Prediction and Feature Assessment

Nicolas Städler

2022-09-20

Contents

Chapter 1

Introduction

This script was written for the course on *Analysis of High-Dimensional Data* of the CAS in Advanced Statistical Data Science (CAS ASDS) held at the University of Bern. Much of the content is based on the book from Hastie et al. (2001). The course has a focus on applications using **R** (R Core Team, 2022). All data sets used throughout the script can be downloaded from github.

What are high-dimensional data and what is high-dimensional statistics? The Statistics Department of the University of California, Berkeley summarizes it as follows:

High-dimensional statistics focuses on data sets in which the number of features is of comparable size, or larger than the number of observations. Data sets of this type present a variety of new challenges, since classical theory and methodology can break down in surprising and unexpected ways.

High-dimensional statistics is often paraphrased with the expression $p >> n$ which refers to a linear regression setting where the number of covariates p is much larger than the number of samples n. Nevertheless, challenges with standard statistical approaches already appear when p is comparable to n and we will see that in more complex models issues due to high-dimensionality often manifest itself in a more subtle manners.

High-dimensional data are omnipresent and the approaches which we will discuss find applications in many disciplines. In this course we will explore examples from molecular biology, health care, speech recognition and finance.

In this script we distinguish between two typical tasks of high-dimensional statistics. The first task considers many explanatory variables $X_1, \dots, X_p$ and one response variable Y. The goal is to predict the response and to identify the most relevant covariates. We refer to this task as *Prediction and Feature Selection*. The second task considers one (or few) explanatory variable X and many response variables $Y_1, \dots, Y_p$. The aim is to identify those variables which

differ with respect to X (e.g. treatment groups A vs B). We refer to this task as *Feature Assessment*.

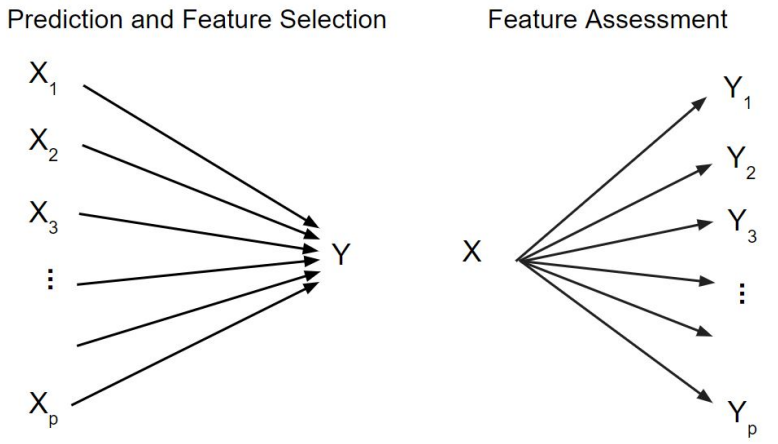


Figure 1.1: Prediction and feature selection (left graph) and feature assessment (right graph).

The script starts with the *multiple linear regression* model and the *least squares* estimator. We discuss the limitations of least squares in the $p >> n$ scenario, explain the challenge of *overfitting*, introduce the *generalization error* and elaborate on the *bias-variance dilemma*. We then discuss methods designed to overcome challenges in high dimensions. We start with *subset-* and *stepwise regression* and then discuss in more detail regularization methods, including *Ridge regression*, *Lasso regression* and *Elasticnet regression*. Next, we turn our attention to binary endpoints. In particular we discuss regularization in the context of the *logistic regression* model. We then talk about *classification trees*, *Random Forest* and *AdaBoost*. We then move to *time-to-event endpoints* and show how to extend the previously introduced methods to *survival data*. We introduce the *Brier score* to assess the generalization error in the survival context. The last section is devoted to *high-dimensional feature assessment*. Based on a *differential gene expression* example we will discuss the issue of *multiple testing*, introduce methods for *p-value adjustment*, and finally we will touch upon *variance shrinkage*.

Chapter 2

Multiple Linear Regression

In this chapter we will review multiple linear regression and in particular the Ordinary Least Squares (OLS) estimator. We will further investigate the challenges which appear in the high-dimensional setting where the number of covariates is large compared to the number of observations, i.e. $p >> n$.

2.1 Notation

We will typically denote the covariates by the symbol X. If X is a vector, its components can be accessed by subscripts X_j. The response variable will be denoted by Y. We use uppercase letters such as X, Y when referring to the generic aspects of a variable. Observed values are written in lowercase; hence the ith observed value of X is written as x_i (where x_i is again a scalar or vector). Matrices are represented by bold uppercase letters; for example, a set of n input p-vectors x_i, $i = 1, \ldots, n$ would be represented by the $n \times p$ matrix $\mathbf{X}$. All vectors are assumed to be column vectors, the ith row of $\mathbf{X}$ is x_i^T.

2.2 Ordinary Least Squares

Given a vector of inputs $X = (X_1, X_2, \ldots, X_p)$, in multiple regression we predict the output Y via the linear model:

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j.$$

The term β_0 is the intercept. If we include the constant variable 1 in X, include $\hat{\beta}_0$ in the vector of coefficients $\hat{\beta}$, then we can write

$$\hat{Y} = X^T \hat{\beta}.$$

How do we fit the linear model to a set of training data (i.e. how do we obtain the estimator $\hat{\beta}$)? We typically use *ordinary least squares* (OLS) where we pick the coefficient β to minimize the residual sum of squares

$$\begin{aligned} \mathrm{RSS}(\beta) &= \sum_{i=1}^{n} (y_i - x_i^T \beta)^2 \\ &= (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta) \\ &= \|\mathbf{y} - \mathbf{X}\beta\|_2^2. \end{aligned}$$

If the matrix $\mathbf{X^T X}$ is nonsingular, then the solution is given by

$$\hat{\beta} = (\mathbf{X^T X})^{-1} \mathbf{X^T y}.$$

Thus, the prediction at the new input point X_{new} is

$$\begin{aligned} \hat{Y} &= \hat{f}(X_{\text{new}}) \\ &= X_{\text{new}}^T \hat{\beta} \\ &= X_{\text{new}}^T (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}. \end{aligned}$$

Figures 2.1 and 2.2 show two geometric representations of the OLS estimator. In Figure 2.1 the n data points $(y_i, x_{i1}, \ldots, x_{ip})$ randomly spread around a p-dimensional hyperplane in a $p+1$-dimensional space; the random spread only occurs parallel to the y-axis and the hyperplane is defined via $\hat{\beta}$. Figure 2.2 shows a different representation where the vector $\mathbf{y}$ is a single point in the n-dimensional space $\mathbf{R}^n$; the fitted $\hat{\mathbf{y}}$ is the orthogonal projection onto the p-dimensional subspace of $\mathbf{R}^n$ spanned by the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_p$.

2.3 Overfitting

Overfitting refers to the phenomenon of modelling the noise rather than the signal. In case the true model is parsimonious (few covariates driving the response Y) and data on many covariates are available, it is likely that a linear combination of all covariates yields a higher likelihood than a combination of the few that are actually related to the response. As only the few covariates related to the response contain the signal, the model involving all covariates

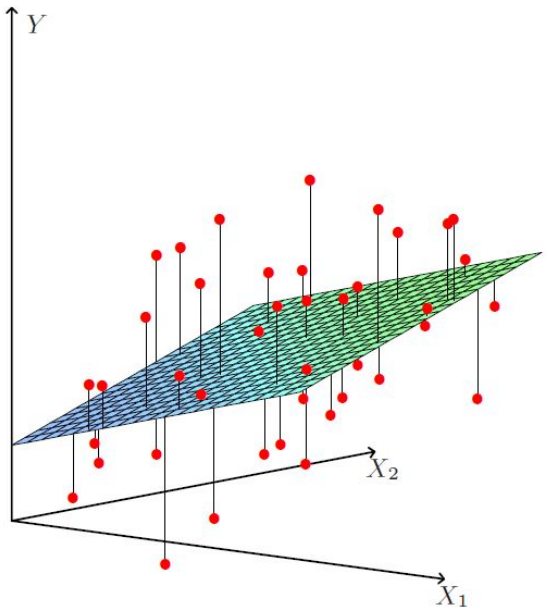


Figure 2.1: Data points spreading around the p-dimensional OLS hyperplane.

