcal{C}_k)}(X_i) \right) \quad (4.3)$$

which is also an estimate of the test set error, or generalization error in (4.1).

The computational burden becomes immense if $d \geq 3$. A computational short-cut is then given by randomization: instead of considering *all* subsets (test sets), we draw B **random** test subsets

$$\mathcal{C}_1^*, \dots, \mathcal{C}_B^* \text{ i.i.d. } \sim \text{Uniform}(\{1, \dots, \binom{n}{d}\}),$$

where the Uniform distribution assigns probability $\binom{n}{d}^{-1}$ to every possible subset of size d . Such a Uniform distribution, or such a random subset \mathcal{C}^* , is easily constructed by **sampling without replacement**:

Draw d times randomly without replacement from $\{1, \dots, n\}$, yielding a subset \mathcal{C}^* .

The random approximation for (4.3) is then

$$B^{-1} \sum_{k=1}^B d^{-1} \sum_{i \in \mathcal{C}_k^*} \rho \left(Y_i, \hat{m}_{n-d}^{(-\mathcal{C}_k^*)}(X_i) \right). \quad (4.4)$$

For $B = \infty$, the two expressions in (4.4) and (4.3) coincide (note that we only would need a finite, maybe huge, amount of computation for evaluation of (4.3)).

In practice, we typically choose $d = \lceil \gamma n \rceil$ with $\gamma \approx 0.1$ (10% test data). For the number of random test and training sets, we choose $B \approx 50 - 500$, depending on the cost to compute $\hat{\theta}_{n-d}^{(-\mathcal{C})}$ for a training set of size $n - d$ (evaluation on the test set \mathcal{C} is usually fast). Thus, in terms of computation, the stochastic version in (4.4) may be even faster than leave-one-out CV in (4.2) if $B < n$, and we can use the stochastic approximation also for leave-one-out CV when sample size n is large.

4.4 Properties of different CV-schemes

4.4.1 One random split into test- and training-data

This depends very strongly on the **one** split and additionally, the *proportion* training vs testing is arbitrary. As an estimate of the generalization error this is poor *both* in bias and variance.

OTOH, this is *fastest* and in clear cut cases its score may give the minimum, i.e., optimum, for approximately correct values of tuning parameters (e.g., the smoothing parameters h or λ in the case of nonparametric regression).

4.4.2 Leave-one-out CV

Leave-one-out CV, which is equal to n -fold CV, is approximately unbiased for the true prediction or generalization error: the only drawback in terms of bias is that we use training sample size $n - 1$ instead of the original n , causing for a slight bias. The variance of leave-one-out CV is typically high, because the n training sets are so similar to each other:

$$\text{Var} \left(n^{-1} \sum_{i=1}^n \rho \left(Y_i, \hat{m}_{n-1}^{(-i)}(X_i) \right) \right) = n^{-2} \sum_{i=1}^n \sum_{j=1}^n \text{Cov} \left(\rho(Y_i, \hat{m}_{n-1}^{(-i)}(X_i)), \rho(Y_j, \hat{m}_{n-1}^{(-j)}(X_j)) \right).$$

Although (X_i, Y_i) is typically assumed to be independent from (X_j, Y_j) , the covariances are substantial because of strong correlation of $\hat{m}_{n-1}^{(-i)}(\cdot)$ and $\hat{m}_{n-1}^{(-j)}(\cdot)$, and hence the double sum in the formula above can be quite large.

4.4.3 Leave- d -out CV

Heuristically, it is clear that leave- d -out CV, with $d > 1$, has higher bias than leave-one-out CV; because we use training samples of sizes $n - d$ instead of the original n , causing some bias. In terms of variance, we average over more, although highly correlated summands in (4.3), which can be shown to decrease variance in comparison to leave-one-out CV.

4.4.4 K -fold CV; stochastic approximations ($B \gg 1$ splits)

K -fold CV has larger bias than leave-one-out CV; because the training sets are of smaller sample size than $n - 1$ (in leave-one-out CV), and also smaller than the original sample size n . In terms of variance, it is not clear (sometimes wrongly stated in the literature) whether K -fold CV has smaller variance than leave-one-out CV.

The stochastic approximation is expected to have higher variance than the computationally infeasible leave- d -out CV.

4.5 Computational shortcut for some linear fitting operators

Consider the special case of fitting a cubic smoothing spline or fitting a least squares parametric estimator: in both cases, we have a linear fitting operator \mathcal{S} ,

$$(\hat{m}(x_1), \dots, \hat{m}(x_n))^\top = \mathcal{S}\mathbf{Y}, \quad \mathbf{Y} = (Y_1, \dots, Y_n)^\top.$$

When focusing on the squared error loss function $\rho(y, x) = |y - x|^2$, there is a surprising result for representing the leave-one-out CV score in (4.2):

$$n^{-1} \sum_{i=1}^n \left(Y_i - \hat{m}_{n-1}^{(-i)}(X_i) \right)^2 = n^{-1} \sum_{i=1}^n \left(\frac{Y_i - \hat{m}(X_i)}{1 - \mathcal{S}_{ii}} \right)^2. \quad (4.5)$$

The interesting property is that we can compute the CV score by fitting the original estimator $\hat{m}(\cdot)$ **once on the full dataset**, without having to do it n times by holding one observation back as a test point. Moreover, by using efficient linear algebra implementations, the elements \mathcal{S}_{ii} can be computed with $O(n)$ operations.

Historically, it was computationally more difficult to obtain all diagonal elements from the hat matrix \mathcal{S}_{ii} ($i = 1, \dots, n$); and computing the trace, $\mathbf{tr}(\mathcal{S})$, which equals the sum of the eigenvalues of \mathcal{S} , has been easier. The generalized cross-validation was then proposed in the late 70's:

$$\text{GCV} = \frac{n^{-1} \sum_{i=1}^n (Y_i - \hat{m}(X_i))^2}{(1 - n^{-1} \mathbf{tr}(\mathcal{S}))^2}.$$

This again requires to compute $\hat{m}(\cdot)$ only once on the full dataset. Moreover, if all the diagonal elements \mathcal{S}_{ii} would be equal (which is typically not the case), GCV would coincide with the formula in (4.5). As outlined already, GCV has been motivated by computational considerations which are nowadays not very relevant anymore. However, the statistical properties of GCV can also be as good (and sometimes better or worse) than the ones from leave-one-out CV.

Chapter 5

Bootstrap

5.1 Introduction

The bootstrap, proposed by Efron (1979), is by now considered as a breakthrough in statistics. Essentially, the bootstrap can be described as “simulating from an estimated model”. This turns out to be tremendously useful for making statistical inference (confidence intervals and testing) and, analogous to the goals in cross-validation, for estimating the predictive power of a model or algorithm and hence also for tuning of statistical procedures.

5.2 Efron’s nonparametric bootstrap

Consider the situation where the data are realizations of

$$Z_1, \dots, Z_n \text{ i.i.d. } \sim P,$$

where P denotes an unknown distribution. The variables Z_i can be real-valued (usually then denoted by X_i), or they can be vector-valued. For example, $Z_i = (X_i, Y_i)$ are the pairs in a regression problem with $X_i \in \mathbb{R}^p$ and $Y_i \in \mathbb{R}$.

We denote a statistical procedure or estimator by

$$\hat{\theta}_n = g(Z_1, \dots, Z_n) \tag{5.1}$$

which is a (known) function g of the data Z_1, \dots, Z_n . The estimator $\hat{\theta}_n$ can be a parameter estimator or also a curve estimator (e.g. in nonparametric regression).

Whenever we want to make statistical inference, we would like to know the probability distribution of $\hat{\theta}_n$. For example, constructing a confidence interval for a true parameter θ requires the knowledge of the distribution of $\hat{\theta}_n$; or constructing a statistical test requires the distribution of $\hat{\theta}_n$ under the null-hypothesis. We also considered in chapter 4 the problem of estimating the generalization error in regression

$$\mathbb{E}[(Y_{new} - \hat{m}(X_{new}))^2]$$

which can be thought of as the expected value, a summary statistic of the distribution, of

$$\hat{\theta}_{n+1} = g(Z_1, \dots, Z_n, Z_{new}) = (Y_{new} - \hat{m}_{Z_1, \dots, Z_n}(X_{new}))^2, \quad Z_i = (X_i, Y_i),$$

where we have a function g of the training and test data.

Deriving the exact distribution of $\hat{\theta}_n$ is typically impossible, unless the function g is simple and P is a mathematically convenient distribution, e.g. P is a Normal distribution.

If exact distributions are not available, much mathematical theory has been developed to get at least the asymptotic distribution of $\hat{\theta}_n$ as n gets large. For example, due to the central limit theorem,

$$\hat{\theta}_n = \frac{1}{n} \sum_{i=1}^n X_i \approx \mathcal{N}(\mu, \sigma^2/n), \quad X_i \in \mathbb{R},$$

where $\mu = \mathbb{E}[X_i]$, $\sigma^2 = \text{Var}(X_i)$. We then only need to estimate the parameters μ and σ in order to have an approximate distribution for $\frac{1}{n} \sum_{i=1}^n X_i$. For the maximum-likelihood estimator in general, estimation of the asymptotic variance is already more subtle. Or for the sample median,

$$\begin{aligned} \hat{\theta}_n &= \text{median}(X_1, \dots, X_n) \approx \mathcal{N}(\theta, \sigma_{asy}^2/n), \quad X_i \in \mathbb{R}, \\ \sigma_{asy}^2 &= \mathbb{E}[(X - \theta)^2] / (4f^2(\theta)) \end{aligned}$$

where the asymptotic variance already involves quantities like the density f of P at the unknown parameter $\theta = \text{median}(P)$. Estimating this asymptotic variance is already a pretty awkward task, and we should keep in mind that we would then only get the asymptotic answer to the problem of getting the distribution of $\hat{\theta}_n$. Finally, for more complex algorithms, mathematical theory is lacking for obtaining the approximate, asymptotic distribution.

A pioneering step has then been taken by Efron (1979). Suppose we knew what the distribution P is: we could then **simulate** to obtain the distribution of any $\hat{\theta}_n$ with arbitrary accuracy (when simulating enough). Because we do not know the distribution P of the data generating mechanism, we use the **empirical distribution** \hat{P}_n which places probability mass $1/n$ on every data point Z_i , $i = 1, \dots, n$. The recipe is then to simulate from \hat{P}_n : generate simulated data

$$Z_1^*, \dots, Z_n^* \text{ i.i.d. } \sim \hat{P}_n.$$

Such a simulated new data set is called a bootstrap sample. We can now compute our estimator $\hat{\theta}_n^* = g(Z_1^*, \dots, Z_n^*)$, analogously to (5.1) but based on the bootstrap sample, and we then repeat this many times to get an approximate distribution, e.g. via the histogram of many simulated $\hat{\theta}_n^*$'s.

5.2.1 The bootstrap algorithm

Bootstrapping an estimator as in (5.1) can be done as follows.

1. Generate a bootstrap sample

$$Z_1^*, \dots, Z_n^* \text{ i.i.d. } \sim \hat{P}_n.$$

This can be realized as follows. Do n **uniform random drawings with replacement** from the data set $\{Z_1, \dots, Z_n\}$, yielding the bootstrap sample.

2. Compute the bootstrapped estimator

$$\hat{\theta}_n^* = g(Z_1^*, \dots, Z_n^*),$$

based on the bootstrap sample; the function $g(\cdot)$ is as in (5.1).

3. Repeat steps 1 and 2 B times to obtain

$$\hat{\theta}_n^{*1}, \dots, \hat{\theta}_n^{*B}.$$

4. These B bootstrapped estimators in 3 can then be used as approximations for the bootstrap expectation, the bootstrap variance and the bootstrap quantiles:

$$\begin{aligned}\mathbb{E}^*[\hat{\theta}_n^*] &\approx \frac{1}{B} \sum_{i=1}^B \hat{\theta}_n^{*i}, \\ \text{Var}^*(\hat{\theta}_n^*) &\approx \frac{1}{B-1} \sum_{i=1}^B \left(\hat{\theta}_n^{*i} - \frac{1}{B} \sum_{j=1}^B \hat{\theta}_n^{*j} \right)^2, \\ \alpha\text{-quantile of distribution of } \hat{\theta}_n^* &\approx \text{empirical } \alpha\text{-quantile of } \hat{\theta}_n^{*1}, \dots, \hat{\theta}_n^{*B}.\end{aligned}$$

The definition of the bootstrap values \mathbb{E}^* , Var^* or the bootstrap distribution are discussed next.

5.2.2 The bootstrap distribution

The bootstrap distribution, denoted here by P^* , is the conditional probability distribution which is induced by i.i.d. resampling of the data

$$Z_1^*, \dots, Z_n^* \text{ i.i.d. } \sim \hat{P}_n, \quad (5.2)$$

given the original data. The fact that we condition on the data allows to treat the bootstrap resampling distribution \hat{P}_n , which is the empirical distribution of the data, as a fixed distribution.

Therefore, the bootstrap distribution P^* of $\hat{\theta}_n^* = g(Z_1^*, \dots, Z_n^*)$ is the distribution which arises when resampling with \hat{P}_n in (5.2) and applying the function g on such a bootstrap sample, exactly as done by simulation in section 5.2.1. (From a theoretical point of view, the bootstrap distribution P^* can be represented by a multinomial distribution that contains the information which and how many times of the original data appears again in the bootstrap sample (5.2)).

The bootstrap expectation of $\hat{\theta}_n^*$ is then denoted by $\mathbb{E}^*[\hat{\theta}_n^*]$ which is a conditional expectation given the data. Likewise, $\text{Var}^*(\hat{\theta}_n^*)$ is a conditional variance given the data. Since \hat{P}_n in (5.2) is “close” to the true data-generating probability P , the bootstrap values are “reasonable” estimates for the true quantities. For example, we can use

$$\widehat{\text{Var}}(\hat{\theta}_n) = \text{Var}^*(\hat{\theta}_n^*).$$

This estimate is approximately computed as in step 4 of the bootstrap algorithm in section 5.2.1. Whether such a bootstrap estimate is consistent will be discussed in the following section.

5.2.3 Bootstrap consistence

The bootstrap is called to be *consistent for* $\hat{\theta}_n$ if, for an increasing sequence a_n , for all x ,

$$\mathbb{P}[a_n(\hat{\theta}_n - \theta) \leq x] - \mathbb{P}^*[a_n(\hat{\theta}_n^* - \hat{\theta}_n) \leq x] \xrightarrow{P} 0 \quad (n \rightarrow \infty). \quad (5.3)$$

In classical situations, $a_n = \sqrt{n}$: for example, the maximum-likelihood estimator $\hat{\theta}_n$ satisfies under regularity assumptions

$$\sqrt{n}(\hat{\theta}_{n,\text{MLE}} - \theta) \xrightarrow{D} \mathcal{N}(0, I^{-1}(\theta)) \quad (n \rightarrow \infty),$$

where $I(\theta)$ denotes the Fisher information at θ . Bootstrap consistency (5.3) then means

$$\sqrt{n}(\hat{\theta}_{n,\text{MLE}}^* - \hat{\theta}_n) \xrightarrow{D^*} \mathcal{N}(0, I^{-1}(\theta)) \text{ in probability } (n \rightarrow \infty).$$

Consistency of the bootstrap typically holds if the limiting distribution of $\hat{\theta}_n$ is Normal, and if the data Z_1, \dots, Z_n are i.i.d.

Consistency of the bootstrap (usually) implies consistent variance and bias estimation:

$$\frac{\text{Var}^*(\hat{\theta}_n^*)}{\text{Var}(\hat{\theta}_n)} \xrightarrow{P} 1,$$

$$\frac{\mathbb{E}^*[\hat{\theta}_n^*] - \hat{\theta}_n}{\mathbb{E}[\hat{\theta}_n] - \theta} \xrightarrow{P} 1.$$

5.2.4 Bootstrap confidence interval: a first approach

Given bootstrap consistence, consistent confidence intervals can be constructed. A two-sided confidence interval with coverage $1 - \alpha$ for a parameter θ is given by

$$[\hat{\theta}_n - q_{1-\alpha/2}, \hat{\theta}_n - q_{\alpha/2}], \quad \text{where } q_\alpha = \alpha\text{-quantile of } \hat{\theta}_n - \theta.$$

This is derived using elementary calculations. In analogy, the bootstrap estimated confidence interval is then defined as

$$[\hat{\theta}_n - \hat{q}_{1-\alpha/2}, \hat{\theta}_n - \hat{q}_{\alpha/2}], \tag{5.4}$$

where $\hat{q}_\alpha = \alpha$ -bootstrap quantile of $\hat{\theta}_n^* - \hat{\theta}_n$.

Due to invariance of the quantile:

$$\hat{q}_\alpha = q_\alpha^* - \hat{\theta}_n, \quad \text{where } q_\alpha^* = \alpha\text{-bootstrap quantile of } \hat{\theta}_n^*.$$

Therefore, the bootstrap confidence interval in (5.4) becomes

$$[2\hat{\theta}_n - q_{1-\alpha/2}^*, 2\hat{\theta}_n - q_{\alpha/2}^*]. \tag{5.5}$$

Note that this is **not the same as simply taking the quantiles of the bootstrap distribution**, i.e., the simple-minded $[q_{\alpha/2}^*, q_{1-\alpha/2}^*]$ is “backwards” and often less appropriate. The derivation which we gave for this “unintuitive” fact hinges on the consistency of the bootstrap in (5.3).

Better bootstrap confidence intervals than (5.4) exist and often have better coverage accuracy — at the price of being somewhat more difficult to implement. See also the double bootstrap below (section 5.3)

An example

We consider here the example about asphalt quality from section 1.3.5, where a linear model has been used to model the log of rutting as a function of 6 predictor variables.

The bootstrap distributions for the 6 coefficients corresponding to the 6 predictor variables are exhibited in Figure 5.1.

The standard confidence intervals based on normally distributed data (or on asymptotic considerations) are (including an intercept)

$$\hat{\beta}_j \pm t_{n-p-1;1-\alpha/2} \cdot \widehat{s.e.}(\hat{\beta}_j),$$

where $t_{n-p-1;\alpha}$ denotes the α -quantile of a t_{n-p-1} distribution ($\text{qt}(\alpha, n-p-1)$). In this example, $n = 31$, $p = 6$ when using $\alpha = 0.05$, we get $t_{n-7;1-\alpha/2} = 2.063899$. The confidence intervals are then given in Table 5.1.