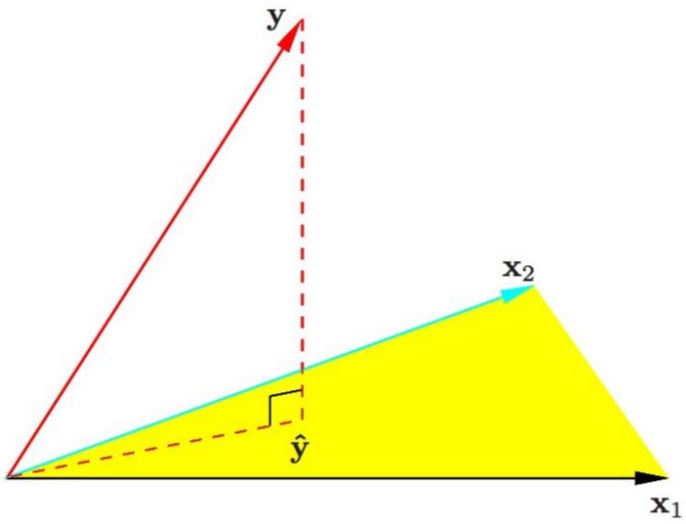


Figure 2.2: OLS fit $\hat{\mathbf{y}}$ as the orthogonal projection of $\mathbf{y}$ onto subspace spanned by covariates.

then cannot but explain more than the signal alone: it also models the error. Hence, it overfits the data.

We illustrate overfitting by generating artificial data. We simulate $n = 10$ training data points, take $p = 15$ and $X_{i1}, \ldots, X_{ip}$ i.i.d. $N(0, 1)$. We assume that the response depends only on the first covariate, i.e. $Y_i = \beta_1 X_{i1} + \epsilon_i$, where $\beta_1 = 2$ and ϵ_i i.i.d. $N(0, 0.5^2)$.

```
set.seed(1)
n <- 10
p <- 15
beta <- c(2,rep(0,p-1))

# simulate covariates
xtrain <- matrix(rnorm(n*p),n,p)
ytrain <- xtrain%*%beta+rnorm(n,sd=0.5)
dtrain <- data.frame(xtrain)
dtrain$y <- ytrain
```

We fit a univariate linear regression model with X1 as covariate and print the `summary`.

```
fit1 <- lm(y~X1,data=dtrain)
summary(fit1)
```

```
##
## Call:
## lm(formula = y ~ X1, data = dtrain)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.59574 -0.41567 -0.06222  0.18490  0.97592
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.1002     0.1785  -0.561     0.59
## X1            1.8070     0.2373   7.614 6.22e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5558 on 8 degrees of freedom
## Multiple R-squared:  0.8787, Adjusted R-squared:  0.8636
## F-statistic: 57.97 on 1 and 8 DF,  p-value: 6.223e-05
```

The coefficient for X1 is close to the true value. The R squared value $R^2 = 0.88$ indicates that the model fits the data well. In order to explore what happens

if we add noise covariates, we re-fit the model with an increasing number of covariates, i.e. $p = 4$, 8 and 15.

```
fit4 <- lm(y~X1+X2+X3+X4,data=dtrain)
fit8 <- lm(y~X1+X2+X3+X4+X5+X6+X7+X8,data=dtrain)
fit15 <- lm(y~.,data=dtrain) #all 15 covariates
```

The next plot shows the data points (black circles) together with the fitted values (red crosses).

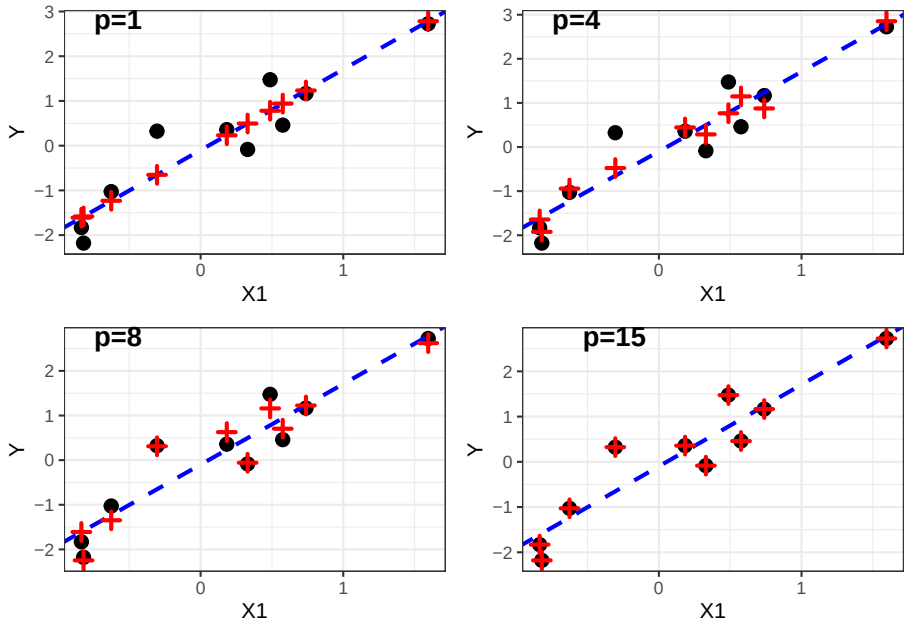


Figure 2.3: Observed and fitted values for models with increasing p.

With increasing p the fitted values start to deviate from the true model (blue line) and they move closer towards the observed data points. Finally, with $p = 15$, the fitted values match perfectly the data, i.e. the model captures the noise and overfits the data. In line with these plots we note that the R squared values increase with p.

The following figure shows the regression coefficients for the different models. The larger the p, the bigger the discrepancy to the true coefficients.

This becomes even more evident when calculating the mean squared error between the estimated and true coefficients.

The `summary` of the full model with $p = 15$ indicates that something went wrong.

Table 2.1: R2 for models with increasing p.

model	R2
p=1	0.88
p=4	0.90
p=8	0.98
p=15	1.00

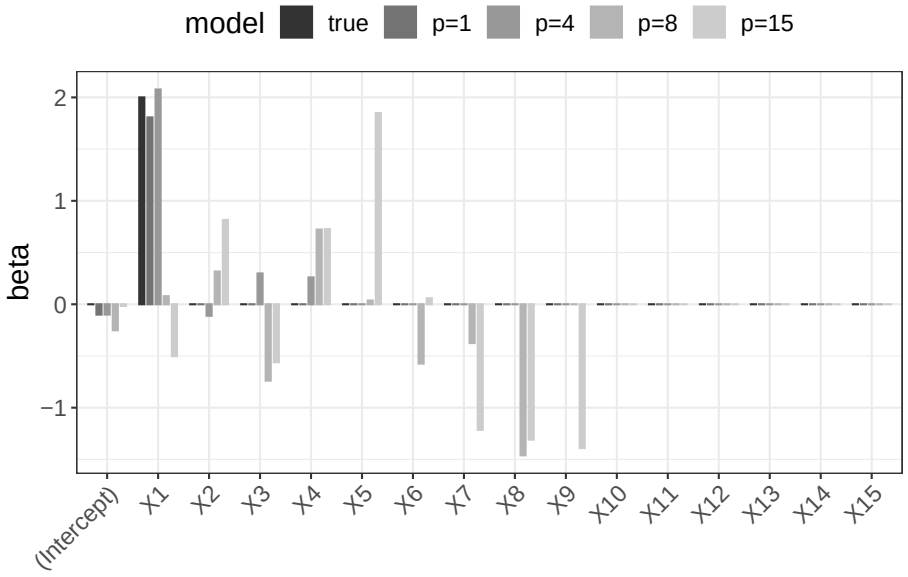


Figure 2.4: Regression coefficients for models with increasing p.

Table 2.2: MSE between estimated and true coefficients.

	mse
p=1	0.02
p=4	0.04
p=8	0.84
p=15	1.63

```
summary(fit15)
```

```
##
## Call:
## lm(formula = y ~ ., data = dtrain)
##
## Residuals:
## ALL 10 residuals are 0: no residual degrees of freedom!
##
## Coefficients: (6 not defined because of singularities)
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.01592        NaN     NaN      NaN
## X1          -0.50138        NaN     NaN      NaN
## X2           0.81492        NaN     NaN      NaN
## X3          -0.56052        NaN     NaN      NaN
## X4           0.72667        NaN     NaN      NaN
## X5           1.84831        NaN     NaN      NaN
## X6           0.05759        NaN     NaN      NaN
## X7          -1.21460        NaN     NaN      NaN
## X8          -1.30908        NaN     NaN      NaN
## X9          -1.39005        NaN     NaN      NaN
## X10               NA         NA      NA       NA
## X11               NA         NA      NA       NA
## X12               NA         NA      NA       NA
## X13               NA         NA      NA       NA
## X14               NA         NA      NA       NA
## X15               NA         NA      NA       NA
##
## Residual standard error: NaN on 0 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:    NaN
## F-statistic:   NaN on 9 and 0 DF,  p-value: NA
```

There is a note saying "no residual degrees of freedom". Furthermore, many entries in the table of coefficients are not available and a note says that coefficients cannot be calculated because of *singularities*. What has happened? In fact, the OLS estimator as introduced above is not well defined. The design matrix $\mathbf{X}$ is rank deficient ($\text{rank}(\mathbf{X}) = n < p$) and therefore the matrix $\mathbf{X}^T\mathbf{X}$ is singular (not invertible). We can check this by calculating the determinant.

```
x <- model.matrix(fit15)
det(t(x)%*%x)
```

```
## [1] -2.8449e-81
```

In this simulation exercise we illustrated the problem of overfitting. We have seen that the models with large p fit the data very well, but the estimated coefficients are far off from the truth. In practice we do not know the truth. How do we know when a model is overfitting and how do we decide what a "good" model is? Shortly we will introduce the Generalization Error which will shed light on this question.

We end this section with the helpful ***10:1 rule:***

> In order to avoid overfitting the number of predictors (or covariates) **p should be less than n/10**. This rule can be extended to binary and time-to-event endpoints. For binary endpoints we replace n with $\min\{n_0, n_1\}$ and for time-to-event with n_{events}.

2.4 Generalization Error

The ultimative goal of a good model is to make good predictions for the future. That is we need to assess how the fitted model generalizes beyond the "observed" data. Conceptually, given new input data x_{new}, the model provides a prediction $\hat{Y} = \hat{f}(x_{\text{new}})$. The *Generalization Error* is the expected discrepancy between the prediction $\hat{Y} = \hat{f}(x_{\text{new}})$ and the actual outcome Y_{new}

$$\text{Err}(x_{\text{new}}) = E[(Y_{\text{new}} - \hat{f}(x_{\text{new}}))^2].$$

One can show that this error can be decomposed into three terms

$$\text{Err}(x_{\text{new}}) \quad = \quad \sigma_\epsilon^2 + \text{Bias}^2(\hat{f}(x_{\text{new}})) + \text{Var}(\hat{f}(x_{\text{new}})), \tag{2.1}$$

where the first term is the irreducible error (or "noise"), the second term describes the systematic bias from the truth and the third term is the variance of the predictive model. For linear regression the expected variance can be approximated by $\sigma_\epsilon^2 \frac{p}{N}$. Complex models (with large number of covariates) have typically a small bias but a large variance. Therefore the equation above is referred to as the bias-variance dilemma as it describes the conflict in trying simultaneously minimize both sources of error, bias and variance.

How do we calculate the Generalization Error in practice? The most simple approach is to separate the data into a training and testing set (see Figure 2.5). The model is fitted (or "trained") on the training data and the Generalization Error is calculated on the test data and quantified using the root-mean-square error (RMSE)

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n_{\text{test}}} (y_{\text{test,i}} - \hat{y}_i)^2}{n_{\text{test}}}}.$$

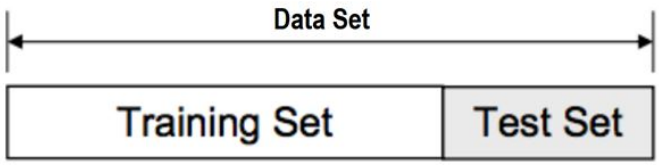


Figure 2.5: Splitting the data into training set and test sets.

We illustrate this based on the dummy data. First we simulate test data.

```
# simulate test data
xtest <- matrix(rnorm(n*p),n,p)
ytest <- xtest%*%beta+rnorm(n,sd=0.5)
dtest <- data.frame(xtest)
dtest$y <- ytest
```

Next, we take the fitted models and make predictions on the test data

```
# prediction
pred1 <- predict(fit1,newdata = dtest)
pred4 <- predict(fit4,newdata = dtest)
pred8 <- predict(fit8,newdata = dtest)
pred15 <- predict(fit15,newdata = dtest)
```

and we calculate the RMSE.

```
# rmse
rmse <- data.frame(
  RMSE(pred1,ytest),RMSE(pred4,ytest),
  RMSE(pred8,ytest),RMSE(pred15,ytest)
)
colnames(rmse) <- paste0("p=",c(1,4,8,15))
rownames(rmse) <- "RMSE"
kable(rmse,digits=2,booktabs=TRUE,
      caption="RMSE for models with increasing p.")
```

The models with $p = 1$ and 4 achieve a good error close to the "irreducible" σ_ϵ. On the other hand the predictions obtained with $p = 8$ and 15 are very poor (RMSEs are 6 to 8-fold larger).

Table 2.3: RMSE for models with increasing p.

	p=1	p=4	p=8	p=15
RMSE	0.54	0.63	3.28	3.7

Chapter 3

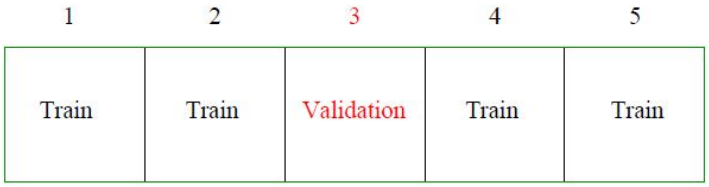
Regularization

We have seen that multiple regression falls short in the high-dimensional context. It leads to overfitting and as a result in large estimates of regression coefficients. Augmentation of the least-squares optimization with constraints on the regression coefficients can decrease the risk of overfitting. In the following we will discuss methods which minimize the residual sum of squares, RSS( ), under some constraints on the parameter β.

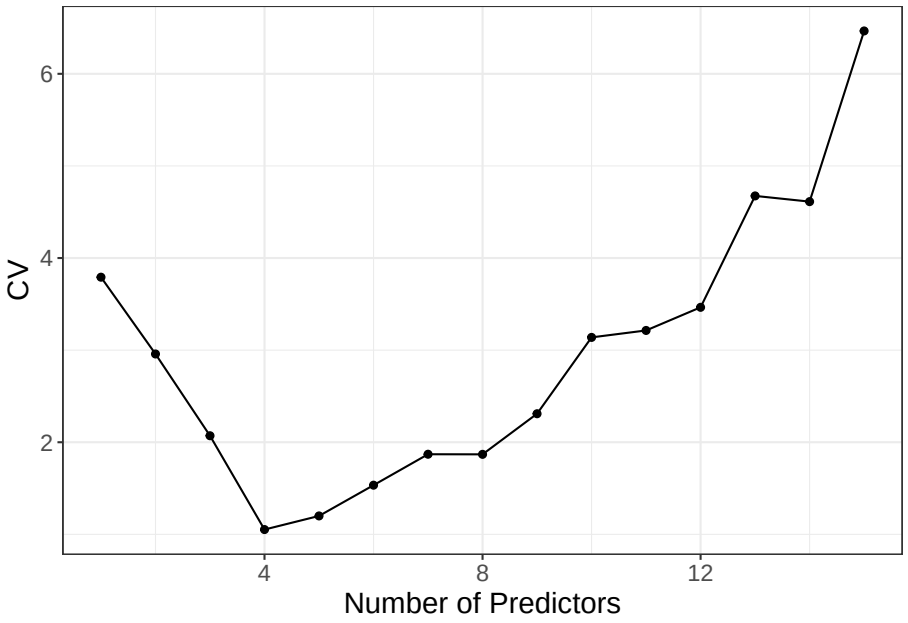
3.1 Model Selection

We will shortly see that the approaches which we introduce do not only fit one single model but they explore a whole series of models (indexed as $m = 1, \ldots, M$). Model selection refers to the choice of an optimal model achieving a low generalization error. A plausible approach would be to fit the different models to the training data and then select the model with smallest error on the test data. However, this is an illegitimate approach as the test data has to be kept untouched for the final evaluation of the selected model. Therefore we guide model selection by approximating the generalization error using training data only. We review now two such approximations, namely, cross-validation and the Akaike information criterion (AIC).