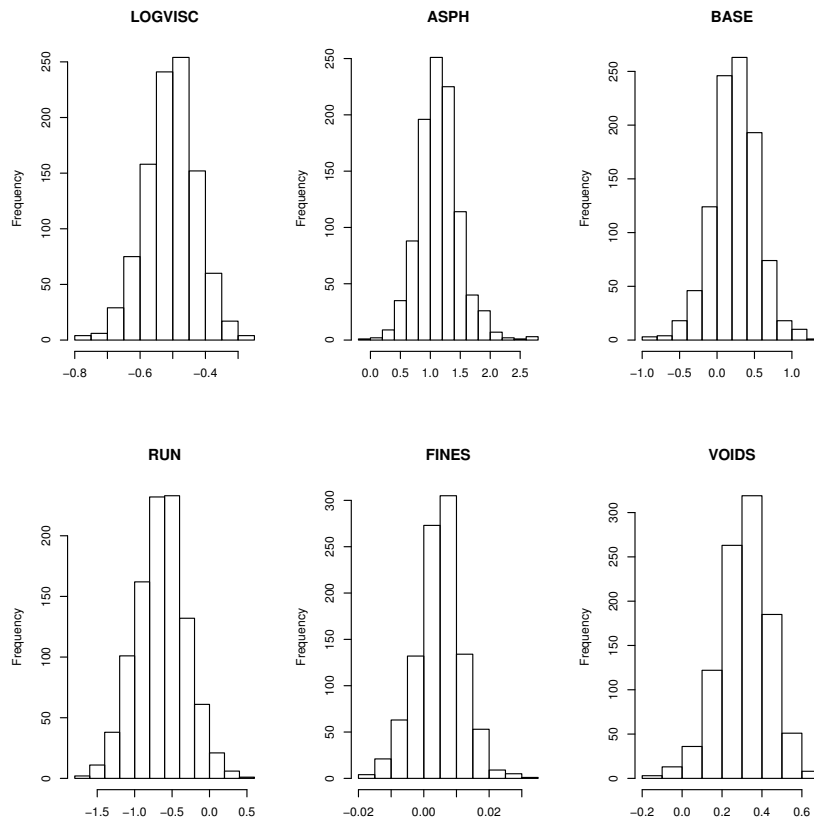


Figure 5.1: Histograms of $B = 1000$ bootstrap values $\hat{\beta}_j^*$ for the 6 coefficients in the linear model about asphalt quality from section 1.3.5.

method	intercept	LOGVISC	ASPH	BASE	RUN	FINES	VOIDS
classical	[-10.86,-0.71]	[-0.66,-0.36]	[0.60,1.70]	[-0.44,0.91]	[-1.23,-0.01]	[-0.02,0.02]	[0.09,0.54]
bootstrap	[-11.97,0.99]	[-0.67,-0.36]	[0.41,1.78]	[-0.37,0.87]	[-1.26,0.09]	[-0.01,0.02]	[0.10,0.60]

Table 5.1: Classical 95% confidence intervals and 95% bootstrap confidence intervals (using $B = 1000$) from (5.4) for the 7 coefficients in a linear model for the asphalt dataset from section 1.3.5. In bold are the bootstrap confidence intervals which suggest non-significance while the classical confidence intervals (or t -tests) would suggest significance.

5.2.5 Bootstrap estimate of the generalization error

Consider the problem of estimating the generalization error

$$\mathbb{E}[\rho(Y_{new}, \hat{m}(X_{new}))]$$

in (4.1), where ρ is a loss function such as $\rho(y, m) = |y - m|^2$ and $\hat{m}(\cdot)$ could be a regression estimator.

The bootstrap approach to estimate the generalization error is from a conceptual view as follows.

1. Generate $(X_1^*, Y_1^*), \dots, (X_n^*, Y_n^*), (X_{new}^*, Y_{new}^*)$ i.i.d. $\sim \hat{P}_n$,
where \hat{P}_n is the empirical distribution of the original data $(X_1, Y_1), \dots, (X_n, Y_n)$.
2. Compute the bootstrapped estimator $\hat{m}^*(\cdot)$ based on $(X_1^*, Y_1^*), \dots, (X_n^*, Y_n^*)$.

3. Compute the bootstrap generalization error

$$\mathbb{E}^*[\rho(Y_{new}^*, \hat{m}^*(X_{new}^*))],$$

where \mathbb{E}^* is with respect to all the bootstrap variables $\text{train}^* = (X_1^*, Y_1^*), \dots, (X_n^*, Y_n^*)$ **and** $\text{test}^* = (X_{new}^*, Y_{new}^*)$. Use this as an estimate of the true generalization error.

The bootstrap generalization error can be re-written as follows:

$$\begin{aligned} \mathbb{E}^*[\rho(Y_{new}^*, \hat{m}^*(X_{new}^*))] &= \mathbb{E}_{\text{train}^*}^* [\mathbb{E}_{\text{test}^*}^*[\rho(Y_{new}^*, \hat{m}^*(X_{new}^*)) \mid \text{train}^*]] \\ &= \mathbb{E}_{\text{train}^*}^* \left[\frac{1}{n} \sum_{i=1}^n \rho(Y_i, \hat{m}^*(X_i)) \right] = \frac{1}{n} \sum_{i=1}^n \mathbb{E}^*[\rho(Y_i, \hat{m}^*(X_i))]. \end{aligned} \quad (5.6)$$

The first equality on the second line follows because: (i) $\text{test}^* = (X_{new}^*, Y_{new}^*)$ is independent (with respect to the bootstrap distribution) from train^* , and hence the inner conditional expectation is a non-conditional expectation using $\hat{m}^*(\cdot)$ as fixed (non-random because we condition on the bootstrap training data train^*); (ii) the bootstrap expectation $\mathbb{E}^*[g(X^*, Y^*)] = \frac{1}{n} \sum_{i=1}^n g(X_i, Y_i)$ for any function $g(\cdot)$.

Therefore, the bootstrap generalization error as represented in (5.6) is the average of the bootstrap errors over the observed original data (X_i, Y_i) . In particular, there is no need to generate (X_{new}^*, Y_{new}^*) as conceptually introduced above in step 1. The practical algorithm then looks as follows.

1. Generate $(X_1^*, Y_1^*), \dots, (X_n^*, Y_n^*)$ by resampling with replacement from the original data.
2. Compute the bootstrapped estimator $\hat{m}^*(\cdot)$ based on $(X_1^*, Y_1^*), \dots, (X_n^*, Y_n^*)$.
3. Evaluate $\text{err}^* = \frac{1}{n} \sum_{i=1}^n \rho(Y_i, \hat{m}^*(X_i))$.
4. Repeat steps 1–3 B times to obtain $\text{err}^{*1}, \dots, \text{err}^{*B}$. Approximate the bootstrap generalization error in (5.6) by

$$\frac{1}{B} \sum_{i=1}^B \text{err}^{*i},$$

and use it as an estimate for the true generalization error in (4.1).

5.2.6 Out-of-bootstrap sample for estimation of the generalization error

The bootstrap generalization error in (5.6) can be criticized because it evaluates at data points (X_i, Y_i) which may have been used in the estimator $\hat{m}^*(\cdot)$; see also step 3 in the computational algorithm. The over-optimism of the bootstrap generalization error (because some of the data points will be involved for training and testing) can be addressed by using the out-of-bootstrap sample.

Denote the bootstrap sample by $\mathcal{L}^* = \{Z_1^*, \dots, Z_n^*\}$. Clearly, not all of the original data points Z_i belong to the bootstrap sample \mathcal{L}^* . Thus, there is an out-of-bootstrap sample

$$\mathcal{L}_{out}^* = \cup_{i=1}^n \{Z_i; Z_i \notin \mathcal{L}^*\}$$

which we will use as test set.

The out-of-bootstrap error is defined in algorithmic terms. As usual, we repeat the bootstrap sampling B times which yields bootstrap and out-of-bootstrap samples

$$\mathcal{L}^{*(b)} \text{ and } \mathcal{L}_{out}^{*(b)}, \quad b = 1, \dots, B.$$

The out-of-bootstrap generalization error is then:

$$\text{gen-err}_{out-of-bootstrap} = \frac{1}{B} \sum_{b=1}^B \frac{1}{|\mathcal{L}_{out}^{*(b)}|} \sum_{i \in \mathcal{L}_{out}^{*(b)}} \rho(Y_i, \hat{m}^{*(b)}(X_i)).$$

Note that $\hat{m}^{*(b)}(\cdot)$ involves only data from $\mathcal{L}^{*(b)}$, and it holds that $(X_i, Y_i) \notin \mathcal{L}^{*(b)}$.

There is a probabilistic statement about the size of the out-of-bootstrap sample, i.e. the size of the test-sample:

$$\begin{aligned} \mathbb{P}^*[Z_i \in \mathcal{L}_{out}^*] &= \mathbb{P}^*[Z_1^* \neq Z_i, Z_2^* \neq Z_i, \dots, Z_n^* \neq Z_i] \\ &= \prod_{k=1}^n \mathbb{P}^*[Z_k^* \neq Z_i] = \left(1 - \frac{1}{n}\right)^n \approx \exp(-1) = 0.368. \end{aligned}$$

Thus, the chance that an original data point is in the out-of-bootstrap sample is a bit more than 1/3. Furthermore, for the expected size of the out-of-bootstrap sample:

$$\mathbb{E}^*[|\mathcal{L}_{out}^*|] = \mathbb{E}^*\left[\sum_{i=1}^n \mathbf{1}_{[Z_i \in \mathcal{L}_{out}^*]}\right] = n\mathbb{P}^*[Z_i \in \mathcal{L}_{out}^*] \approx 0.368n.$$

Hence, we can think of the out-of-bootstrap generalization error as some sort of cross-validation with 36.8% test data. The difference to cross-validation is that some of the other 63.2% of the original data are occurring more than once yielding a bootstrap sample (i.e. training sample) which is always of size n .

5.3 Double bootstrap

We describe here how the bootstrap can be used twice (or multiple times) aiming to improve a bootstrap confidence interval. The same idea can also be used for potentially improving other bootstrap estimates than confidence intervals.

Suppose we wish to construct a $(1 - \alpha)$ -confidence interval for a parameter θ based on some estimator $\hat{\theta}$. The bootstrap interval $I^*(1 - \alpha)$, as defined in (5.4), is not exact, i.e.,

$$\mathbb{P}[\theta \in I^*(1 - \alpha)] = 1 - \alpha + \Delta_n,$$

with some approximation error Δ_n which will diminish as $n \rightarrow \infty$.

The main idea is now as follows: when changing the nominal coverage to $1 - \alpha'$ and using $I^*(1 - \alpha')$, we can get an exact actual coverage

$$\mathbb{P}[\theta \in I^*(1 - \alpha')] = 1 - \alpha.$$

The problem is that α' is unknown. But another level of bootstrap can be used to **estimate** α' , denoted by $\hat{\alpha}'$, which typically achieves

$$\mathbb{P}[\theta \in I^*(1 - \hat{\alpha}')] = 1 - \alpha + \Delta'_n,$$

where the new approximation error Δ'_n is typically smaller than Δ_n above.

A second level of bootstrap

In case where the original data Z_1, \dots, Z_n are replaced by n i.i.d. bootstrap realizations Z_1^*, \dots, Z_n^* , we can get an exact answer for the level α' above. We can proceed in analogy to the setting above, step by step.

First, suppose the data is Z_1^*, \dots, Z_n^* . By using the bootstrap for the bootstrap data Z_1^*, \dots, Z_n^* , i.e., a second level bootstrap with $Z_1^{**}, \dots, Z_n^{**}$ (see below), we can construct a confidence interval $I^{**}(1 - \alpha)$ as in (5.4). We can now inspect the actual coverage for this second level bootstrap interval:

$$\mathbb{P}^*[\hat{\theta}_n \in I^{**}(1 - \alpha)] =: h^*(1 - \alpha)$$

for some function $h^* : [0, 1] \rightarrow [0, 1]$ which is increasing. Now use

$$1 - \alpha'^* = h^{*-1}(1 - \alpha) \tag{5.7}$$

so that

$$\mathbb{P}^*[\hat{\theta}_n \in I^{**}(1 - \alpha'^*)] = h^*(h^{*-1}(1 - \alpha)) = 1 - \alpha$$

is an exact confidence interval “in the bootstrap world” for the “parameter” $\hat{\theta}_n$. Therefore, the bootstrap estimate for the adjusted nominal coverage level is

$$\widehat{1 - \alpha'} = 1 - \alpha'^* \text{ from (5.7).}$$

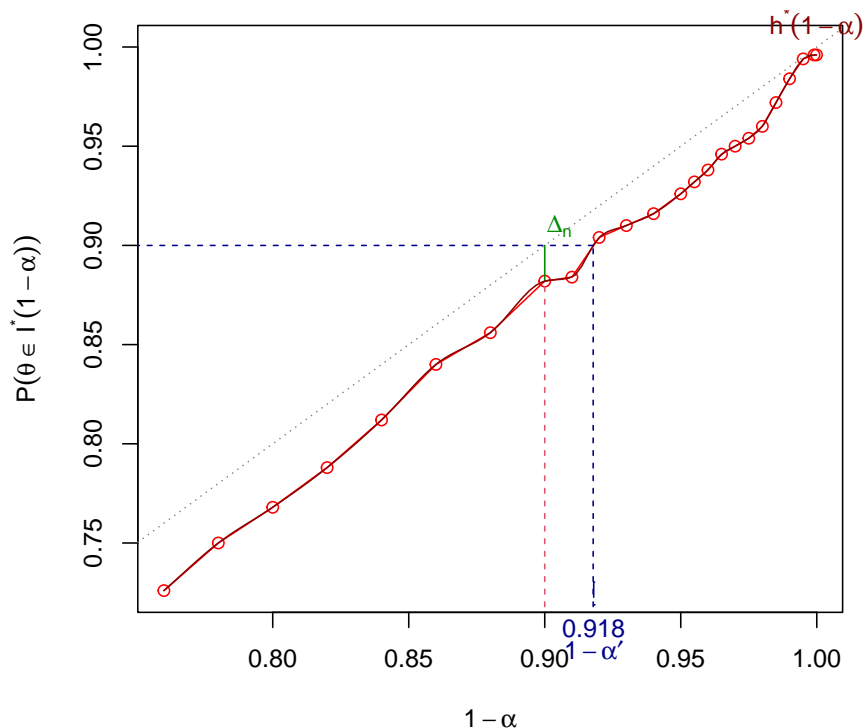


Figure 5.2: Visualizing $h^*(1 - \alpha) := \mathbb{P}^*[\hat{\theta}_n \in I^{**}(1 - \alpha)]$ in double bootstrap (for $1 - \alpha = 0.90$ and $x = 0$ in the example below)

Computation of bootstrap adjusted nominal level $1 - \alpha'^*$

We can use a double (two-level) bootstrap scheme to approximately compute the value $1 - \alpha'^*$ in (5.7).

1. Draw a bootstrap sample Z_1^*, \dots, Z_n^* by resampling with replacement.
 - (a) Compute a bootstrap interval. That is, generate a second level bootstrap sample

$$Z_1^{**}, \dots, Z_n^{**}$$

by resampling with replacement from Z_1^*, \dots, Z_n^* . Then, analogous to (5.4), construct the double bootstrap confidence interval

$$I^{**}(1 - \alpha) = [\hat{\theta}^* - \hat{q}_{1-\alpha/2}^*, \hat{\theta}^* - \hat{q}_{\alpha/2}^*],$$

where $\hat{q}_\alpha^* = \alpha$ - quantile of $\hat{\theta}^{**} - \hat{\theta}^*$, $\hat{\theta}^{**} = g(Z_1^{**}, \dots, Z_n^{**})$. This is computed by repeating step 1(a) B times to approximate \hat{q}_α^* by the empirical α -quantile of $\hat{\theta}^{**1} - \hat{\theta}^*, \dots, \hat{\theta}^{**B} - \hat{\theta}^*$.

- (b) Evaluate whether the “parameter” $\hat{\theta}$ in the “bootstrap world” is in $I^{**}(1 - \alpha)$: i.e., consider

$$\text{cover}^*(1 - \alpha) = \mathbf{1}_{[\hat{\theta} \in I^{**}(1 - \alpha)]}.$$

2. Repeat M times all of step 1 to obtain $\text{cover}^{*1}(1 - \alpha), \dots, \text{cover}^{*M}(1 - \alpha)$. This amounts to M first level bootstrap replications $\mathbf{Z}^{*m} = Z_1^{*m}, \dots, Z_n^{*m}$ ($m = 1, \dots, M$) and for **each** \mathbf{Z}^{*m} , we run B second level bootstrap replications for step 1(a). Use

$$p^*(\alpha) := \frac{1}{M} \sum_{i=1}^M \text{cover}^{*i}(1 - \alpha) \quad (5.8)$$

as an approximation for $\mathbb{P}^*[\hat{\theta} \in I^{**}(1 - \alpha)]$.

3. Vary α (in all of step 1 and 2!) to find α'^* such that

$$p^*(\alpha'^*) = 1 - \alpha \quad (\text{the desired nominal level}) \quad \text{and use} \quad \widehat{1 - \alpha'} = 1 - \alpha'^*.$$

The search for α'^* (a “zero finding problem”) can be done on a grid and/or by using a bisection strategy, in R by using `uniroot()`

The total amount of computation requires $B \cdot M$ bootstrap samples. In case where the bootstrap interval in (5.4) is computed with B bootstrap samples, and hence also the interval I^{**} in step 1(a), the adjustment with the double bootstrap may be less important and it is then reasonable to use $M < B$ since the magnitude of M only determines the approximation for computing the actual level $\mathbb{P}^*[\hat{\theta} \in I^{**}(1 - \alpha)]$ (for I^{**} computed with B bootstrap replications).

An example

We illustrate now the double bootstrap for confidence intervals in curve estimation. Figure 5.3 displays the data, having sample size $n = 100$, and a curve estimator.

Figure 5.4 then shows how the double bootstrap is used to estimate the actual coverage: displayed is an approximation of $\mathbb{P}^*[\hat{\theta}_n \in I^{**}(1 - \alpha)]$ for various nominal levels $1 - \alpha$. It also indicates the values for the corrected levels $1 - \alpha'^*$ and it also demonstrates the effect when using a double-bootstrap corrected confidence interval instead of an ordinary interval.