K-fold cross-validation approximates the prediction error by splitting the training data into K chunks as illustrated below (here $K = 5$).



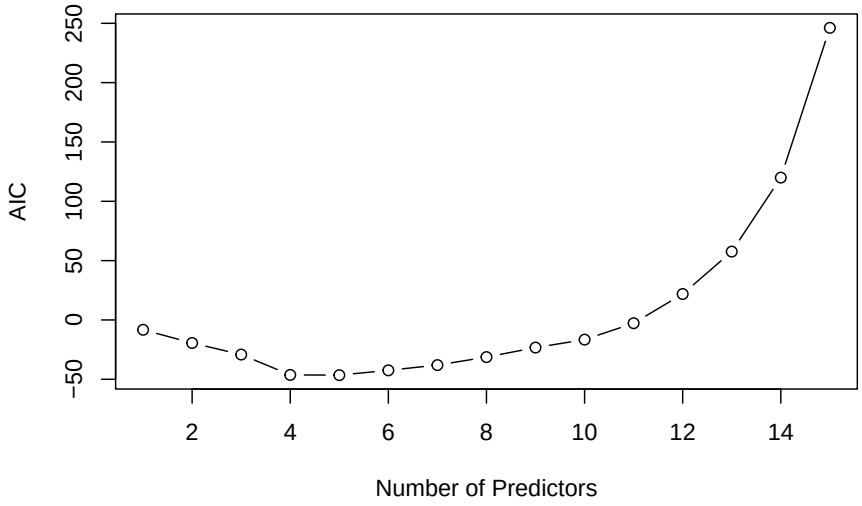
Each chunk is then used as "hold-out" validation data to estimate the error of mth model trained on the other $K-1$ data chunks. In that way we obtain K error estimates and we typically take the average as the cross-validation error of model m (denoted by CV_m). The next plot shows a typical cross-validation error plot. This curve attains its minimum at a model with $p_m = 4$ (p_m is the number of included predictors in model m).



The AIC approach is founded in information theory and selects the model with smallest AIC

$$\text{AIC}_m = -2\,\text{loglik} + 2\,p_m.$$

Thus, AIC rewards goodness of fit (as assessed by the likelihood function *loglik*) and penalizes model complexity (by the term $2p_m$). The figure below shows for the same example the AIC curve. Also the AIC approaches suggests to use a model with $p_m = 4$ predictors.



3.2 Subset- and Stepwise Regression

The most common approach to impose constraints is subset selection. In this approach we retain only a subset of the variables, and eliminate the rest from the model. OLS is used to estimate the coefficients of the inputs that are retained. More formally, given a subset $S \subset \{1, \dots, p\}$ we solve the optimization problem

$$\hat{\beta}_S = \arg\min_{\beta_j = 0 \ \forall j \notin S} \mathrm{RSS}(\beta).$$

It is easy to show that this is equivalent to OLS regression based on subset S covariates, i.e.

$$\hat{\beta}_S = (\mathbf{X}_S^T \mathbf{X}_S)^{-1} \mathbf{X}_S^T \mathbf{y}.$$

In practice we need to explore a sequence of subsets $S_1, \dots, S_M$ and choose an optimal subset by either a re-sampling approach or by using an information criterion (see Section 3.1). There are a number of different strategies available. *Best subsets regression* consists of looking at all possible combinations of covariates. Rather than search though all possible subsets, we can seek a good path through them. Two popular approaches are *backward stepwise* regression which starts with the full model and sequentially deletes covariates, whereas *forward*

Table 3.1: Inclusion of covariates in forward stepwise regression.

Step	Df	Deviance	Resid. Df	Resid. Dev	AIC
	NA	NA	9	22.468	10.095
+ X1	1	20.017	8	2.450	-10.064
+ X4	1	0.883	7	1.567	-12.535
+ X9	1	0.376	6	1.191	-13.277

stepwise regression starts with the intercept, and then sequentially adds into the model the covariate that most improves the fit.

In `R` we can use `regsubsets` from the `leaps` package or `stepAIC` from the `MASS` package to perform subset- and stepwise regression. For example to perform forward stepwise regression based on AIC we proceed as follows.

```
# Forward regression
fit0 <- lm(y~1,data=dtrain)
up.model <- paste("~", paste(colnames(dtrain[,-(p+1)]), collapse=" + "))
fit.fw <- stepAIC(fit0,
                  direction="forward",
                  scope=
                    list(lower=fit0,
                         upper=up.model)
                    ,
                  trace = FALSE
)
```

We can summarize the stepwise process.

```
kable(as.data.frame(fit.fw$anova),digits=3,booktabs=TRUE
      ,caption="Inclusion of covariates in forward stepwise regression.")
```

Finally we can retrieve the regression coefficients of the optimal model.

```
kable(broom::tidy(fit.fw),digits=3,booktabs=TRUE,
      caption="Regression coefficients of the optimal model.")
```

3.3 Ridge Regression

Subset selection as outlined above works by either including or excluding covariates, i.e. constrain specific regression coefficients to be zero.

Table 3.2: Regression coefficients of the optimal model.

term	estimate	std.error	statistic	p.value
(Intercept)	0.210	0.157	1.334	0.231
X1	1.611	0.243	6.624	0.001
X4	-0.508	0.205	-2.475	0.048
X9	-0.322	0.234	-1.376	0.218

An alternative is *Ridge regression*, which regularizes the optimization problem by shrinking regression coefficients towards zero. This discourages complex models because models that overfit tend to have larger coefficients. Ridge regression can be formulated as a constrained optimization problem

$$\hat{\beta}_c^{\text{Ridge}} = \arg \min_{\|\beta\|_2^2 \leq c} \text{RSS}(\beta).$$

The geometry of the optimization problem is illustrated in Figure 3.1. It shows the levels sets of RSS(β), ellipsoids centered around the OLS estimate, and the circular ridge parameter constraint, centered around zero with radius $c > 0$. The Ridge estimator is the point where the smallest level set hits the constraint. Exactly at that point the RSS( ) is minimized over those β's that "live" inside the constraint.

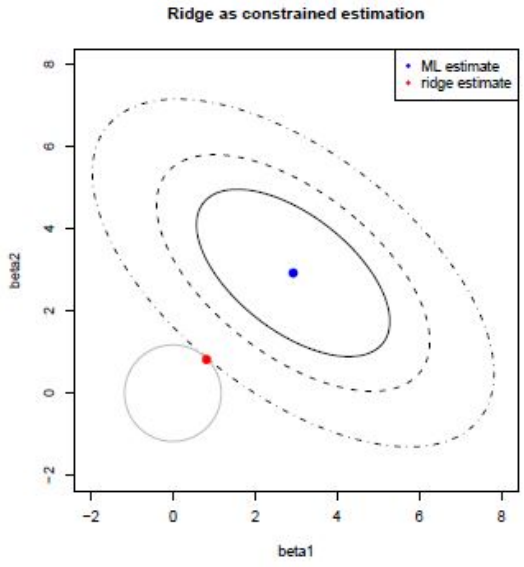


Figure 3.1: Geometry of Ridge regression.

Alternatively, Ridge regression can be cast as the optimization of the penalised residual sum of squares with a *penalty* on the magnitude of the coefficients, i.e.

$$\hat{\beta}_\lambda^{\text{Ridge}} = \arg\min_\beta \text{RSS}(\beta) + \lambda\|\beta\|_2^2.$$

Both formulations are equivalent in the sense that there is a one-to-one relationship between the tuning parameters c and λ. We will use more often the latter "penalisation" formulation. The parameter λ is the amount of penalisation. Note that with no penalization, $\lambda = 0$, Ridge regression coincides with OLS. Increasing λ has the effect of shrinking the regression coefficients to zero.