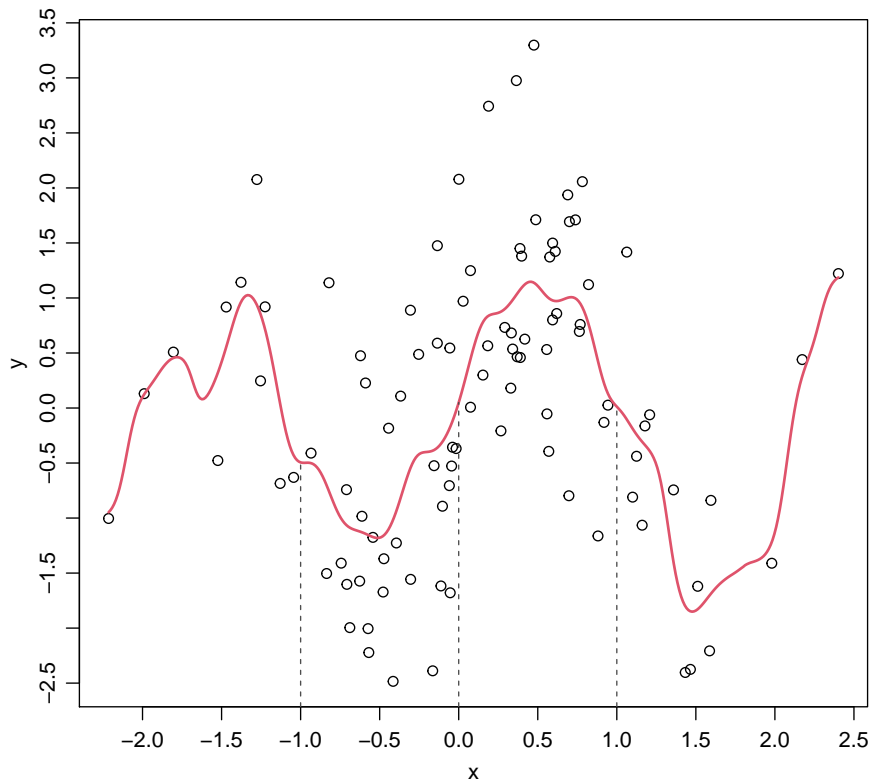


Figure 5.3: Data ($n = 100$) and estimated curve (red) using a Nadaraya Watson Gaussian kernel estimator with bandwidth $h = 0.25$.

5.4 Model-based bootstrap

Efron's nonparametric bootstrap can be viewed as simulating from the empirical distribution \hat{P}_n : that is, we simulate from a very general estimated nonparametric model, where the model says that the data is i.i.distributed with an unknown distribution P .

5.4.1 Parametric bootstrap

Instead of such a general nonparametric model, we sometimes assume that the data are realizations from

$$Z_1, \dots, Z_n \text{ i.i.d. } \sim P_\theta,$$

where P_θ is given up to an unknown parameter (vector) θ .

As one among very many examples: the data could be real-valued assumed to be from the parametric model

$$X_1, \dots, X_n \text{ i.i.d. } \sim \mathcal{N}(\mu, \sigma^2), \theta = (\mu, \sigma^2).$$

In order to simulate from the parametric model, we first estimate the unknown parameter θ by $\hat{\theta}$ such as least squares in regression or maximum likelihood in general. The parametric bootstrap then proceeds by using

$$Z_1^*, \dots, Z_n^* \text{ i.i.d. } \sim P_{\hat{\theta}},$$

instead of (5.2). Everything else, e.g. construction of confidence intervals, can then be done exactly as for Efron's nonparametric bootstrap.

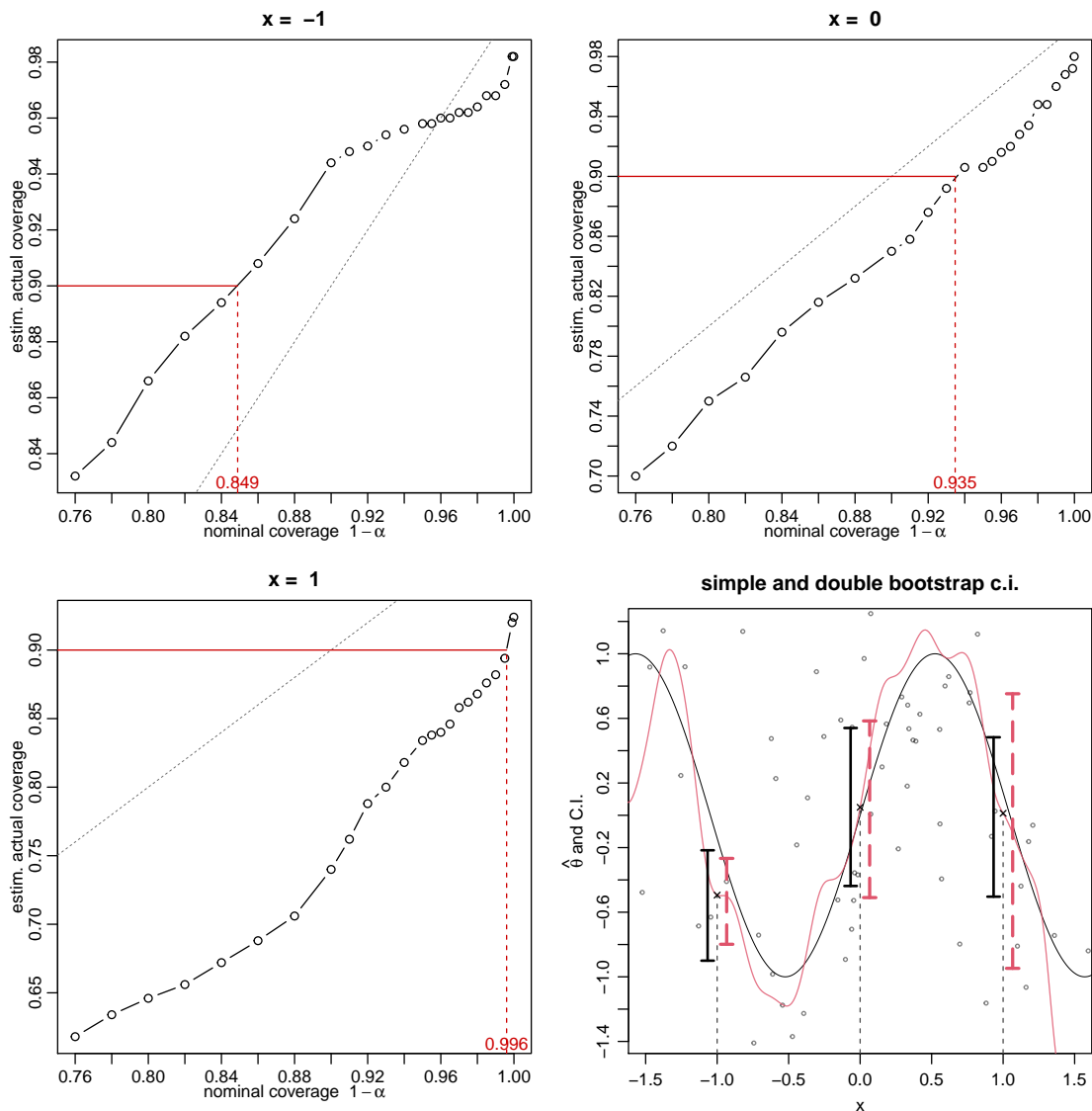


Figure 5.4: Double bootstrap confidence intervals for nonparametric curve at three predictor points $x \in \{-1, 0, 1\}$. The data ($n = 100$) and estimated curve are shown in Figure 5.3. The first three panels show the estimated actual coverages ($p^*(\alpha)$) of a bootstrap confidence interval by using the double bootstrap. The values $1 - \alpha'^*$ (for actual level $1 - \alpha = 0.9$) are 0.86, 0.922, 0.998 for the points $x = -1, 0, 1$, respectively. The fourth panel shows the ordinary bootstrap confidence intervals (solid line) and the double bootstrap corrected versions (dotted line, in red) for $x \in \{-1, 0, 1\}$. The double bootstrap was used with $B = 1000$ and $M = 500$.

Advantages and disadvantages

Why should we choose the parametric instead of the nonparametric bootstrap? The answer is “classical”: if the parametric model is a very good description for the data, then the parametric bootstrap should yield more accurate variance estimates or confidence intervals since $P_{\hat{\theta}}$ is then “closer” to the true data-generating P than the nonparametric empirical distribution \hat{P}_n . Particularly when sample size n is small, the nonparametric estimate \hat{P}_n may be poor. On the other hand, the nonparametric bootstrap is not (or less) sensitive to model-misspecification.

5.4.2 Parametric bootstrap for model structures beyond i.i.d.

We exemplify the principle considering two examples:

1) Linear model with fixed predictors

A linear model with fixed predictors $x_i \in \mathbb{R}^p$ and Gaussian errors

$$Y_i = \beta^\top x_i + \varepsilon_i \quad (i = 1, \dots, n),$$

$$\varepsilon_1, \dots, \varepsilon_n \text{ i.i.d. } \sim \mathcal{N}(0, \sigma^2), \quad \theta = (\beta, \sigma^2)$$

is a parametric model. The bootstrap sample can then be constructed as follows:

1. Simulate $\varepsilon_1^*, \dots, \varepsilon_n^*$ i.i.d. $\sim \mathcal{N}(0, \hat{\sigma}^2)$.
2. Construct

$$Y_i^* = \hat{\beta}^\top x_i + \varepsilon_i^*, \quad i = 1, \dots, n.$$

The parametric bootstrap regression sample is then

$$(x_1, Y_1^*), \dots, (x_n, Y_n^*),$$

where the predictors x_i are as for the original data.

2) Autoregressive models for time series

A Gaussian autoregressive model of order p for stationary time series is

$$X_t = \sum_{j=1}^p \phi_j X_{t-j} + \varepsilon_t \quad (t = 1, \dots, n),$$

$$\varepsilon_1, \dots, \varepsilon_n \text{ i.i.d. } \sim \mathcal{N}(0, \sigma^2),$$

where $X_t \in \mathbb{R}$. Such a model produces correlated observations and is widely used for describing time-dependent observations. Parametric bootstrapping can then be done as follows:

1. Generate $\varepsilon_1^*, \dots, \varepsilon_{n+m}^*$ i.i.d. $\sim \mathcal{N}(0, \hat{\sigma}^2)$ with “burn-in time” $m \approx 1000$.
2. Construct recursively, starting with $X_0^* = X_{-1}^* = \dots = X_{-p+1}^* = 0$,

$$X_t^* = \sum_{j=1}^p \hat{\phi}_j X_{t-j}^* + \varepsilon_t^*, \quad t = 1, \dots, n + m.$$

3. Use the bootstrap sample

$$X_{m+1}^*, \dots, X_{n+m}^*.$$

The reason to throw away the first values X_1^*, \dots, X_m^* is to obtain a bootstrap sample which is approximately a stationary process (by choosing m large, the arbitrary starting values in step 2 will be almost forgotten).

5.4.3 The model-based bootstrap for regression

A compromise between Efron's non- and the parametric bootstrap for regression is given by assuming possibly non-Gaussian errors. The model for the original data is

$$Y_i = m(x_i) + \varepsilon_i, \\ \varepsilon_1, \dots, \varepsilon_n \text{ i.i.d. } \sim P_\varepsilon,$$

where P_ε is unknown with expectation 0. The regression function $m(\cdot)$ may be parametric or nonparametric. The model-based bootstrap works then as follows:

1. Estimate \hat{m} from the original data and compute the residuals $r_i = Y_i - \hat{m}(x_i)$.
2. Consider the centered residuals $\tilde{r}_i = r_i - \frac{1}{n} \sum_{i=1}^n r_i$. In case of linear regression with an intercept, the residuals are already centered. Denote the empirical distribution of the centered residuals by $\hat{P}_{\tilde{r}}$.

3. Generate

$$\varepsilon_1^*, \dots, \varepsilon_n^* \text{ i.i.d. } \sim \hat{P}_{\tilde{r}}.$$

Note that $\hat{P}_{\tilde{r}}$ is an estimate of P_ε .

4. Construct the bootstrap response variables

$$Y_i^* = \hat{m}(x_i) + \varepsilon_i^*, \quad i = 1, \dots, n,$$

and the bootstrap sample is then $(x_1, Y_1^*), \dots, (x_n, Y_n^*)$.

Having the bootstrap sample from step 4, we can then proceed as for Efron's nonparametric bootstrap for constructing variance estimates or confidence intervals.

The advantage of the model-based bootstrap is that we do not rely on a Gaussian error assumption. The same discussion then applies about advantages and disadvantages as in section 5.4.1.

Chapter 6

Classification

6.1 Introduction

Often encountered in applications is the situation where the response variable Y takes values in a finite set of labels. For example, the response Y could encode the information whether a patient has disease type A, B or C; or it could describe whether a customer responds positively about a marketing campaign.

We always encode such information about classes or labels by the numbers $0, 1, \dots, J - 1$, that is $Y \in \{0, \dots, J - 1\}$, without any ordering among these numbers. In other words, our sample space consists of J different *groups* (“sub-populations”) and our goal is to *classify* the observations using the (p -dimensional) explanatory variables.

Given data which are realizations from

$$(X_1, Y_1), \dots, (X_n, Y_n) \text{ i.i.d. ,}$$

the goal is often to assign the probabilities

$$\pi_j(x) = \mathbb{P}[Y = j \mid X = x] \quad (j = 0, 1, \dots, J - 1),$$

which is similar to the regression function $m(x) = \mathbb{E}[Y \mid X = x]$ in regression. The multivariate function $\pi_j(\cdot)$ then also allows to predict the class Y_{new} at a new observed predictor X_{new} .

6.2 The Bayes classifier

A classifier $\mathcal{C} : \mathbb{R}^p \rightarrow \{0, \dots, J - 1\}$ is a function which assigns to a predictor $X \in \mathbb{R}^p$ a class or label which is a prediction for the corresponding Y . The quality of a classifier is often measured by the expected zero-one¹ test set error:

$$\mathbb{P}[\mathcal{C}(X_{new}) \neq Y_{new}].$$

In case where $\mathcal{C} = \widehat{\mathcal{C}}$ is estimated from training data, we consider the generalization error

$$\mathbb{P}_{\text{train}, (X_{new}, Y_{new})}[\widehat{\mathcal{C}}(X_{new}) \neq Y_{new}].$$

¹More generally (and realistically, even just for binary disease testing!), $L(k, k')$ is the loss for predicting class k' when k is true, with $L(j, j) \equiv 0$, $L(\cdot)$ corresponds to a $J \times J$ matrix with diagonal zero and positive numbers otherwise, typically *not* symmetric.

The optimal classifier with respect to the zero-one error is the **Bayes classifier**, defined “pointwise”, i.e., for each x individually, as

$$\mathcal{C}_{Bayes}(x) = \arg \max_{0 \leq j \leq J-1} \pi_j(x). \quad (6.1)$$

Its corresponding expected zero-one ² test set error is called the **Bayes risk**

$$\mathbb{P}[\mathcal{C}_{Bayes}(X_{new}) \neq Y_{new}].$$

In practice, we do not know $\pi_j(\cdot)$ (and hence the Bayes classifier or risk are as unknown as MSE or bias in regression). Various methods and models are then used to come up with multivariate function estimates, either parametric or nonparametric, to obtain $\hat{\pi}_j(\cdot)$. With this, we can then estimate a classifier by plugging into the Bayes classifier

$$\hat{\mathcal{C}}(x) = \arg \max_{0 \leq j \leq J-1} \hat{\pi}_j(x). \quad (6.2)$$

Such estimated classifiers are widely used, by using various models for $\pi_j(\cdot)$. However, there are also direct ways to come up with estimated classifiers without trying to estimate the conditional probability function $\pi_j(\cdot)$: an important class of examples are the support vector machines (which we will not discuss in this course). Another is the discriminant analysis view:

6.3 The view of discriminant analysis

6.3.1 Linear discriminant analysis

For the so-called linear discriminant analysis, we assume the following model:

$$\begin{aligned} (X | Y = j) &\sim \mathcal{N}_p(\mu_j, \Sigma), \\ \mathbb{P}[Y = j] &= p_j, \quad \sum_{j=0}^{J-1} p_j = 1; \quad j = 0, 1, \dots, J-1. \end{aligned} \quad (6.3)$$

The conditional distribution of $Y | X$ can then be computed by the Bayes formula

$$\mathbb{P}[Y = j | X = x] = \pi_j(x) = \frac{f_{X|Y=j}(x)p_j}{\sum_{k=0}^{J-1} f_{X|Y=k}(x)p_k}, \quad (6.4)$$

where $f_{X|Y=j}(\cdot)$ denotes the density of the p -dimensional Gaussian distribution $\mathcal{N}_p(\mu_j, \Sigma)$. We can interpret this conditional distribution as the a-posteriori distribution of Y given X by using the a-priori distribution p_j for Y .

²In the general case with losses $L(j, k)$, the Bayes classifier $\mathcal{C}(x)$ must minimize $\sum_{j=0}^{J-1} L(j, k)\pi_j(x)$ over k , pointwise for each x , i.e.,

$$\mathcal{C}_{Bayes_G}(x) = \arg \min_{0 \leq k \leq J-1} \sum_{j=0}^{J-1} L(j, k)\pi_j(x).$$

The unknown parameters in (6.4) are μ_j and Σ (or Σ_j if the covariances may differ per group, see 6.3.2) which can be estimated by standard moment estimators:

$$\begin{aligned}\hat{\mu}_j &= \frac{\sum_{i=1}^n X_i \mathbf{1}_{[Y_i=j]}}{\sum_{i=1}^n \mathbf{1}_{[Y_i=j]}} = \frac{1}{n_j} \sum_{i:Y_i=j} X_i, \quad \text{where } n_j = \#\{i; Y_i = j\}, \\ \hat{\Sigma} &= \frac{1}{n-J} \sum_{j=0}^{J-1} \sum_{i=1}^n (X_i - \hat{\mu}_j)(X_i - \hat{\mu}_j)^\top \mathbf{1}_{[Y_i=j]}, \quad \text{and} \\ \hat{\Sigma}_j &= \frac{1}{n_j-1} \sum_{i=1}^n (X_i - \hat{\mu}_j)(X_i - \hat{\mu}_j)^\top \mathbf{1}_{[Y_i=j]}\end{aligned}\tag{6.5}$$

Moreover, we need to specify the (a-priori) distribution for Y : quite often, one takes $\hat{p}_j = n^{-1} \sum_{i=1}^n \mathbf{1}_{[Y_i=j]} = n_j/n$. Using these parameter estimates, we obtain a classifier via formula (6.4) and (6.2):

$$\begin{aligned}\hat{C}_{lin.discr.}(x) &= \arg \max_{0 \leq j \leq J-1} \hat{\delta}_j(x), \\ \hat{\delta}_j(x) &= x^\top \hat{\Sigma}^{-1} \hat{\mu}_j - \hat{\mu}_j^\top \hat{\Sigma}^{-1} \hat{\mu}_j / 2 + \log(\hat{p}_j) = \\ &= (x - \hat{\mu}_j / 2)^\top \hat{\Sigma}^{-1} \hat{\mu}_j + \log(\hat{p}_j).\end{aligned}$$

This classifier is called “linear discriminant classifier” because the estimated decision functions $\hat{\delta}_j(\cdot)$ are **linear** in the predictor variables x and since the regions are determined by $\hat{\delta}_j(x) - \hat{\delta}_{j'}(x) \geq 0$, the decision boundaries are hyperplanes (i.e., lines for $p = 2$), $x^\top \hat{\Sigma}^{-1} (\hat{\mu}_j - \hat{\mu}_{j'}) + c_{j,j'} = 0$.

6.3.2 Quadratic discriminant analysis

The model underlying linear discriminant analysis assumes equal covariances for all the groups, see formula (6.3). More generally, we sometimes assume

$$\begin{aligned}(X | Y = j) &\sim \mathcal{N}_p(\mu_j, \Sigma_j), \\ P[Y = j] &= p_j, \quad \sum_{j=0}^{J-1} p_j = 1; \quad j = 0, 1, \dots, J-1.\end{aligned}$$

with non-equal covariances for all the groups $j \in \{0, 1, \dots, J-1\}$. Analogously to linear discriminant analysis, we then obtain discriminant functions which are **quadratic** in the predictor variables x ,

$$\hat{\delta}_j(x) = -\log(\det(\hat{\Sigma}_j))/2 - (x - \hat{\mu}_j)^\top \hat{\Sigma}_j^{-1} (x - \hat{\mu}_j)/2 + \log(\hat{p}_j).$$

Such quadratic discriminant classifiers are more flexible and general than their linear “cousins”: but the price to pay for this flexibility are $J \cdot p(p+1)/2$ parameters for all covariance matrices Σ_j ($j = 0, 1, \dots, J-1$) instead of $p(p+1)/2$ for one Σ in linear discriminant analysis. Particularly when p is large, quadratic discriminant analysis typically overfits (too large variability).

6.4 The view of logistic regression

As we have seen in (6.1), all we need to know for a good classifier is a good estimator for the conditional probabilities $\pi_j(\cdot)$.

6.4.1 Binary classification

For the case with binary response $Y \in \{0, 1\}$, the conditional probability function

$$\pi(x) = \mathbb{P}[Y = 1 \mid X = x]$$

provides the full information about the conditional distribution of Y given X (since $\mathbb{P}[Y = 0 \mid X = x] = 1 - \pi(x)$). The logistic model for $\pi(\cdot)$ in general is

$$\begin{aligned} \log\left(\frac{\pi(x)}{1 - \pi(x)}\right) &= g(x), \\ g : \mathbb{R}^p &\rightarrow \mathbb{R}. \end{aligned} \quad (6.6)$$

Note that the so-called logistic transform $\pi \mapsto \log(\pi/(1 - \pi))$ maps the interval $(0, 1)$ to the real line \mathbb{R} . Models for real-valued functions can thus be used for $g(\cdot)$.

Linear logistic regression

In analogy to linear regression in chapter 1, a popular and simple model for $g(\cdot)$ is

$$g(x) = \sum_{j=1}^p \beta_j x_j. \quad (6.7)$$

The model in (6.6) with $g(\cdot)$ from (6.7) is called linear logistic regression.

Fitting of the parameters is usually done by maximum-likelihood. For fixed predictors x_i , the probability structure of the response variables is

$$Y_1, \dots, Y_n \text{ independent, } Y_i \sim \text{Bernoulli}(\pi(x_i)).$$

The likelihood is thus

$$L(\beta; (x_1, Y_1), \dots, (x_n, Y_n)) = \prod_{i=1}^n \pi(x_i)^{Y_i} (1 - \pi(x_i))^{1 - Y_i},$$

and the negative log-likelihood becomes

$$\begin{aligned} -\ell(\beta; (x_1, Y_1), \dots, (x_n, Y_n)) &= -\sum_{i=1}^n (Y_i \log(\pi(x_i)) + (1 - Y_i) \log(1 - \pi(x_i))) \\ &= -\sum_{i=1}^n \left(Y_i \sum_{j=1}^p \beta_j x_{ij} - \log(\exp(\sum_{j=1}^p \beta_j x_{ij}) + 1) \right). \end{aligned}$$

Minimization of the negative log-likelihood is a **nonlinear** problem. It is usually solved numerically by versions of Newton's gradient descent method which then yield the maximal likelihood estimate $\hat{\beta}_{p \times 1}$.

Asymptotic inference and an example

Based on classical theory for maximum likelihood estimation, the distribution of the estimated coefficients $\hat{\beta}$ can be derived from an asymptotic point of view where the sample size $n \rightarrow \infty$. The output in R then yields a summary of the estimated coefficients, of their estimated standard errors $\widehat{s.e.}(\hat{\beta}_j)$, of the individual t -test statistics $\hat{\beta}_j / \widehat{s.e.}(\hat{\beta}_j)$ (which are asymptotically $\mathcal{N}(0, 1)$ distributed under the null-hypothesis $H_{0,j} : \beta_j = 0$) and the P-values for the individual t -tests for the null-hypotheses $H_{0,j}$, $j = 1, \dots, p$.

As an example, we consider a dataset about survival of prenatal babies (represented by the response variable $Y \in \{0, 1\}$, survival: $Y = 1$) as a function of age (in weeks), weight (in grams) and 3 other clinical variables (see Fig. 6.1). Fitting a linear logistic regression model can be done in R using the function `glm` (GLM = Generalized Linear Model) :


```
> d.baby <- read.table("http://stat.ethz.ch/Teaching/Datasets/baby.dat", header=TRUE)
> fit <- glm(Survival ~ ., data = d.baby, family = "binomial")
> summary(fit)
```

Call:

```
glm(formula = Survival ~ ., family = "binomial", data = d.baby)
```

Coefficients:

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	-3.0933685	14.3053767	-0.216	0.8288
Weight	0.0037341	0.0008468	4.410	1.03e-05 ***
Age	0.1588001	0.0761061	2.087	0.0369 *
X1.Apgar	0.1159864	0.1108339	1.046	0.2953
X5.Apgar	0.0611499	0.1202222	0.509	0.6110
pH	-0.7380214	1.8964578	-0.389	0.6972

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 319.28 on 246 degrees of freedom

Residual deviance: 236.14 on 241 degrees of freedom

AIC: 248.14

Number of Fisher Scoring iterations: 4

As seen from the P -values for the individual hypotheses $H_{0,j} : \beta_j = 0$, the predictor variables weight and age, which are strongly correlated, turn out to be significant for describing whether a prenatal baby will survive.

For classification, we can extract the probabilities with the function `predict`:

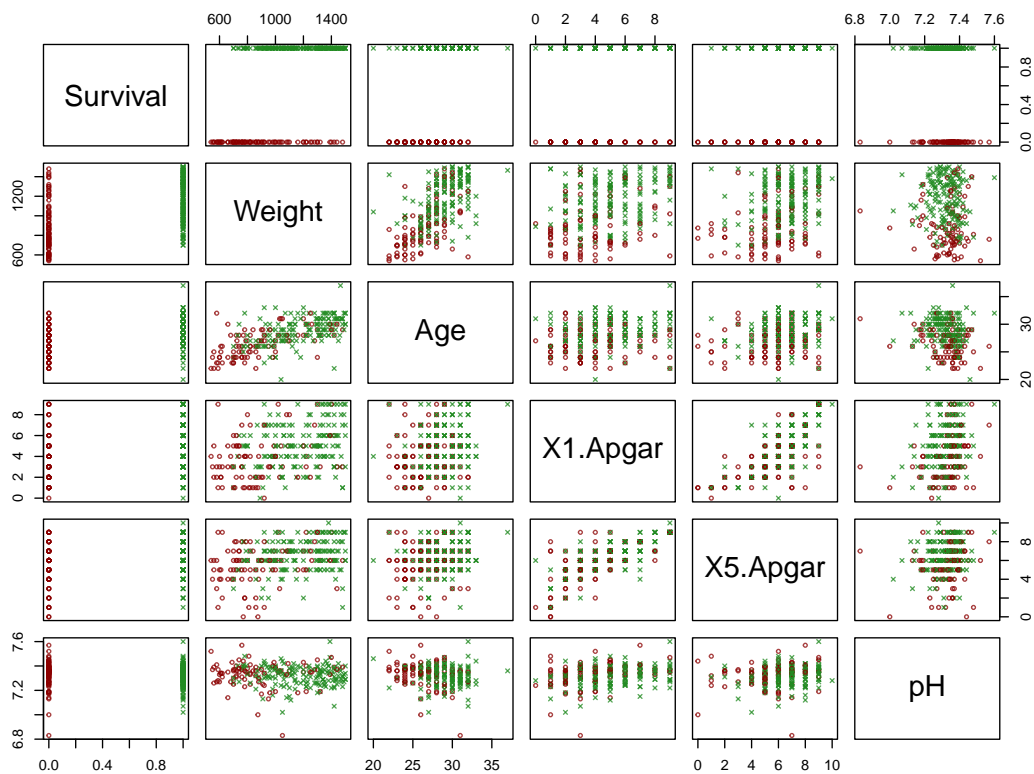


Figure 6.1: Survival of 247 prenatal babies as a function of age (in weeks), weight (in grams) and 3 other clinical variables.

```
predict(fit, type="response")
```

which yields the estimated probabilities $\hat{\pi}(x_i)$, $i = 1, \dots, n$. Thus, the average in-sample classification accuracy $\frac{1}{n} \sum_{i=1}^n \mathbf{1}_{[Y_i = \hat{c}(x_i)]}$ is given by

```
mean((predict(fit, type = "response") > 0.5) == d.baby$Survival) ,
```

which turns out to be 0.789. Remember that such an in-sample estimate is over-optimistic for measuring the true generalization performance.

Linear logistic regression or LDA?

In linear logistic regression, the model for the log-odds (the logit-transform)

$$\log \left(\frac{\pi(x)}{1 - \pi(x)} \right) = \sum_{j=1}^p \beta_j x_j$$

is *linear* in the predictors. But also the log-odds from LDA in model (6.4) yield a linear model in the predictor variables which is a consequence of the Gaussian assumption. Thus, the two methods are quite similar. In our example, the LDA coefficients of the five predictors are 0.00309, 0.124, 0.0685, 0.067, -0.763, and the in-sample accuracy is slightly smaller, at 0.785.

The linear logistic regression does not make any assumptions on the predictor variables such a multivariate Gaussianity; but the restriction comes in by requiring a linear log-odds function. Logistic regression can also be used when having factor variables (e.g. $x \in \{0, 1, \dots, K\}$) as predictors.

While LDA does make a Gaussian assumption for the predictors, it can also be used as a “linear technique” in case of non-Gaussian predictors (even with factors). Empirically, LDA and linear logistic regression yield similar answers, even for non-Gaussian predictors, with respect to classification accuracy.

6.4.2 Multiclass case, $J > 2$

Logistic regression cannot be directly applied to the multiclass case with $Y \in \{0, 1, \dots, J-1\}$ and $J > 2$. There are several (general) approaches:

1) We can always encode a multiclass problem with J classes as J binary class problems by using

$$Y_i^{(j)} = \begin{cases} 1 & \text{if } Y_i = j, \\ 0 & \text{otherwise.} \end{cases}$$

This means that we consider class j against all remaining classes.

We can then run (linear or “general”) logistic regressions

$$\log \left(\frac{\pi_j(x)}{1 - \pi_j(x)} \right) = g_j(x) = \sum_{r=1}^p \beta_r^{(j)} x_r$$

yielding estimates

$$\hat{\pi}_j(x) = \frac{\exp(\sum_{r=1}^p \hat{\beta}_r^{(j)} x_r)}{1 + \exp(\sum_{r=1}^p \hat{\beta}_r^{(j)} x_r)}. \quad (6.8)$$

The estimates $\hat{\pi}_j(\cdot)$ will not sum up to one: but a normalization will do the job,

$$\tilde{\pi}_j(x) = \frac{\hat{\pi}_j(x)}{\sum_{j=0}^{J-1} \hat{\pi}_j(x)}.$$

If the goal is only classification, i.e., $\hat{C}(x) = \arg \max_{0 \leq j \leq J-1} \hat{\pi}_j(x)$, the normalization is not needed.

2) Note that for this specific case (parametric linear logistic), the problem can be formulated more nicely, but slightly differently, using the *multinomial* distribution, and solved by maximum likelihood, very similarly to the linear logistic ($J = 2$) case³. This is implemented in R's `multinom()` function (standard package `nnet`).

3) A related approach works with a *reference class*, say, $j = 0$, and instead of “one against all” models “*everyone against the reference*”,

$$\log(\pi_j(x)/\pi_0(x)) = g_j(x), \quad \text{for } j = 1, \dots, J-1.$$

4) Other ways are also possible to reduce a J class problem into several binary class problems. Instead of modelling class j against the rest, we can also model class j against another class k for all pairs (j, k) with $j \neq k$. This will require fitting $\binom{J}{2}$ logistic models (involving $\binom{J}{2} \cdot p$ estimated parameters in the linear case), instead of J models in the one against the rest approach. We should keep in mind that the models are different: in the one against the rest approach, the coefficients in (6.8) describe the effect of the predictors for distinguishing class j from all others, and in a pairwise approach, we would model the distinction between two classes.

5) Note that for the particular situation of **ordered classes**, one can use a more simple “proportional odds” model

$$\text{logit}(P[Y \leq k | x]) = \alpha_k + g(x), \quad k = 0, 1, \dots, J-1, \quad \text{with } \alpha_0 \leq \alpha_1 \leq \dots \leq \alpha_{J-1}. \quad (6.9)$$

In R, this particularly simple proportional odds model is available via `polr()` (= **proportional odds logistic regression**, standard package `MASS`).

6.5 Receiver operating characteristic (ROC) curve

One way to measure performance of a classifier is to measure the misclassification error on previously unseen test data. Lets assume the binary case $Y \in \{0, 1\}$. The estimated probabilities $\hat{\pi}(x) = \hat{P}(Y = 1 | X = x)$ are then converted by to a predicted class label using a threshold of θ :

$$\hat{Y}(x) = 1 \text{ iff } \hat{\pi}(x) > \theta.$$

For a given value of θ , the outcome on test data can be converted into a 2×2 table, the “confusion matrix”,

estimated class	$\hat{Y} = 0$	$\hat{Y} = 1$
true class		
$Y = 0$	TN	FP
$Y = 1$	FN	TP,

where TP is the number of so-called true positives, FP number of false positives, FN number of false negatives and TN number of true negatives. Let also (the table row sums)

³The Likelihood is $L = \pi_0^{n_0} \pi_1^{n_1} \cdot \pi_{J-1}^{n_{J-1}}$, the log-likelihood therefore $l = \sum_{j=0}^{J-1} n_j \log \pi_j$, where the constraint $\sum_j \pi_j \equiv 1$ has to be fulfilled.

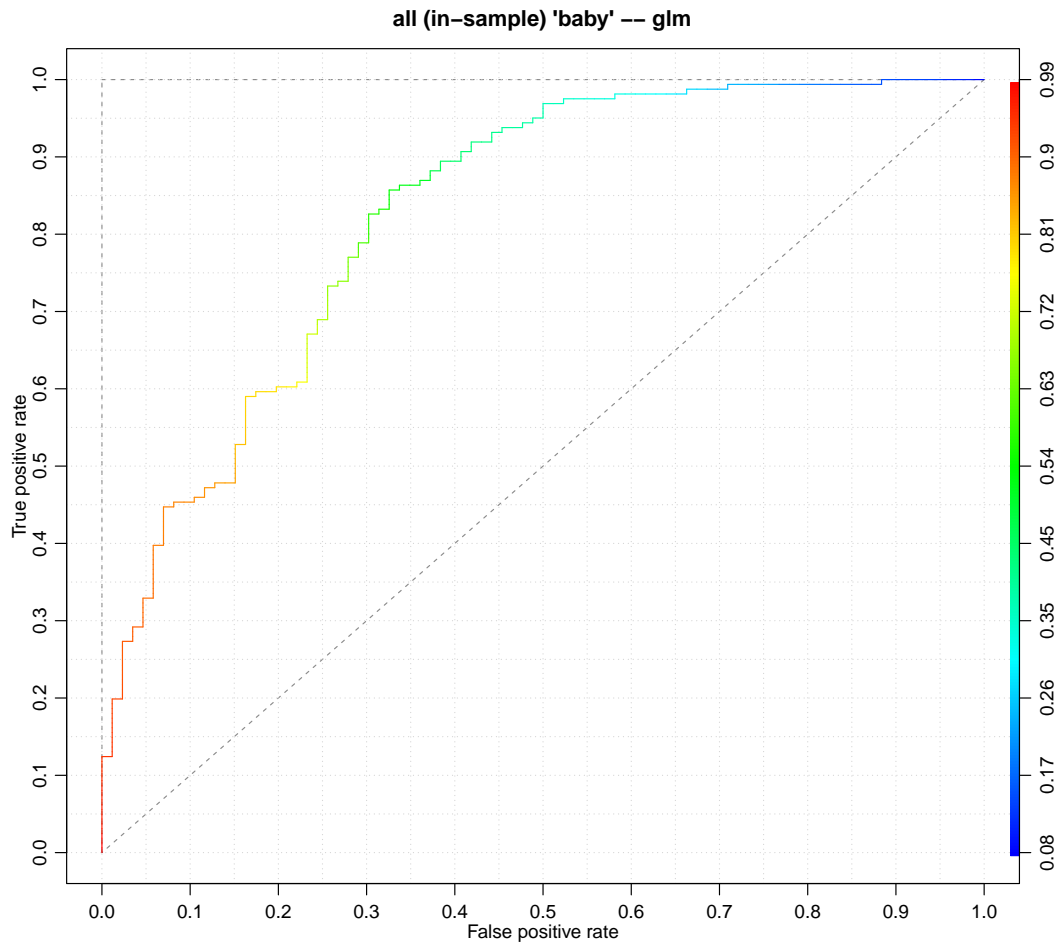


Figure 6.2: ROC curve for the `glm(., d.baby)` fit; using CRAN R package `ROCR`, its `prediction()`, `performance()` and own R code.