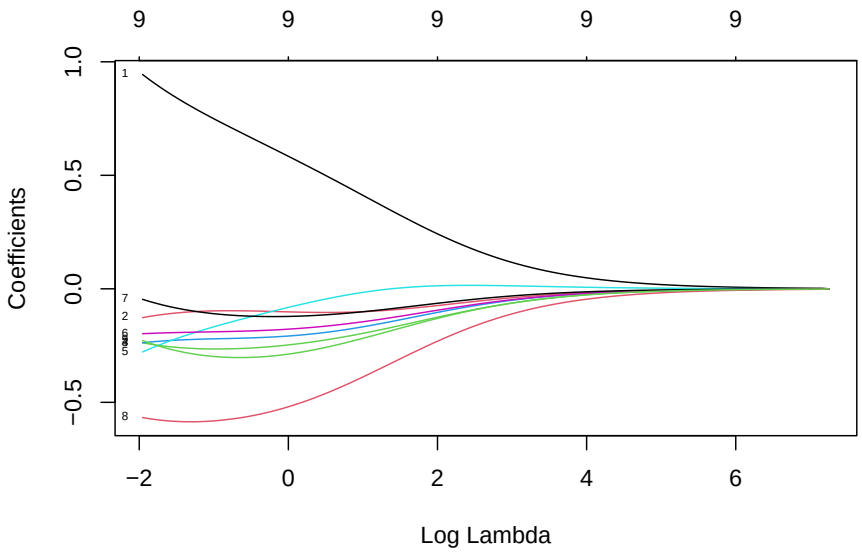
The Ridge optimization problem has the closed form solution (see exercises)

$$\hat{\beta}_\lambda^{\text{Ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}.$$

Note that for $\lambda > 0$ the matrix $\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}$ has always full rank and therefore Ridge regression is well defined even in the high-dimensional context (in contrast to OLS).

Ridge regression is implemented in the package `glmnet`. We use `alpha=0` and can call

```
fit.ridge.glmnet <-glmnet(x=xtrain,y=ytrain,alpha=0)
plot(fit.ridge.glmnet,xvar="lambda",label=TRUE)
```



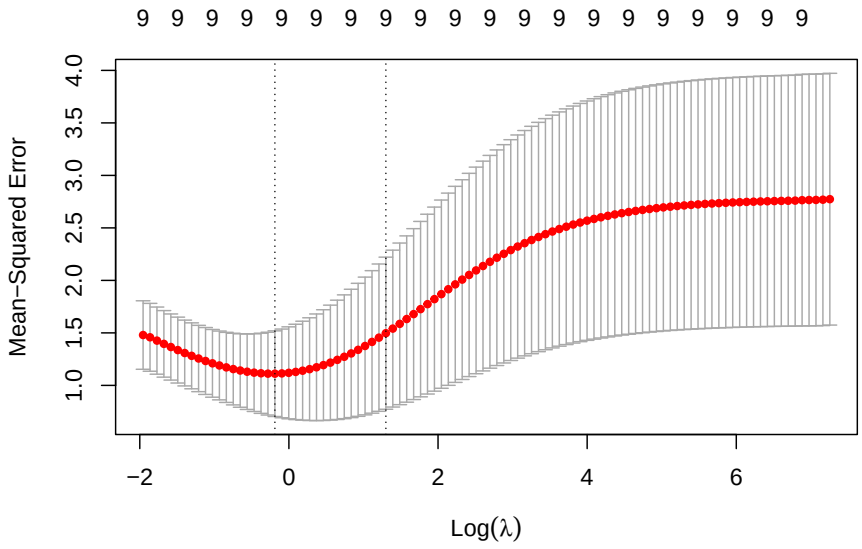
3.3.1 Choice of penalty parameter

In subset- and stepwise regression we had to identify the optimal subset. Similarly, for Ridge regression model selection consists of selecting the tuning parameter λ. We proceed by choosing a grid of values $0 < \lambda_1 < \lambda_2 < ... < \lambda_M < \infty$ and proceed as explained in Section 3.1, that is we choose the optimal λ_{opt} by either re-sampling or information criteria. In `glmnet` we use cross-validation using the command `cv.glmnet`.

```
cv.ridge.glmnet <-cv.glmnet(x=xtrain,y=ytrain,alpha=0)
```

The next plot shows the cross-validation error with upper and lower standard deviations as a function of the lambda values (note the log scale for the lambdas).

```
plot(cv.ridge.glmnet)
```



The tuning parameter with the smallest cross-validation error is stored in the argument `lambda.min`.

```
cv.ridge.glmnet$lambda.min
```

```
## [1] 0.8286695
```

Another choice is `lambda.1se` which denotes the largest λ within 1 standard error of the smallest cross-validation error.

```
cv.ridge.glmnet$lambda.1se
```

```
## [1] 3.671521
```

3.3.2 Shrinkage property

The OLS estimator becomes unstable (high variance) in presence of collinearity. A nice property of Ridge regression is that it counteracts this by shrinking low-variance components more than high-variance components.

This can be best understood by rotating the data using a principle component analysis (see Figure 3.2). In particular, we consider the singular value decomposition

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T,$$

where the columns of $\mathbf{U}$ form an orthonormal basis of the column space of $\mathbf{X}$, $\mathbf{D}$ is a diagonal matrix with entries $d_1 \geq d_2 \geq ... \geq d_p \geq 0$ called the singular values, and the columns of $\mathbf{V}$ represent the principle component directions. For OLS the vector of fitted values $\hat{\mathbf{y}}^{\text{OLS}}$ is the orthogonal projection of $\mathbf{y}$ onto the column space of $\mathbf{X}$. Therefore, in terms of rotated data we have

$$\hat{\mathbf{y}}^{\text{OLS}} = \sum_{j=1}^{p} \mathbf{u}_j \mathbf{u}_j^T \mathbf{y}.$$

Similarly, we can represent the fitted values from Ridge regression as

$$\hat{\mathbf{y}}^{\text{Ridge}} = \sum_{j=1}^{p} \mathbf{u}_j \frac{d_j^2}{d_j^2 + \lambda} \mathbf{u}_j^T \mathbf{y}.$$

This shows that the level of shrinkage $\frac{d_j^2}{d_j^2+\lambda}$ is largest in the direction of the last principle component, which in return is the direction where the data exhibits smallest variance.

3.3.3 Effective degrees of freedom

Although Ridge regression involves all p covariates the *effective degrees of freedom* are smaller than p as we have imposed constraints through the penalty. In

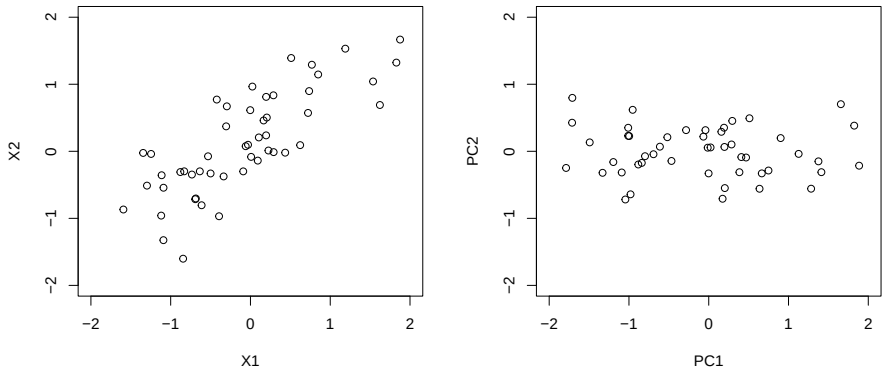


Figure 3.2: Left plot: 2-dimensional input data. Right plot: input data rotated using principle component analysis.

the book Hastie et al. (2001) it is shown that the effective degrees of freedom for Ridge regression, $\nu_\lambda^{\text{ridge}}$, are given by

$$\nu_\lambda^{\text{ridge}} = \sum_{j=1}^{p} \frac{d_j^2}{d_j^2 + \lambda},$$

where $d_1, \ldots, d_p$ are the singular values of $\mathbf{X}$.

```
# get singular values
fit.svd <- svd(xtrain) #fit.svd$d

# ridge degree of freedom for lambdaopt
df_lambdaopt <- sum(fit.svd$d^2/(fit.svd$d^2+cv.ridge.glmnet$lambda.min))
df_lambdaopt
```

```
## [1] 6.167042
```

3.3.4 Bayesian interpretation

We have introduced regularization by least-squares optimization with additional constraints on β. An alternative approach to regularization is based on Bayesian statistics. In a Bayesian setting the parameter $\beta = (\beta_1, \ldots, \beta_p)$ is itself a random variable with *prior* distribution $p(\beta)$. Bayesian inference is based on the *posterior* distribution

$$p(\beta|D) = \frac{p(D|\beta)p(\beta)}{p(D)},$$

where D denotes the data and $p(D|\beta)$ is the likelihood function. In the exercises we will show that the Ridge solution can be viewed as the maximum a posteriori (MAP) estimate of a hierarchical Bayesian model where the data follows a multivariate regression model

$$Y_i|X_i, \beta \sim N(X_i^T\beta, \sigma^2),\ i = 1, \ldots, n$$

and the regression coefficients are equipped with prior

$$\beta_j \sim N(0, \tau^2),\ j = 1, \ldots, p.$$

For many practical problems the posterior distribution is analytically not tractable and inference is typically based on sampling from the posterior distribution using a procedure called Markov chain Monte Carlo (MCMC). The software packages BUGS and JAGS automatically build MCMC samplers for complex hierarchical models. We use `rjags` to illustrate the procedure for the Bayesian Ridge regression model (see Figure 3.3).