$N := TN + FP = \#\{Y_i = 0\}$ be the total number of class 0 labels and analogously for $P := FN + TP$ for $Y_i = 1$.

The table is a function of the used threshold θ . One way to map the behaviour of the classifier under different thresholds is to define

$$\begin{aligned} \text{sensitivity}(\theta) &= \frac{TP}{P} \\ \text{specificity}(\theta) &= \frac{TN}{N} \implies 1 - \text{specificity}(\theta) = \frac{FP}{N} \end{aligned}$$

and plot sensitivity as a function of specificity or rather as function of 1-specificity. Both sensitivity and (1-) specificity are in $[0, 1]$. One can always reach a value of 1 for either of them (but then often putting the other quantity to 0). Note that these quantities are defined for a finite test sample size but can be seen as estimates of the underlying population probabilities of $P(\hat{Y} = 1|Y = 1)$ and $P(\hat{Y} = 0|Y = 0)$ respectively.

What is often compared is the area under the curve (AUC). Randomly guessing will always achieve an AUC value of 0.5. A perfect classifier will achieve AUC=1. A simple argument shows that

$$AUC = P(\hat{\pi}_{I_0} < \hat{\pi}_{I_1}),$$

where I_0 is an index sampled uniformly at random from all samples with $Y = 0$ whereas I_1 is an index sampled uniformly at random from all $Y = 1$ samples.

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

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

7.2 Additive models

Additive models require additivity of the function: the model is

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

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 ($\mathbb{R} \rightarrow \mathbb{R}$) function.

¹as 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

Backfitting² 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

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

for any response vector $(U_1, \dots, U_n)^\top$. 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 $j = 1, \dots, p$.
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 $\mathbf{Y} = (Y_1, \dots, Y_n)^\top$ and $\hat{\mathbf{g}}_j = (\hat{g}_j(X_{1j}), \dots, \hat{g}_j(X_{nj}))^\top$. 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` (generalized additive model) 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

²Backfitting was the algorithm for fitting additive models when they were proposed (Hastie and Tibshirani, 1990, “Generalized Additive Models”), available in R package `gam`. The more sophisticated `gam()` function from package `mgcv` uses a different algorithm for fitting.

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")
> chgNames <- function(dfr, oldNms, newNms) {
  colnames(dfr)[colnames(dfr) %in% oldNms] <- newNms
  dfr
}
> d.ozone <- chgNames(ozone,
  oldNms = c("upo3", "wdsp", "hmdt", "sbtp"),
  newNms = c("O3", "wind", "humidity", "temp"))

> pairs(d.ozone, pch = ".", gap = 0) ## --> Scatterplot Matrix
```

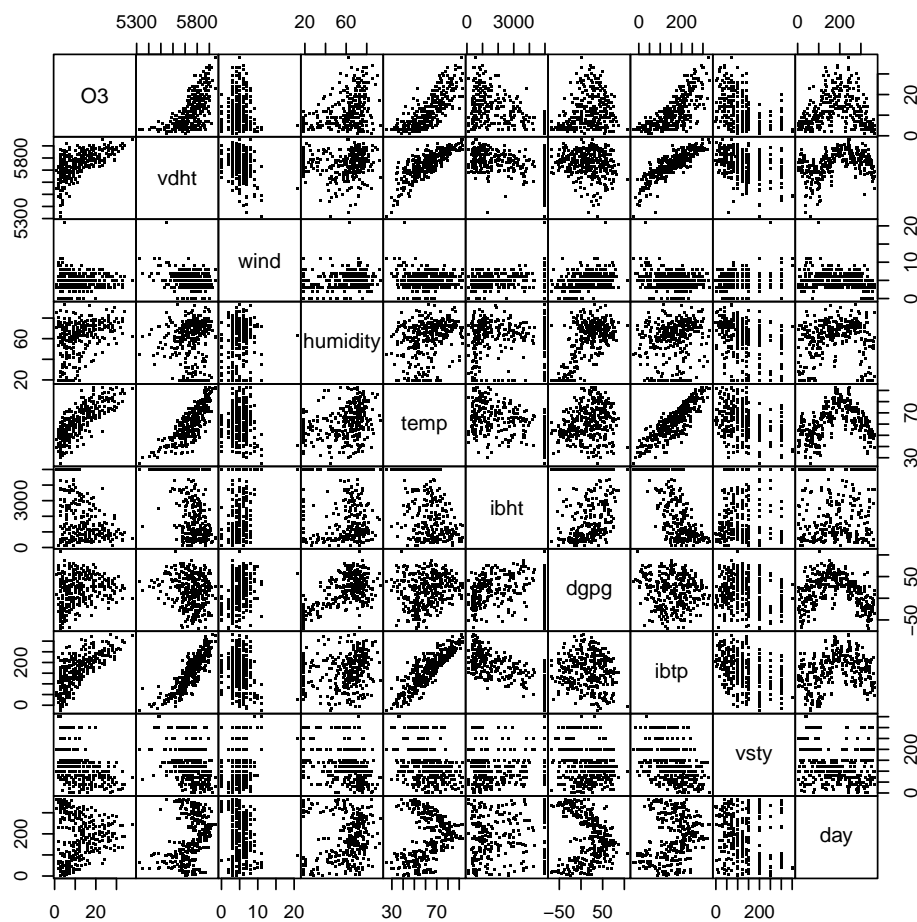


Figure 7.1: Daily ozone concentration in the Los Angeles basin as a function of 9 predictor variables.

```
> fitA <- gam(O3 ~ s(vdht)+ s(wind)+ s(humidity)+ s(temp)+
  s(ibht)+ s(dgpg)+ s(ibtp)+ s(vsty)+ s(day), data = d.ozone)

> summary(fitA)
```

```

.....
Parametric coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  11.7758     0.1988   59.22  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
              edf Ref.df      F p-value
s(vdht)      1.000  1.000  8.712 0.003407 **
s(wind)      1.000  1.000  4.271 0.039610 *
s(humidity)  3.631  4.517  2.763 0.018552 *
s(temp)      4.361  5.396  4.864 0.000189 ***
s(ibht)      3.043  3.725  1.292 0.358112
s(dgpg)      3.230  4.108 10.603 < 2e-16 ***
s(ibtp)      1.939  2.504  1.808 0.197390
s(vsty)      2.232  2.782  5.822 0.000947 ***
s(day)       4.021  5.055 15.817 < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.797   Deviance explained = 81.2%
GCV = 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.

```
> plot(fitA, pages = 1, shade = TRUE)
```

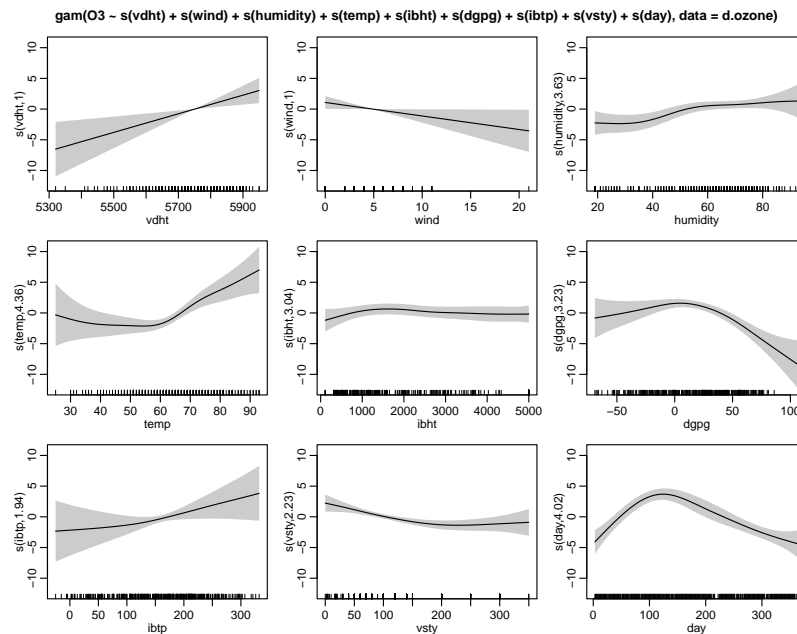


Figure 7.2: Estimated functions \hat{g}_j ($j = 1, 2, \dots, 9$) for the ozone data set.

We get an indication about the relevance of a predictor either from Figure 7.2 or from the P -values in the summary output from R.

Model fit can be checked via residual analysis as in chapter 1. The Tukey-Anscombe

```
> sfsmisc::TA.plot(fitA, labels = "o", cex.main = 0.7)
```

```
TA-pl: gam(O3 ~ s(vdht) + s(wind) + s(humidity) + s(temp) + s(ibht) + s(dgpg) + s(ibtp) + s(vsty) + s(day))
```

```
sfsmisc::TA.plot(lm.res = fitA, labels = "o", cex.main = 0.7)
```

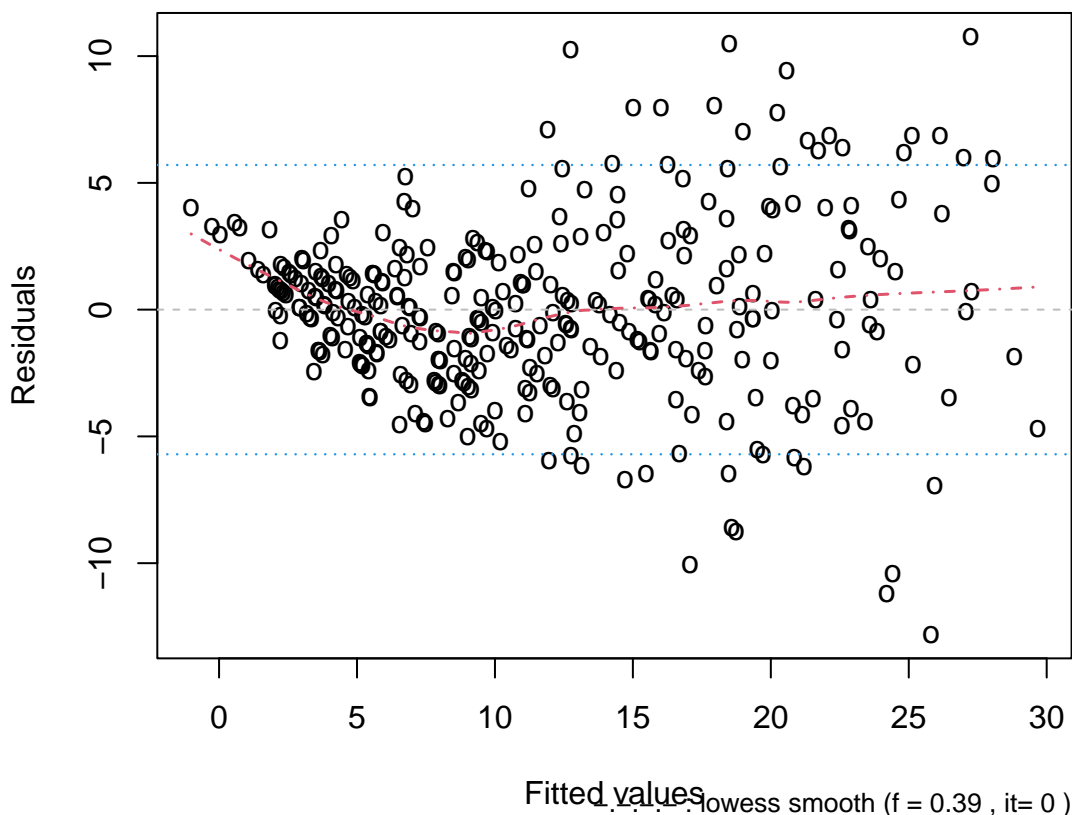


Figure 7.3: Tukey-Anscombe plot for the ozone data set.

plot indicates heteroscedastic errors. We thus try the log-transform for the response variable and re-do the whole analysis.

```
> fitA2 <- update(fitA, log(O3) ~ .)
```

Also, we omit observation (no. 92) which is an outlier wrt. wind speed. The results are given below.

```
> with(d.ozone, which(wind > 15))
```

```
[1] 92
```

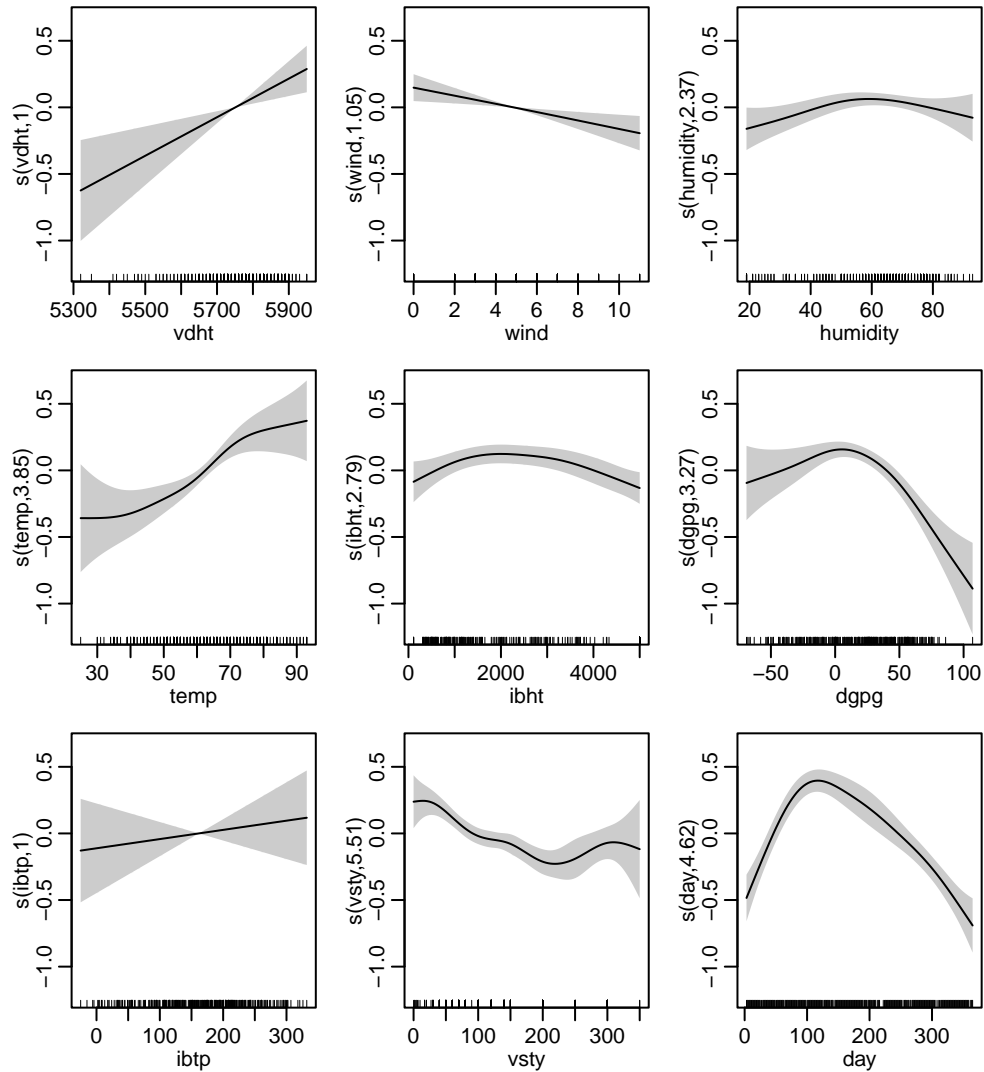
```
> fitA2.o <- update(fitA2, subset = -92)
```

```
> plot(fitA2.o, pages=1, shade = TRUE)
```

```
> sfsmisc::TA.plot(fitA2.o, labels="o", cex.main = 0.66)
```

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

`gam(log(O3) ~ s(vdht) + s(wind) + s(humidity) + s(temp) + s(ibht) + s(dgpg) + s(ibtp) + s(vsty) + s(day), data = d.ozone, subset = -92)`



TA-pl: `gam(log(O3) ~ s(vdht) + s(wind) + s(humidity) + s(temp) + s(ibht) + s(dgpg) + s(ibtp) + s(vsty) + s(d`

`sfsmisc::TA.plot(lm.res = fitA2.o, labels = "o", cex.main = 0.66)`

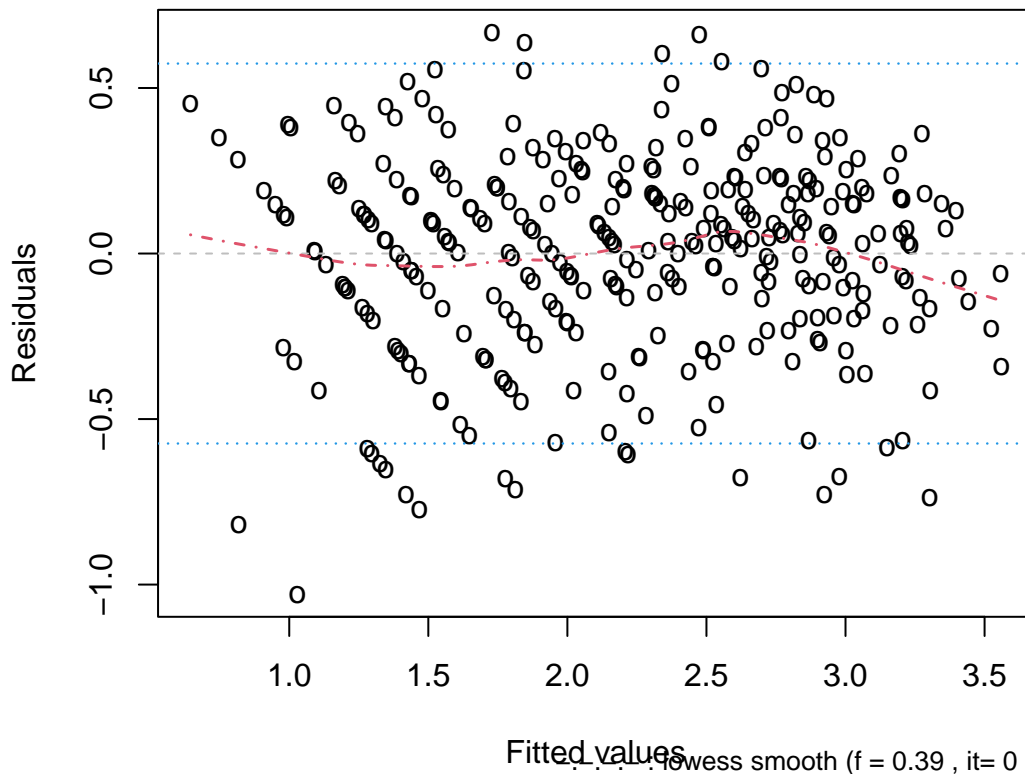


Figure 7.4: Estimated additive functions and Tukey-Anscombe plot for the ozone dataset with $\log(O_3)$ as response variable and after omitting one outlier.