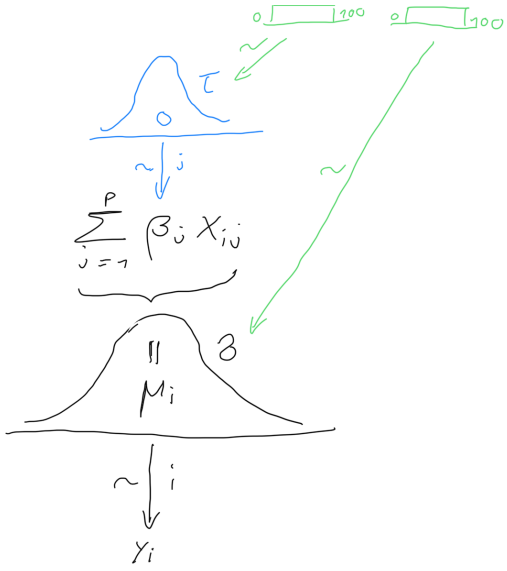


Figure 3.3: The Bayesian Ridge regression model.

First we specify the model, prepare the input data and provide initial values.

```
library(rjags)

# model
bayesian_ridge <-"model{
  for (i in 1:n){
```

```
    y[i] ~ dnorm (mu[i], 1/sig^2)
    mu[i] <- inprod(b,x[i,])
  }
  for (j in 1:p){
    b[j] ~ dnorm (0, 1/tau^2)
  }
  sig~dunif(0,100)
  tau~dunif(0,100)
}
"

# data
dat.jags <- list(x=xtrain,y=ytrain,p=ncol(xtrain),n=nrow(xtrain))

# initial values
inits <- function (){

  list (b=rnorm(dat.jags$p),sig=runif(1),tau=runif(1))

}
```

We use the function `jags.model` to setup an MCMC sampler with `n.chains=3` chains (the number of samples, or MCMC iterations, used for adaptation is per default set to 1000).

```
# setup jags model
jags.m <- jags.model(textConnection(bayesian_ridge),
                     data=dat.jags,
                     inits=inits,
                     n.chains=3,
                     quiet=TRUE)
```

After a burn-in period of 500 steps we use `coda.samples` to generate the posterior samples.

```
# burn-in
update(jags.m, n.iter=500)

# mcmc samples for inference
posterior.samples <- coda.samples( jags.m,
                                   variable.names = c("b","sig","tau"),
                                   n.iter=10000,thin=10) # thinning=10
```

There are several R packages to investigate the posterior distribution. For example with `MCMCsummary` we can extract key summary information, i.e. mean,

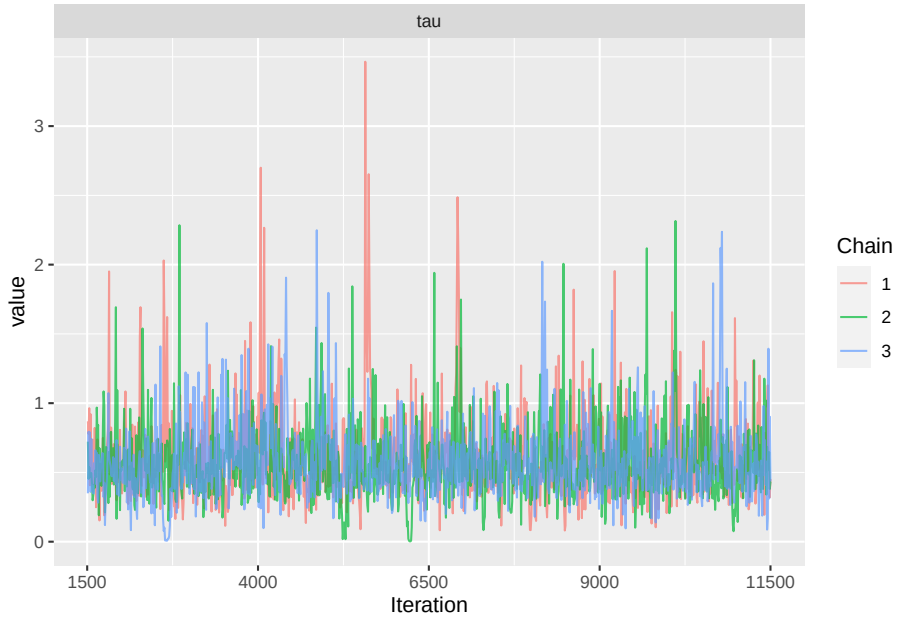
median, quantiles, Gelman-Rubin convergence statistic and the number of effective samples.

```
library(MCMCvis)
MCMCsummary(posterior.samples,
            round=2,
            params=c("sig","tau","b"))%>%
  kable
```

	mean	sd	2.5%	50%	97.5%	Rhat	n.eff
sig	0.86	0.41	0.36	0.76	1.89	1.01	1515
tau	0.58	0.30	0.17	0.53	1.28	1.00	1479
b[1]	0.74	0.60	-0.25	0.69	2.08	1.01	1421
b[2]	-0.09	0.30	-0.69	-0.09	0.55	1.00	2151
b[3]	-0.29	0.38	-1.07	-0.30	0.48	1.00	2270
b[4]	-0.19	0.37	-0.95	-0.18	0.50	1.01	2625
b[5]	-0.13	0.53	-1.35	-0.10	0.84	1.00	1910
b[6]	-0.14	0.33	-0.80	-0.14	0.51	1.00	2705
b[7]	-0.05	0.31	-0.63	-0.05	0.61	1.01	1915
b[8]	-0.52	0.43	-1.40	-0.52	0.32	1.00	2269
b[9]	-0.16	0.41	-0.92	-0.17	0.72	1.00	1831

Or, we can use the package`ggmcmc` and produce a traceplot to check the representativeness of the MCMC samples.

```
library(ggmcmc)
ggs.mcmc <- ggs(posterior.samples)
ggs_traceplot(ggs.mcmc,family="tau")
```

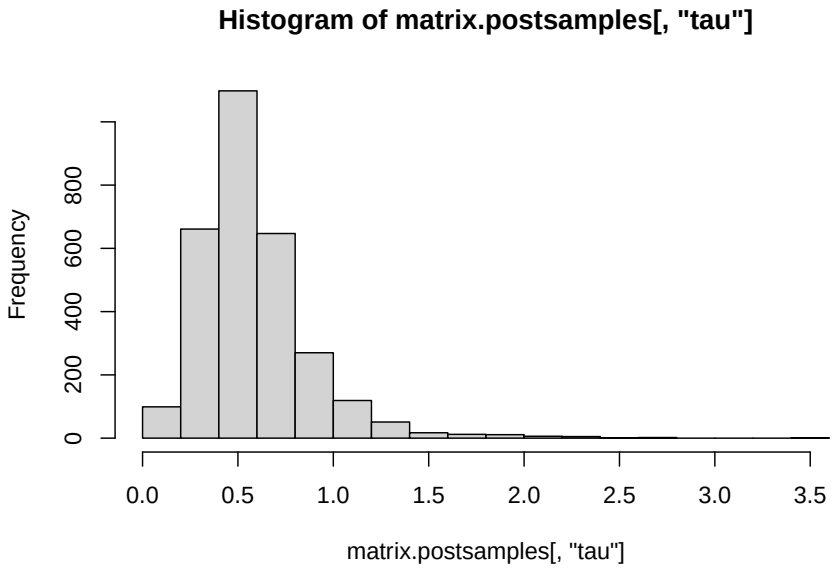


Alternatively, we can directly access the posterior samples and calculate any summary statistics of interest.

```
# posterior samples as matrix
matrix.postsamples <- as.matrix(posterior.samples)
dim(matrix.postsamples)
```

```
## [1] 3000   11
```