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

$$(x_j - d)_+ = \begin{cases} x_j - d & \text{if } x_j > d, \\ 0 & \text{otherwise.} \end{cases} \quad (7.1)$$

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

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

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

$$g(\mathbf{x}) = \mu + \sum_{m=1}^M \beta_m h_m(\mathbf{x}) = \sum_{m=0}^M \beta_m h_m(\mathbf{x}),$$

where we set $\beta_0 := \mu$, $h_0(\mathbf{x}) \equiv 1$ and each other $h_m(\cdot)$ 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 $h_0(\mathbf{x}) \equiv 1$. 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

$$\begin{aligned} h_{2r-1}(\cdot) &= h_\ell(\cdot) \times (x_j - d)_+, \\ h_{2r}(\cdot) &= h_\ell(\cdot) \times (d - x_j)_+, \end{aligned} \quad (7.2)$$

for some h_ℓ in the model set \mathcal{M} which does *not* contain x_j ³ 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

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

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 (7.2).

³consequently, each $h_r(\cdot)$ will be a product of basis functions (in \mathcal{B}) but contain a predictor variable x_j at most once.

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:

$$\begin{aligned}
 h_0(\mathbf{x}) &= 1, \quad \mathcal{M} = \{1\}, \\
 h_1(\mathbf{x}) &= (x_2 - x_{7,2})_+, \quad h_2(\mathbf{x}) = (x_{7,2} - x_2)_+, \quad \mathcal{M} = \{1, (x_2 - x_{7,2})_+, (x_{7,2} - x_2)_+\}, \\
 h_3(\mathbf{x}) &= (x_{7,2} - x_2)_+ \cdot (x_1 - x_{5,1})_+, \quad h_4(\mathbf{x}) = (x_{7,2} - x_2)_+ \cdot (x_{5,1} - x_1)_+, \\
 &\quad \mathcal{M} = \{1, (x_2 - x_{7,2})_+, (x_{7,2} - x_2)_+, (x_{7,2} - x_2)_+ \cdot (x_1 - x_{5,1})_+, h_4(\mathbf{x})\} \\
 &\dots
 \end{aligned}$$

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 originally been implemented in S and R in function `mars` from package `mda`. A more flexible alternative is now provided by package `earth`'s function `earth()` and syntax

```

> require("earth")
> Mfit <- earth(Volume ~ ., data = trees)
> summary(Mfit)

              coefficients
(Intercept)    29.0599535
h(14.2-Girth)  -3.4198062
h(Girth-14.2)   6.2295143
h(Height-75)   0.5813644

Selected 4 of 5 terms, and 2 of 2 predictors
Termination condition: RSq changed by less than 0.001 at 5 terms
Importance: Girth, Height
Number of terms at each degree of interaction: 1 3 (additive model)
GCV 11.25439   RSS 209.1139   GRSq 0.959692   RSq 0.9742029

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

```

Volume
 [1,] -2.402263

 [11,] 36.950387

7.4 Neural Networks

Neural networks have been very popular in the 90's in the machine learning but have gained much increased popularity since about 2011 with so-called “deep learning”⁴. From a statistical perspective, they can be viewed as high-dimensional nonlinear regression models.

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

$$g_k(\mathbf{x}) = f_0\left(\alpha_k + \sum_{h=1}^q w_{hk}\phi\left(\tilde{\alpha}_h + \sum_{j=1}^p \tilde{w}_{jh}x_j\right)\right). \quad (7.3)$$

which is predicting multivariate $(g_k)_k$ where for regression only g_0 is used and for clas-

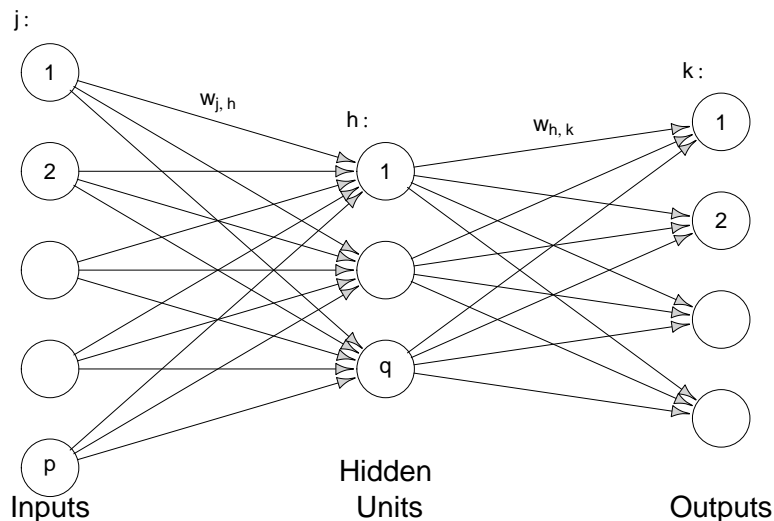


Figure 7.5: Simple “feedforward” neural net with one hidden layer.

sification, one typically uses g_0, \dots, g_{J-1} and $\hat{\mathcal{C}}(\mathbf{x}) := \arg \max_j \hat{g}_j(\mathbf{x})$. For estimation of conditional class probabilities, often a so-called softmax operation is used,

$$\hat{\pi}(Y = k | \mathbf{X} = \mathbf{x}) = \frac{\exp(g_k(\mathbf{x}))}{\sum_j \exp(g_j(\mathbf{x}))}.$$

Popular nonlinear activation functions $\phi(\cdot)$ include the sigmoid function

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

or the ‘Rectified Linear Unit’ (ReLU) activation function, the same as MARS basis (7.1),

$$\phi(t) = \max\{0, t\} = (t)_+ = (t - 0)_+.$$

⁴notably Google’s decisions to use the “tensorflow” library (<https://en.wikipedia.org/wiki/TensorFlow>) in many of its web products, and to open source it as of Nov. 2015.

For regression, the activation function $f_0(\cdot)$ in the output layer is typically chosen as the identity, and for classification, typically as the same nonlinearity $\phi(t)$ as in the hidden layer(s). The $w_{hk}, \tilde{w}_{jh}, \alpha_k, \tilde{\alpha}_h$ all are unknown parameters. The so-called input layer consists of the p predictor variables; the values $w_{hk} \phi\left(\tilde{\alpha}_h + \sum_{j=1}^p \tilde{w}_{jh} x_j\right)$ ($h = 1, \dots, q$) 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 (7.3) is a model which includes a linear regression component:

$$g(\mathbf{x}) = f_0 \left(\alpha + \sum_{j=1}^p w_{j,lin} x_j + \sum_{k=1}^q w_k \phi \left(\alpha_k + \sum_{j=1}^p w_{jk} x_j \right) \right). \quad (7.4)$$

These 'shortcut' connections that skip one (or more) layers appear prominently in so-called ResNets.

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 .

We use the `d.ozone` data from above, using `log(O3)` and scaling the x-variables (to mean zero and st.dev. one):

```
> sc.ozone <- data.frame(scale(subset(d.ozone, select = - O3)),
                          log.O3 = log(d.ozone[, "O3"]))# ^^^^
> library(nnet)
> set.seed(22) ## (also try others; randomness (!!), see below)

> fit <- nnet(log.O3 ~ . , data = sc.ozone, size = 3, decay = 4e-4,
              linout = TRUE, skip = TRUE, maxit = 500)

# weights: 43
initial value 2943.804833
iter 10 value 568.319707
.....
.....
iter 260 value 34.631819
final value 34.631808
converged

> sum(residuals(fit)^2)

[1] 34.21102

> 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
-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

```

Now, without a linear model component: skip=FALSE

```

> set.seed(22)
> Nfit <- nnet(log.O3 ~ . , data = sc.ozone,
              size = 3,          # chooses 3 hidden units, regularized by
              decay = 4e-4,     # "weight decay"
              linout = TRUE,    # linear regression scale (*not* classification)
              skip = FALSE,     # a N.N. with*OUT* a linear component
              maxit = 500)

# weights:  34
initial value 2702.113181
iter  10 value 128.630579
iter  20 value 63.825542
.....
.....
iter 350 value 43.737156
final value 43.737155
converged

> sum(residuals(Nfit)^2)

[1] 42.86296

> summary(Nfit)

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
-21.73 12.18 -4.03  7.86  3.70  3.70 -7.09 -8.94 -12.91  1.21
b->h2 i1->h2 i2->h2 i3->h2 i4->h2 i5->h2 i6->h2 i7->h2 i8->h2 i9->h2
 0.13 -0.19 -0.03  0.07  1.39 -0.36  0.05 -0.22 -0.03 -0.21
b->h3 i1->h3 i2->h3 i3->h3 i4->h3 i5->h3 i6->h3 i7->h3 i8->h3 i9->h3
-17.17  7.15  0.16  1.76 -20.96  1.05  6.25 17.61 -2.25 -2.81
b->o  h1->o  h2->o  h3->o
 0.86  0.41  2.42  0.86

```

Note that `size = 3` and `weight decay = 4e-4` are tuning parameters. The random seed is explicitly set, as by default, `nnet()` uses random starting values for the high-dimensional optimization.

7.4.2 Deeper networks

Let us define a multivariate version of the nonlinear activation function ϕ by applying the univariate function elementwise, so that for the rectified linear activation function and vector input $\mathbf{z} \in \mathbb{R}^p$ we get $\phi : \mathbb{R}^p \mapsto \mathbb{R}^p$ and

$$\phi(\mathbf{z})_k = \max\{0, \mathbf{z}_k\} = (\mathbf{z}_k)_+ \quad \text{for } k \in \{1, \dots, p\}.$$

We also assume that an intercept is always included in each layer to make notation simpler. Then we can build a network with d layers as

$$g(\mathbf{x}) = \phi(W_{d,d-1}\phi(W_{d-1,d-2}\phi(\dots\phi(W_{2,1}\mathbf{x})))).$$

The choice of the number d of layers and dimensions (p_1, p_2, \dots, p_d) of hidden layers is called the “architecture” and a (meta-)parameter one needs to search over generally (using cross-validation for example), along with “training of” / estimating the weight matrices $W_{2,1}, W_{3,2}, \dots, W_{d,d-1}$. For larger networks (millions to billions of weights and sample size in the millions), using R directly is not computationally efficient, not least because training needs to happen on a GPU. Often, packages like TensorFlow or pytorch are used (and glued together with python code). A nice alternative is to do training in Julia because all training can happen within Julia, using packages like for example Flux (<https://fluxml.ai/>). For some stand-alone examples see for example <https://github.com/FluxML/model-zoo>.

Deep networks are not often built as such combinations of so-called dense layers, where the weight matrices are non-sparse. Rather, often some special structure of the input is used to build more specialized architectures. If such a structure (such as spatial for images, time for audio or text) can be used, then these are often applications where deep networks are much more accurate than say linear models or trees, given sufficient large training size. Some special architectures include

1. Convolutional neural networks. These are well suited for image classification. The weights in the first hidden layers are just non-zero for a small part of an image (so-called patch) and weights are identical for the patches that are identical modulo translation.
2. Recurrent neural networks. Well suited for input with a time-structure such as text or classical time-series data. The hidden layers at time t connect also to the hidden layers at time $t + 1$.
3. Auto-encoders. Here the goal is to build a so-called encoder e and decoder d such that for a p -dimensional input x , the encoder maps to a lower-dimensional space, $e : \mathbb{R}^p \mapsto \mathbb{R}^q$ with $q \ll p$ and the decoder d maps back to the original space $d : \mathbb{R}^q \mapsto \mathbb{R}^p$. The reconstruction error of x can then for example be measured by squared error loss $\|x - d(e(x))\|_2^2$ and we try to optimize the parameters for e and d such that the expected mean squared error is as small as possible. Question: what is the solution if we constrain e and d to be linear functions?
4. Many more interesting applications such as generative adversarial networks (with connection to density estimation), word embeddings, speech recognition, translation etc.

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 (7.3), we use the following model:

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

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

The functions $f_k(\cdot) : \mathbb{R} \rightarrow \mathbb{R}$ are nonparametric (i.e. “arbitrary” smooth); the linear combinations $\sum_{j=1}^p \alpha_{jk}x_j$ are linear projections: For the unit vector $\boldsymbol{\alpha}_k = (\alpha_{1k}, \dots, \alpha_{pk})^\top$, $\boldsymbol{\alpha}_k^\top x$ 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

$$g_{\text{PPR}}(\mathbf{x}) = \mu + \sum_{k=1}^q \beta_k f_k\left(\sum_{j=1}^p \alpha_{jk}x_j\right),$$