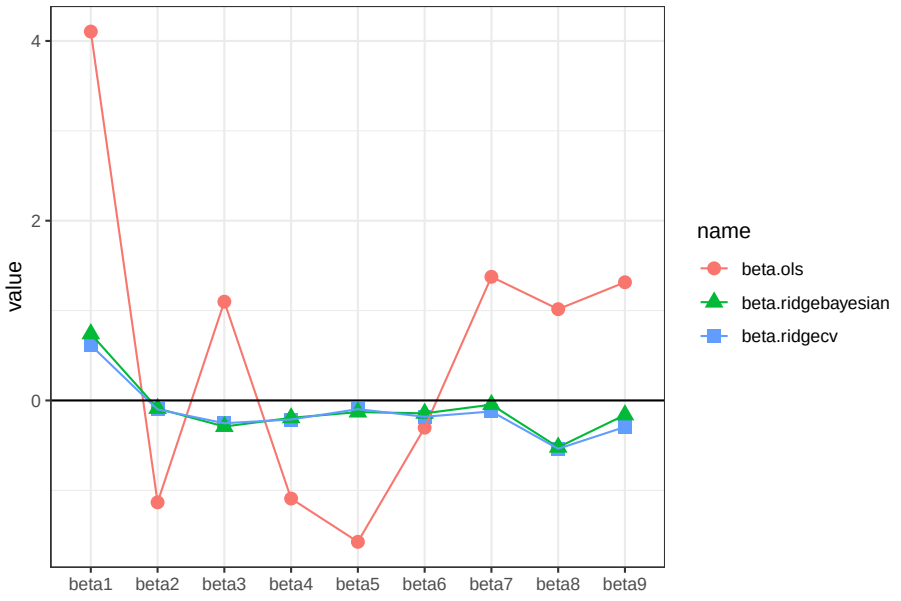
```
# histogram of posterior
hist(matrix.postsamples[,"tau"])
```




```
# posterior mean
colMeans(matrix.postsamples) # posterior mean
```

```
##        b[1]        b[2]        b[3]        b[4]        b[5]        b[6]
##  0.74111453 -0.08923934 -0.29030549 -0.19335623 -0.12935140 -0.14329942
##        b[7]        b[8]        b[9]         sig         tau
## -0.04671801 -0.52066557 -0.16102812  0.85820256  0.58398058
```

Finally, we compare the regression coefficients from OLS, Ridge regression (λ obtained using cross-validation) and Bayesian Ridge regression.



The coefficients obtained from Ridge regression and Bayesian Ridge regression are almost identical.

3.3.5 Splines

Ridge regression and high-dimensionality play a role in many subfields of statistics. We illustrate this with the example of smoothing splines for univariate non-parametric regression.

Sometimes it is extremely unlikely that the true function $f(X)$ is actually linear in X. Consider the following example.

How can we approximate the relationship between Y and X? The most simple approximation is a straight horizontal line (dashed blue line; the true sinusoidal function is depicted in black).

Clearly this approximation is too rigid. Next, we try a piecewise constant approximation with two inner "knots".

Finally, we use a piecewise linear function.

The approximation improves. Nevertheless it would be nice if the different line segments would line up. What we need are piecewise polynomials which are "smooth" at the knots. Such functions are called "splines". We assume that f can be expressed by a set of basis functions

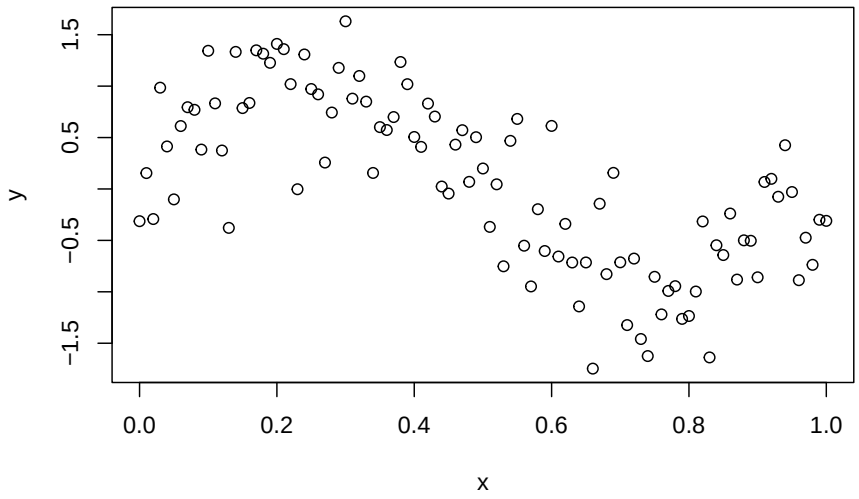


Figure 3.4: Non-linear (sinusoidal) relationship between Y and X.

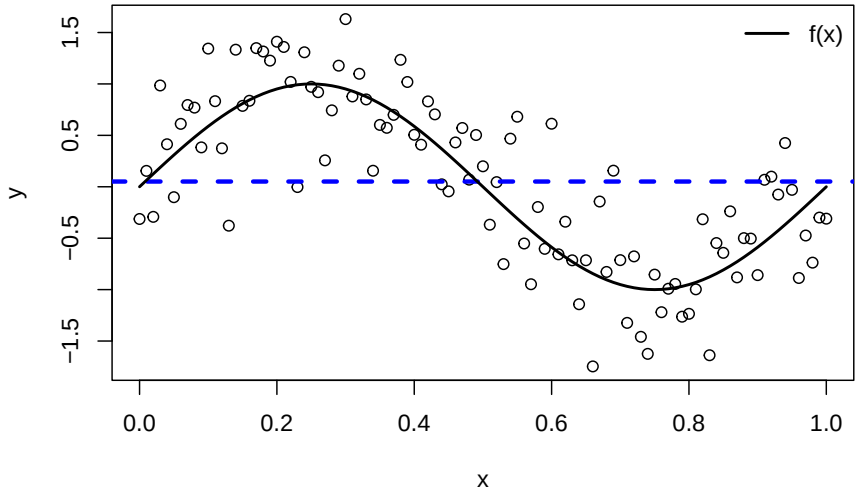


Figure 3.5: Approximation by a constant.

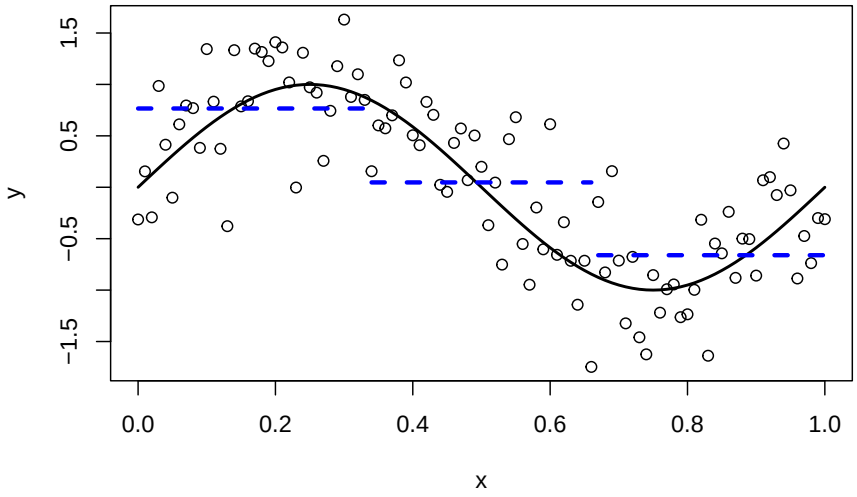


Figure 3.6: Piecewise constant approximation.