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

Consider the ozone data set from above.

```
> fit <- ppr(log(O3) ~ . , data = d.ozone, nterms = 4) # nterms specifies
>                                     ##         the number of ridge functions
> sfsmisc::mult.fig(4, marP = c(-1,-1,-1, 0)) # or just par(mfrow=c(2,2))
> plot(fit, cex = 1/4) ## 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

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

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

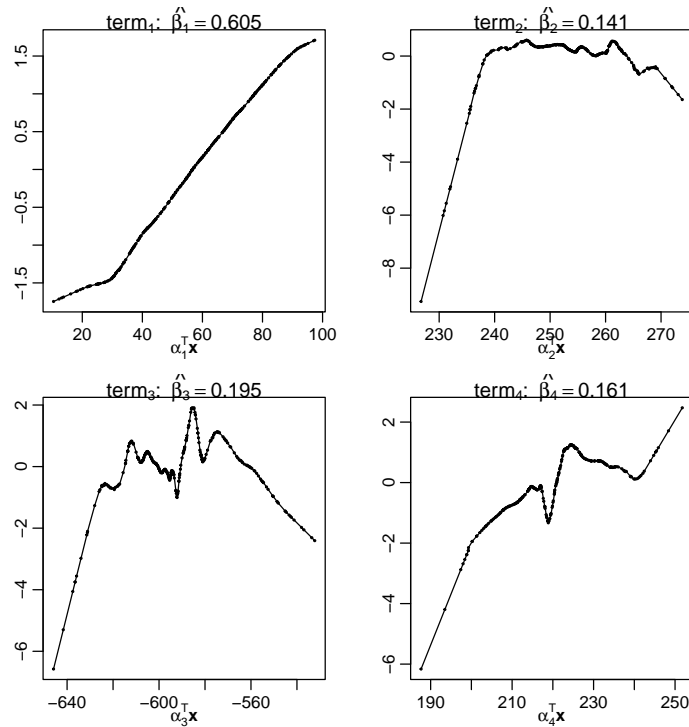


Figure 7.6: Four estimated ridge functions $f_k(\cdot)$ from a projection pursuit fit to the ozone data ($\log(\text{O3}) \sim \cdot$).

7.6.1 Tree structured estimation and tree representation

Parameter estimation $\hat{\beta}_1, \dots, \hat{\beta}_M$ is easy when the partition $\mathcal{P} = \{\mathcal{R}_1, \dots, \mathcal{R}_M\}$ is given. We use

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

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, e.g,

$$\hat{\pi}_j(\mathbf{x}) = \frac{\sum_{i=1}^n \mathbf{1}_{[Y_i=j]} \mathbf{1}_{[\mathbf{x}_i \in \mathcal{R}_r]}}{\sum_{i=1}^n \mathbf{1}_{[\mathbf{x}_i \in \mathcal{R}_r]}} \quad \text{for } \mathbf{x} \in \mathcal{R}_r,$$

and do not need to go with a one against the rest approach as in 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 $M = 1$ subset, $\mathcal{P} = \{\mathcal{R}\} = \{\mathbb{R}^p\}$.

2. Refine \mathcal{R} into $\mathcal{R}_{left} \cup \mathcal{R}_{right}$ where:

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

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 $d \in \{\text{mid-points of observed values}\}$). 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, $M = M_{max}$, of partition cells.

5. Backward deletion: prune the tree (see below) until a reasonable model size, typically determined via cross-validation, is achieved.

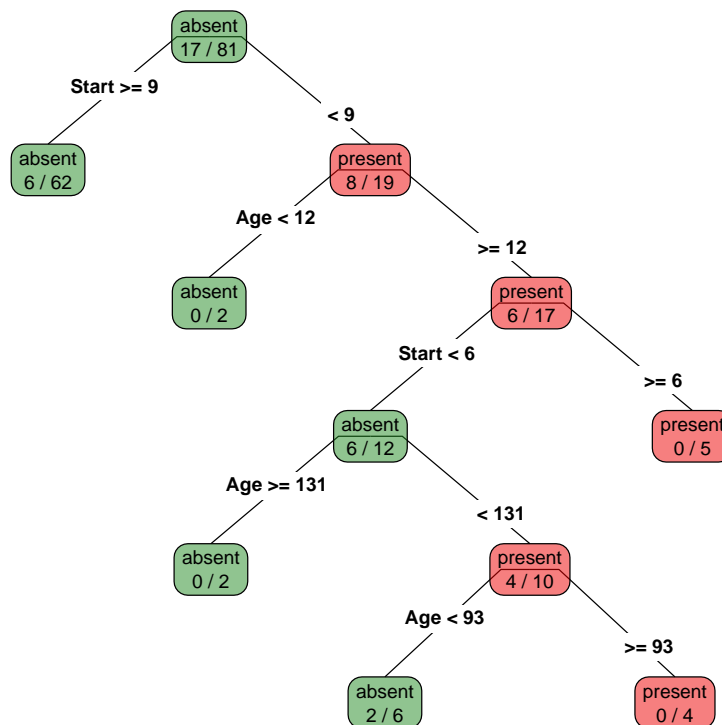


Figure 7.7: Classification tree for the kyphosis data set. Each node is labeled “present” or “absent” (of kyphosis) according to majority vote, i.e., maximum likelihood. The ratios $a/(a + b)$ are empirical misclassification rates, i.e., in this binary case, $b = \#\{\text{cases with given label}\}$, and a counts the other, “wrongly” labeled ones.

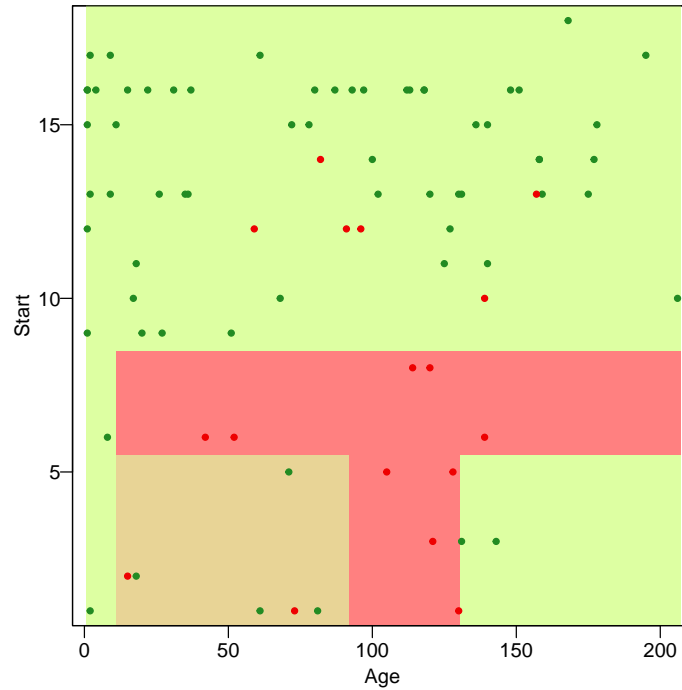


Figure 7.8: Partition of \mathbb{R}^2 for the same kyphosis data `rpart()` fit as in Fig. 7.7. Majority voting in each rectangle then determines the classification rule (indicated by color here).

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 (using `plotmo::plotmo()`, see R script `rpart-Ex.R`).

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 ($M = M_{max}$). 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 $R(\cdot)$). This will produce a sequence of trees

$$\mathcal{T}_M \supset \mathcal{T}_{M-1} \supset \dots \supset \mathcal{T}_1 = \mathcal{T}_0 := \{\mathcal{R}_0\}, \quad \mathcal{R}_0 = \text{root tree} = \mathbb{R}^p, \quad (7.5)$$

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,

$$R_\alpha(\mathcal{T}) := R(\mathcal{T}) + \alpha \times \text{size}(\mathcal{T}), \quad \alpha \geq 0 \quad (7.6)$$

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

Now pruning is done such that for (conceptually) each α an optimally pruned tree is chosen, $\mathcal{T}(\alpha) := \arg \min_{\mathcal{T} \subset \mathcal{T}_M} R_\alpha(\mathcal{T})$. 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 (7.5). For model selection, i.e., determining the amount of pruning, we now only need to choose the best α (or its

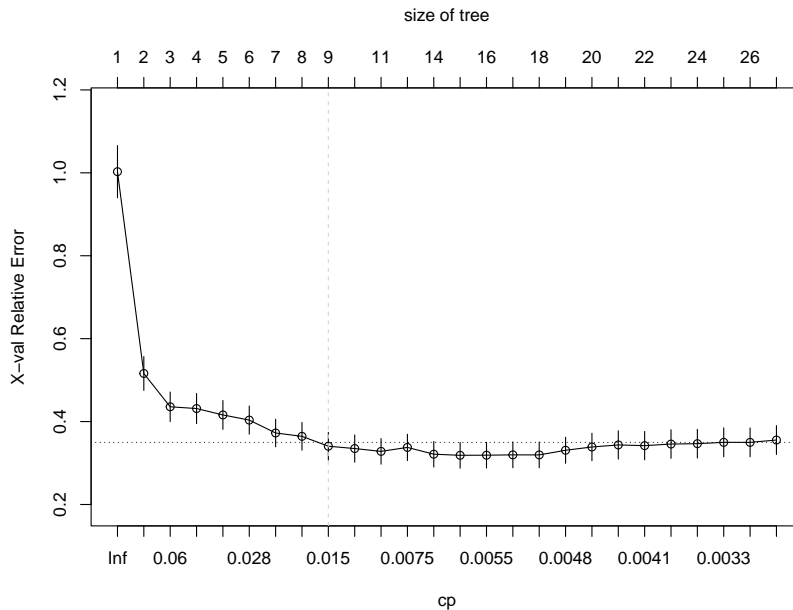


Figure 7.9: a `plotcp(.)` for the ozone example with a *larger* tree (using `cp = .003`, `min-split = 7!`) of `rpart(.)`-models as function of α or `cp = $\alpha/R(\mathcal{T}_\theta)$` , respectively. Whereas the minimum is at size 15 (and 16, 17, 18 are visually very close), a tree with 9 leaves (final nodes) is optimal according to the “1 SE rule”.

normalization, `cp = $\alpha/R(\mathcal{T}_\theta)$` 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 α . Instead of directly using the minimal error rate to choose α , 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.

7.6.3 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.

```
> f.rp <- rpart(log(O3) ~ . , data = d.ozone) ; f.rp
```

```
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
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 *

> require(rpart.plot) # for nice tree graphs :
> prp(f.rp, type = 3, extra = 1)

```

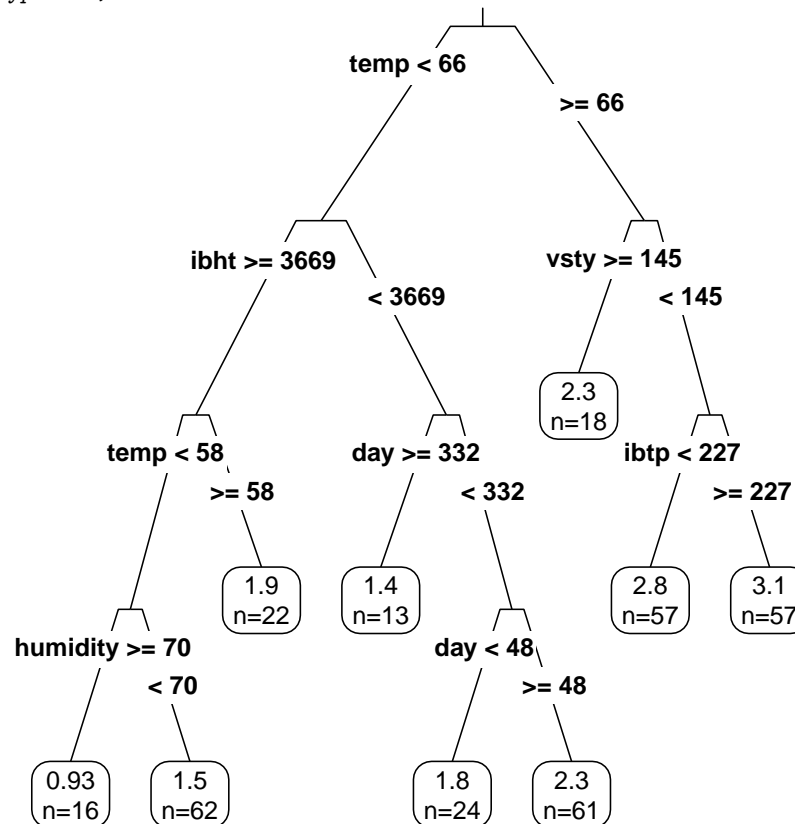


Figure 7.10: Regression tree for the ozone data set.

7.6.4 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.

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”.

7.6.5 “RandomForest” for powerful stable prediction

The main ingredient ideas of “random forests” prediction are

1. trees
2. bootstrap (ch. 5)
3. aggregation (→ bagging, ensemble learning, see chapter 8)

The “random forests” or “RandomForest” algorithm (for both classification and regression) consists of the following steps:⁵

1. Draw n_{tree} bootstrap samples from the original data.
2. For each of the bootstrap samples, grow an *unpruned* classification or regression tree, with the following modification: at each node, rather than choosing the best split among all predictors, randomly sample m_{try} of the predictors and choose the best split from among those variables.
3. Predict new data by *aggregating* the predictions of the n_{tree} trees (i.e., majority votes for classification, average for regression).

Note that the usual “Bagging” (:= “Bootstrap aggregating”) can be thought of as the special case of random forests obtained when $m_{\text{try}} = p$, the number of predictors.

An estimate of the error rate can be obtained, based on the training data, by the following:

1. At each bootstrap iteration, predict the data not in the bootstrap sample (what Breiman calls “out-of-bag”, or **OOB**, data) using the tree grown with the bootstrap sample.
2. Aggregate the OOB predictions. (On the average, each data point would be out-of-bag around 36% of the times, so aggregate these predictions.) Calculate the error rate, and call it the *OOB estimate of error rate*.

Our experience has been that the OOB estimate of error rate is quite accurate, given that enough trees have been grown (otherwise the OOB estimate can bias upward; see Bylander(2002).

Extra information from Random Forests in R :

The R package `randomForest` optionally produces two additional pieces of information:

1. a measure of the importance of the predictor variables
2. a measure of the internal structure of the data (the proximity of different data points to one another).

With `randomForest` and related R packages such as `randomSurvivalForest`, `varSelRF` etc, many trees are fit to the data and their variable selection and predictions combined into a final model.

These often lead to very good prediction with the drawback of losing an *interpretable* model.

⁵adapted from *RNews (3) 2002, p.18*

RF as Adaptive kernel

Random Forests can be seen to produce a kernel estimate, where the kernel adapts to the data. Assume each of the B trees in the forest is grown to contain only a single sample of the original training data. The prediction for a new sample $x \in \mathbb{R}^p$ is then

$$\hat{Y}(x) = \frac{1}{B} \sum_{b=1}^B \hat{Y}_b(x)$$

where $\hat{Y}_b(x)$ is the prediction of tree b for x and $\hat{Y}_b(x) = \sum_{r=1}^M \hat{\beta}_r^b \mathbb{1}\{x \in R_r^b\}$ and R_r^b are the subspace of \mathbb{R}^p defined by leaf node r in tree b . Furthermore, $\hat{\beta}_r^b = \sum_{i=1}^n Y_i \mathbb{1}\{x_i \in R_r^b\}$ as only a single observation is left in each node. Hence

$$\begin{aligned} \hat{Y}(x) &= \frac{1}{B} \sum_{b=1}^B \sum_{r=1}^M \sum_{i=1}^n Y_i \mathbb{1}\{x_i \in R_r^b\} \mathbb{1}\{x \in R_r^b\} \\ &= \sum_{i=1}^n w_i Y_i, \end{aligned}$$

where the weights $w_i \in [0, 1]$, $i = 1, \dots, n$ with $\sum_i w_i = 1$ are given by

$$w_i = P(x_i, x \text{ in same leaf node of randomly chosen tree}).$$

As one consequence, the predictions will always be in the convex hull of the original observations Y_i , $i = 1, \dots, n$. If the trees split only on a given subset of variables, then the kernel “adapts”. By ignoring all other dimensions/variables, the problem is locally reduced again to a lower-dimensional problem, thus being potentially able to avoid the curse of dimensionality.

7.7 Variable Selection, Regularization, Ridge 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) $Y_i = \mathbf{x}_i^\top \boldsymbol{\beta} + \varepsilon_i$, ($i = 1, \dots, n$), equivalently

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \quad (7.7)$$

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 $X^\top X$ 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 $p \gg n$. One of the most important of such applications is the analysis of genomic data, where e.g., a so called “microarray” contains gene expression levels for thousands of genes ($p = 1000\text{--}50'000$) for typically a few dozen cells ($n = 20\text{--}100$). 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 (7.7), or

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

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

$$Y_i = \beta'_0 + \beta_1(x_{i,1} - \bar{x}_{.,1}) + \dots + \beta_p(x_{i,p} - \bar{x}_{.,p}) + \varepsilon_i.$$

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

$$Y_i = \beta_1 x_{i,1} + \dots + \beta_p x_{i,p} + \varepsilon_i, \quad i = 1, \dots, n. \quad (7.8)$$

Note that in order to have β_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,

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

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 β_j - β_k -plane the high confidence region forms narrow ellipses, i.e., the β 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 $\mathbf{X}^\top \mathbf{X}$ is close to a singular matrix.⁶ One approach to this problem is to *regularize* it by improving the condition of the matrix corresponding to $\mathbf{X}^\top \mathbf{X}$.

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*

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

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

$$\hat{\boldsymbol{\beta}}^*(\lambda) = \arg \min_{\boldsymbol{\beta}} \{ \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta}\|^2 \}, \quad (7.9)$$

where there is a one-to-one relationship between λ and the bound s above, and the limit $\lambda \rightarrow 0$ corresponds to $s = \max \|\boldsymbol{\beta}\|^2 \rightarrow \|\hat{\boldsymbol{\beta}}^{LS}\|^2$, i.e., the ordinary least squares of chapter 1.

⁶such that $\text{Cov}(\hat{\boldsymbol{\beta}}) = \sigma^2(\mathbf{X}^\top \mathbf{X})^{-1}$ (section 1.4.1) will have very large entries corresponding to the high variance (and correlation) of $\hat{\beta}_j$ and $\hat{\beta}_k$.

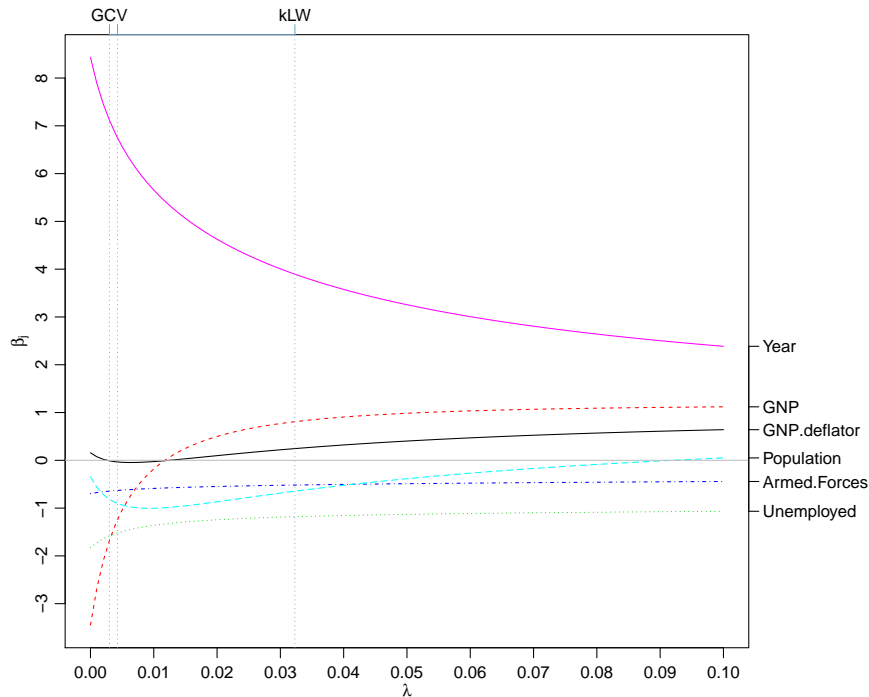


Figure 7.11: Ridge traces for the six coefficients $\beta_j(\lambda)$ ($j = 1, \dots, 6$) for the Longley data. The vertical lines indicate traditional estimates of the optimal ridge parameter λ .

As there, by setting derivatives $\partial/\partial\beta$ to zero, this minimization problem is equivalent to the “normal equations”

$$(X^T X + \lambda I) \hat{\beta}^* = X^T \mathbf{Y}, \quad (7.10)$$

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

The ridge penalty entails that $\hat{\beta}_j(\lambda) \rightarrow 0$ (“shrinking”) when $\lambda \rightarrow \infty$, and also, typically, $\hat{\beta}_j \rightarrow \hat{\beta}_j'$ (“shrinking together”) for two different coefficients.

Hence it seems intuitive that $\hat{\beta}^*$ will have some bias ($\mathbb{E}[\hat{\beta}^*] \neq \beta$), but that its variance(s) can be considerably smaller than $\text{Var} \hat{\beta}^{LS}$ such that mean squared errors are smaller. As for smoothing (in particular, spline smoothing, sect. 3.4) we have a regularization parameter λ 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 λ .

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 λ . In practice, one often wants to look at the *ridge traces*, i.e., a plot of the coefficients $\hat{\beta}_j(\lambda)$ vs λ . As an example we consider Longley’s macro economical data, for once modelling $y = \text{GNP.deflator}$ 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 λ 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.,

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

or

$$\begin{aligned} \hat{\boldsymbol{\beta}}^*(\lambda) &= \arg \min_{\boldsymbol{\beta}} \left\{ \|\mathbf{Y} - X\boldsymbol{\beta}\|^2 + \lambda \sum_{j=1}^p |\beta_j| \right\}, \\ &= \arg \min_{\boldsymbol{\beta}} \left\{ \|\mathbf{Y} - X\boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta}\|_1 \right\}. \end{aligned} \quad (7.11)$$

For two predictors, $\boldsymbol{\beta} \in \mathbb{R}^2$, the lasso and ridge problems are depicted *constrained* least squares in Fig. 7.12

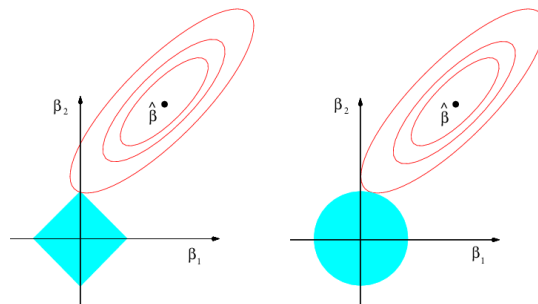


Figure 7.12: Lasso and ridge regression for 2-dimensional $\boldsymbol{\beta}$, as *constrained* least squares problem (from ISLR (2013), Fig.6.7, with permission).

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 λ 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”, Fig. 7.13.

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` and now `glmnet`, too. For any fixed non-negative λ_1 and λ_2 , we define the naïve elastic net criterion

$$L(\lambda_1, \lambda_2, \boldsymbol{\beta}) = \|\mathbf{Y} - X\boldsymbol{\beta}\|^2 + \lambda_2 \|\boldsymbol{\beta}\|^2 + \lambda_1 \|\boldsymbol{\beta}\|_1, \quad (7.12)$$

and the naïve elastic net estimator $\hat{\boldsymbol{\beta}}$ is the minimizer of (7.12), $\hat{\boldsymbol{\beta}} = \arg \min_{\boldsymbol{\beta}} L(\lambda_1, \lambda_2, \boldsymbol{\beta})$.

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 (7.12) is equivalent to the optimization problem

$$\hat{\boldsymbol{\beta}} = \arg \min_{\boldsymbol{\beta}} \|\mathbf{Y} - X\boldsymbol{\beta}\|^2, \quad \text{subject to } (1 - \alpha)\|\boldsymbol{\beta}\|_1 + \alpha \|\boldsymbol{\beta}\|^2 \leq t \text{ for some } t. \quad (7.13)$$

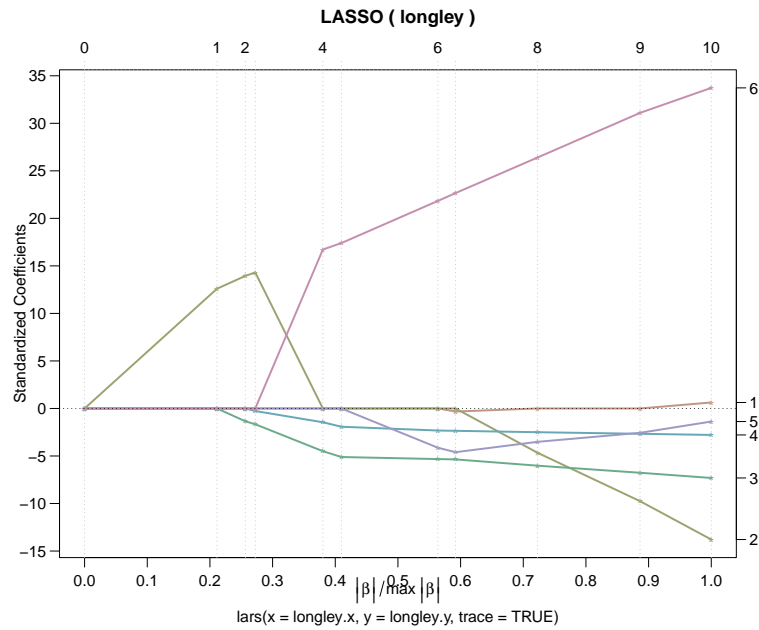


Figure 7.13: Lasso traces for the six coefficients $\beta_j(\lambda)$ ($j = 1, \dots, 6$) for the Longley data.

We call the function $(1 - \alpha)\|\beta\|_1 + \alpha\|\beta\|^2$ 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. 7.14.

2) The Adaptive Lasso and its Oracle Properties

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

$$\arg \min_{\beta} \left\| \mathbf{Y} - \sum_{j=1}^p \beta_j \mathbf{x}_j \right\|^2 + \lambda \cdot \sum_{j=1}^p w_j |\beta_j|. \quad (7.14)$$

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 $\hat{\beta}$ is a \sqrt{n} consistent estimator of β . For example, we can use the least squares $\widehat{\beta}_{ols}$. Pick a $\gamma > 0$, and define the weight vector $\hat{\mathbf{w}}$ componentwise as $\hat{w}_j = \frac{1}{|\hat{\beta}_j|^\gamma}$. The *adaptive lasso estimates* $\widehat{\beta}_{(n)}^*$ are then given by

$$\widehat{\beta}_{(n)}^* = \arg \min_{\beta} \left\| \mathbf{Y} - \sum_{j=1}^p \beta_j \mathbf{x}_j \right\|^2 + \lambda \cdot \sum_{j=1}^p \hat{w}_j |\beta_j|. \quad (7.15)$$

It is worth emphasizing that (7.15) is a convex optimization problem, and thus has no multiple local minimal and its global minimizer can be efficiently found.

Note that “the major” lasso R package’s `glmnet`’s `glmnet()` now has an optional argument `penalty.factor` which allows to set the vector of weights $(w_j)_{j=1, \dots, n}$, and there has been `adalasso()` in R package `parcor`.

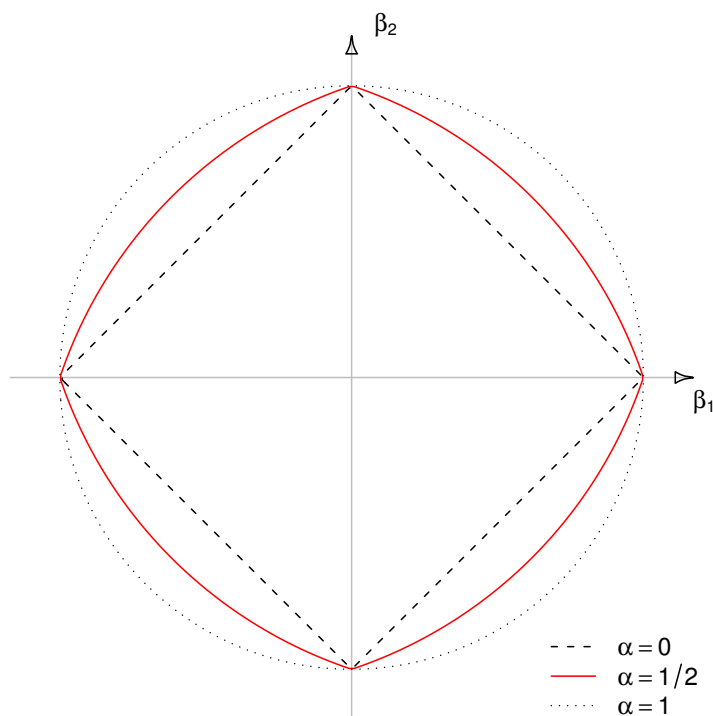


Figure 7.14: Two-dimensional contours of the ridge, lasso, and elastic net ($\alpha = \frac{1}{2}$) penalties.

3) The “Relaxed Lasso”

Developed by Nicolai Meinshausen in 2006, CSDA **52**, 374–393 (<https://doi.org/10.1016/j.csda.2006.12.019>) 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 $\widehat{\boldsymbol{\beta}}^*(\lambda) = (\widehat{\beta}_1^\lambda, \dots, \widehat{\beta}_p^\lambda)^\top$ be the lasso estimated parameter vector of (7.11), and define

$$\mathcal{M}_\lambda = \{1 \leq k \leq p \mid \widehat{\beta}_k^\lambda \neq 0\}, \quad (7.16)$$

the set of “significant” variables.

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

$$\widehat{\boldsymbol{\beta}}^{\lambda, \phi} = \arg \min_{\boldsymbol{\beta}} n^{-1} \sum_{i=1}^n (Y_i - \sum_{k \in \mathcal{M}_\lambda} \beta_k x_{i,k})^2 + \phi \lambda \cdot \|\boldsymbol{\beta}\|_1. \quad (7.17)$$

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) (ϕ, λ) combinations which is only slightly more expensive than the fast lasso algorithm.

4) The (sparse) group lasso

The lasso, and the lasso extensions mentioned till now all have the drawback that variable selection fails in a way when the predictor variables are *categorical* with more than two categories, as such variables are coded as more than one “dummy” variables —typically,

but not necessarily 0–1 variables— which are not treated as a “group”, but rather selected or deselected independently.

The group lasso and its generalization, the sparse group lasso, address this problem by allowing to consider groups of predictor variables. As previously, we have a $n \times p$ “design” matrix \mathbf{X} and p -dimensional parameter vector $\boldsymbol{\beta}$. We assume now that these p predictors are divided into L groups, of size p_l each, hence $\sum_{l=1}^L p_l = p$, the design matrix \mathbf{X} is made of L blocks of columns \mathbf{X}_l and the $\boldsymbol{\beta}$ vector is partitioned into subvectors $\boldsymbol{\beta}_l$, such that

$$\mathbf{X}\boldsymbol{\beta} = \mathbf{X}_1\boldsymbol{\beta}_1 + \dots + \mathbf{X}_L\boldsymbol{\beta}_L = \sum_{l=1}^L \mathbf{X}_l\boldsymbol{\beta}_l.$$

Yuan & Lin (2007) proposed the group lasso as minimizer of a modified, convex, lasso penalty,

$$\hat{\boldsymbol{\beta}}^g(\lambda) = \arg \min_{\boldsymbol{\beta} \in \mathbb{R}^p} \left\| \mathbf{Y} - \sum_{l=1}^L \mathbf{X}_l\boldsymbol{\beta}_l \right\|_2^2 + \lambda \cdot \sum_{l=1}^L \sqrt{p_l} \|\boldsymbol{\beta}_l\|_2, \quad (7.18)$$

where $\|\cdot\|_2$ is the euclidean norm (*not squared*). If all the groups are of size 1 (and hence $L = p$), this reduces to the lasso, as $\|\boldsymbol{\beta}_l\|_2 = |\beta_l|$ when $\boldsymbol{\beta}_l$ is one-dimensional.

The group lasso acts like the lasso “at the group level”: an entire group of β 's shrinks to zero simultaneously, i.e., for the same value of λ , and thus the entire corresponding group predictors drops out of the model simultaneously.

Meier et al.(2008) extend this to logistic regression, and to further problems subsequently, implemented in the R package `grplasso`.

Hastie et al.(2010) propose a generalization, the *sparse group lasso*,

$$\hat{\boldsymbol{\beta}}^g(\lambda_1, \lambda_2) = \arg \min_{\boldsymbol{\beta} \in \mathbb{R}^p} \left\| \mathbf{Y} - \sum_{l=1}^L \mathbf{X}_l\boldsymbol{\beta}_l \right\|_2^2 + \lambda_1 \cdot \sum_{l=1}^L \sqrt{p_l} \|\boldsymbol{\beta}_l\|_2 + \lambda_2 \|\boldsymbol{\beta}\|_1, \quad (7.19)$$

which applies “the elastic net” idea (7.12) to the group lasso, and also brings sparsity to the β 's *within* each group.

Lasso – sparse modelling

In June 1, 2017, there were 79 R packages on CRAN (out of 10'730) all with the word “LASSO” (or “lasso”, “Lasso”, ...) in their title. On May 27, 2021, using package name, title and description, 217 from 20'788 CRAN and Bioconductor packages matched an upper or lower case version of “lasso”;

and there are more R packages using lasso and its techniques, e.g., `hdi` from Sfs, ETHZ, which do not have “lasso” in their titles or descriptions.

Sparse high-dimensional modelling has been a hot topic for the last 20 years and will probably remain important in the age of big data.

Chapter 8

Bagging and Boosting

8.1 Introduction

Bootstrap **aggregating** (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, notably in “Random Forests”.

8.2 Bagging

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

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

(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

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

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

2. Repeat step 1 B times, yielding

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

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 $\hat{g}(\cdot)$,

$$\text{Var}(\hat{g}_{Bag}(x)) \stackrel{\text{asympt.}}{<} \text{Var}(\hat{g}(x)),$$

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 $\mathbf{1}_{[\cdot \leq d]}$. This causes a variance reduction at the price of some bias. For example,

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

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

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

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.

Another way to view the variance reduction for regression for regression trees is as follows. Assume a regression tree is fit such that there are exactly m training samples in each leaf node. The prediction for a new sample x is then

$$\hat{Y}(x) = \sum_{i=1}^n w_i Y_i$$

with weights

$$w_i = \frac{1}{m} \mathbf{1}\{x_i \text{ and } x \text{ in same leaf node}\}.$$

Bagging such a tree B times yields predictions

$$\hat{Y}_{bag}(x) = \sum_{i=1}^n \left(\frac{1}{B} \sum_{b=1}^B w_i^{*,b} \right) Y_i$$

where the weights $w_i^{*,b} \in \{0, \frac{1}{m}\}$ are the weights corresponding to the tree for the b th bootstrap sample. Hence

$$\hat{Y}_{bag}(x) = \sum_{i=1}^n w_{bag,i} Y_i$$

with $w_{bag,i} = \frac{1}{B} \sum_{b=1}^B w_i^{*,b}$. Again $\sum_{i=1}^n w_{bag,i} = 1$ and $w_{bag,i} \in [0, \frac{1}{m}]$. Hence, due to convexity of the ℓ_2 -norm

$$\sum_{i=1}^n w_{bag,i}^2 \leq \sum_{i=1}^n w_i^2.$$

This means the variance of the bagged trees is upper-bounded by the variance of the original tree under homoscedastic noise where $Y_i = E(Y|X = x_i) + \varepsilon_i$ with $E(\varepsilon_i^2) = \sigma^2$ as

$$\text{Var}(\hat{Y}(x)) = \sigma^2 \sum_{i=1}^m w_i^2,$$

and

$$\text{Var}(\hat{Y}_{bag}(x)) = \sigma^2 \sum_{i=1}^m w_{bag,i}^2 \leq \text{Var}(\hat{Y}(x)).$$

Note that the argument above assumes that the tree structures are fixed and the variance stems only from the variance in the training observations (ie tree structure is fitted with half of training samples and leaf values estimated with second half for example).

8.2.3 Subbagging

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

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

for some $m < n$. In some simple cases, it can be shown that $m = \lceil n/2 \rceil$ is equivalent to bagging. Thus, subbagging with $m = \lceil n/2 \rceil$ 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.

8.3 Boosting

Boosting (or “Gradient Boosting” and variants) is a very different method to generate multiple predictions (function estimates) and combine them linearly.

The first famous boosting algorithm has been Freund and Schapire (1999)’s “AdaBoost”, see, e.g., <https://en.wikipedia.org/wiki/AdaBoost>.

As with bagging, we have a base procedure (“base learner”) 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 $\{(X_i, Y_i); i = 1, \dots, n\}$ yielding a first function estimate $\hat{g}_1(\cdot)$.
Compute (“current”) residuals

$$U_i = Y_i - \hat{f}_1(X_i) \quad (i = 1, \dots, n),$$

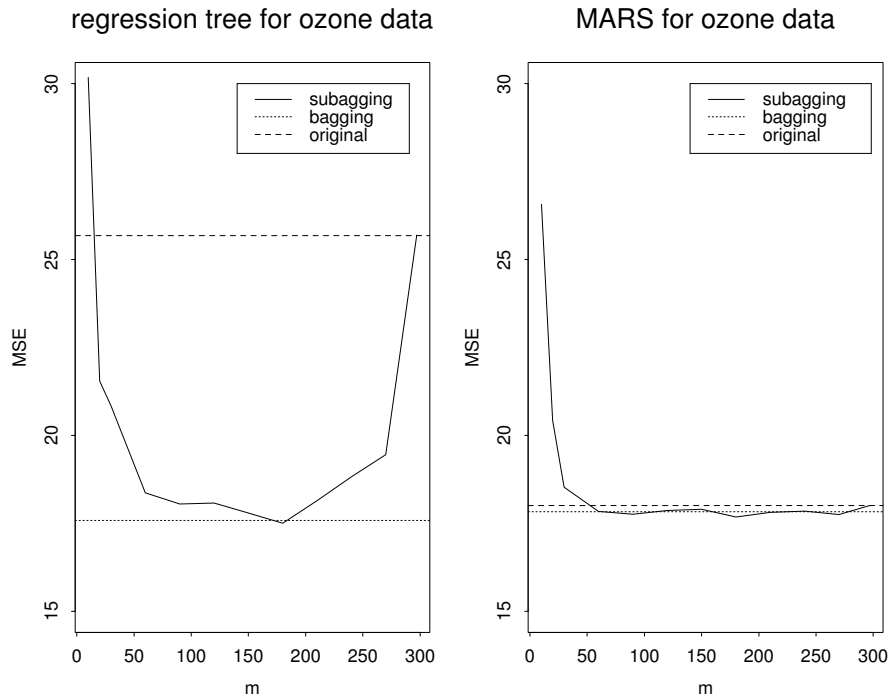


Figure 8.1: Mean squared error performance for a large regression tree and MARS and their (su-)bagged versions for an ozone data (different from the previous one).

where $\hat{f}_1(\cdot)$ is $\hat{g}_1(\cdot)$ or shrunken version of it, $\hat{f}_1(\cdot) = \nu \cdot \hat{g}_1(\cdot)$, where $0 < \nu \leq 1$.

2. For $m = 2, 3, \dots, M$ do:
Fit the residuals U_i (instead of the Y_i)

$$(X_i, U_i) \rightarrow \hat{g}_m(\cdot)$$

and set

$$\hat{f}_m(\cdot) = \hat{f}_{m-1}(\cdot) + \nu \hat{g}_m(\cdot).$$

Update the current residuals

$$U_i = Y_i - \hat{f}_m(X_i) \quad (i = 1, \dots, n).$$

The shrinkage parameter ν can be chosen to be small, e.g. $\nu = 0.1$. The stopping parameter M is typically the main tuning parameter of boosting. For ν 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.

There are 64 CRAN / Bioconductor R packages with the term “boosting” (as of May 2021); some “prominent” ones are

mboost “Model-Based Boosting” NA providing parametric and non-parametric, often well *interpretable* boosting model fits.

`xgboost` “eXtreme Gradient Boosting”, “an efficient implementation of the gradient boosting framework from Chen & Guestrin (2016)”. Also available outside R ; Wikipedia page <https://en.wikipedia.org/wiki/XGBoost>.

`gbm` “Generalized Boosted Regression Models”. Extensions to ‘Adaboost’ and Friedman’s gradient boosting machine. Regression methods for many loss functions (least squares (L_2), absolute (L_1), t-distrib., quantile regr., logistic, Poisson, Cox proportional hazards, partial likelihood, . . . etc.