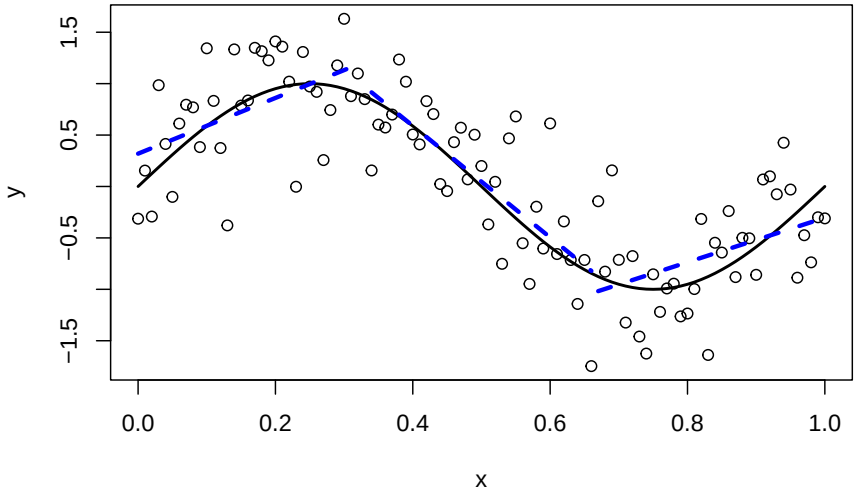
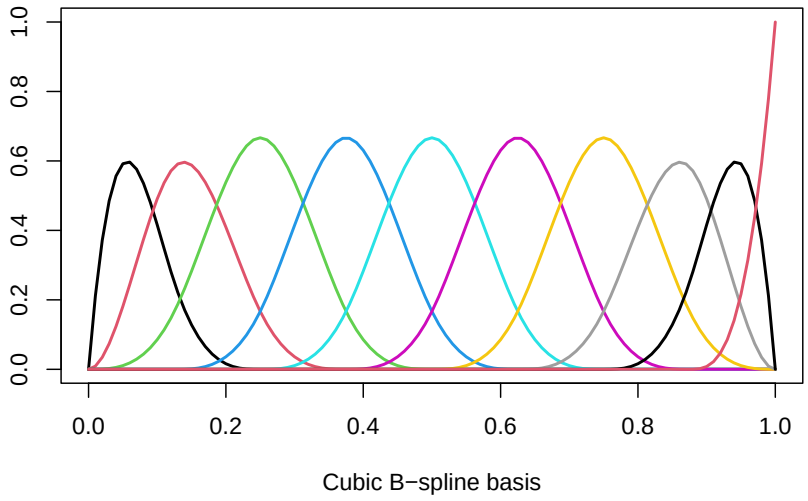


Figure 3.7: Piecewise linear approximation.

$$f(X) = \sum_{j=1}^{p} \beta_j B_j(X).$$

For example for a cubic spline with K fixed knots and fixed polynomial degree $d = 3$ ("cubic") we have $p = K + d + 1$ and the $B_j(x)$'s form a B-spline basis (one could also use the truncated-power basis). The coefficients β_m are estimated using OLS. Although we have only one single variable X, the design matrix consists of $p = K + d + 1$ features and we quickly run into issues due to overfitting. In `R` we obtain a B-spline basis with `bs` and we can plot the basis functions $B_j(x)$ as follows.

```
spl <- bs(x,df=10) # cubic spline with p=10 degrees of freedom
plot(spl[,1]~x, ylim=c(0,max(spl)), type='l', lwd=2, col=1,
     xlab="Cubic B-spline basis", ylab="")
for (j in 2:ncol(spl)) lines(spl[,j]~x, lwd=2, col=j)
```



The estimated coefficients $\hat{\beta}_j$ are obtain using `lm`.

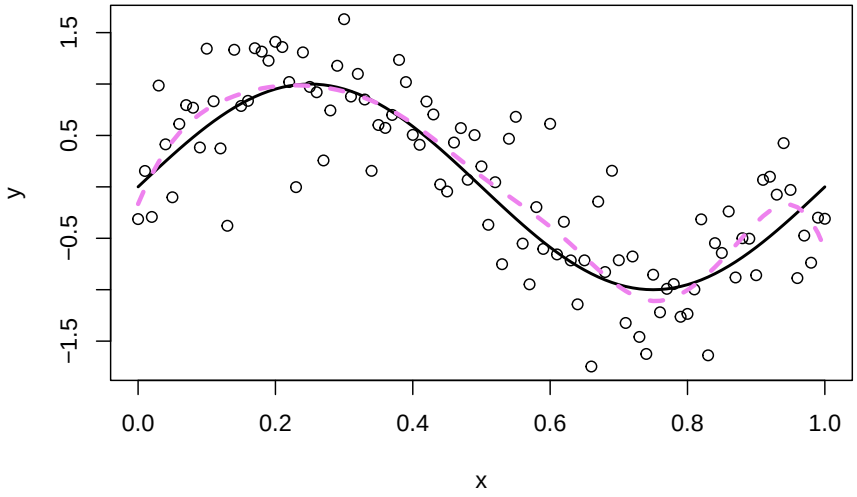
```
fit.csp <- lm(y~spl)
#fit.csp <- lm(y~bs(x,df=10))
coef(fit.csp)
```

```
## (Intercept)        spl1        spl2        spl3        spl4        spl5
```

```
##  -0.1664090   0.6710022   1.0956429   1.2056968   0.9713568   0.2323033
##        spl6        spl7        spl8        spl9       spl10
##  -0.2876482  -1.2456044  -0.3914716   0.2894841  -0.4376537
```

The cubic spline with $p = 10$ degrees of freedom fits the data well as shown in the next plot (in dashed violet).

```
plot(x, y)
lines(x, fx, lwd = 2)
lines(x, predict(fit.csp), lty = 2, col = "violet",lwd=3)
```



An alternative approach are so-called *smoothing splines*, where we take $p = n$ and the $B_j(x)$'s are an n-dimensional set of basis functions representing the family of natural cubic splines with knots at the unique values of x_i, $i = 1, \dots, n$. The coefficients β_j cannot be estimated using OLS as the number p of basis functions (columns of the design matrix) equals the number of observations n. Smoothing splines overcome this hurdle by imposing a generalized ridge penalty on the spline coefficients β_j, i.e.

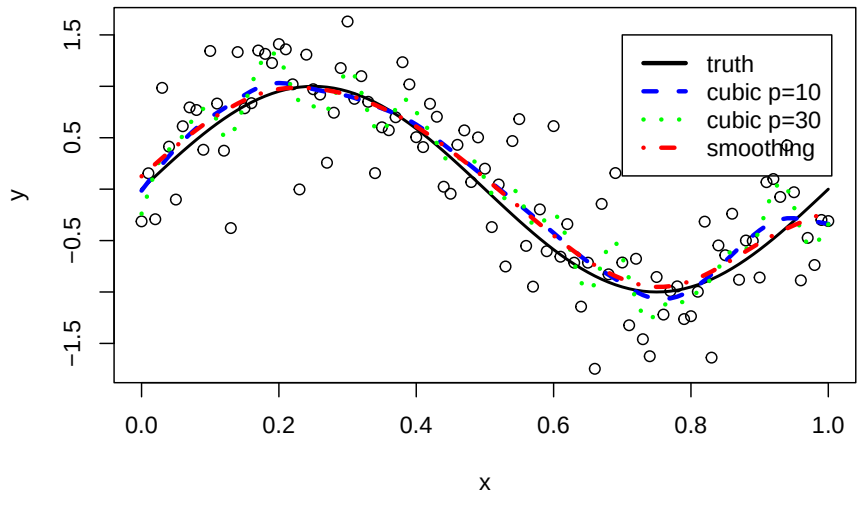
$$\hat{\beta}_\lambda = \arg\min_\beta \ \|\mathbf{y} - \mathbf{B}\beta\|^2 + \lambda\beta^T\Omega\beta,$$

where $\mathbf{B}$ is the design matrix with *jth* column $(B_j(x_1), \dots, B_j(x_n))^T$. In practice we can fit smoothing splines using the function `smooth.spline`. The penalty

term is specified by setting the effective degrees of freedom ν or by selecting λ using cross-validation (see Section 3.3.1).

We fit smoothing splines to our simulation example.

```
# smoothing spline with 10 effective degrees of freedom
fit.smsp.df10 <- smooth.spline(x, y, df = 10)

# smoothing spline with 30 effective degrees of freedom
fit.smsp.df30 <- smooth.spline(x, y, df = 30)

# smoothing spline with effective degrees of freedom estimated by cv
fit.smsp.cv <- smooth.spline(x, y)

plot(x, y)
lines(x, fx, lwd = 2)
lines(x, fit.smsp.df10$y, lty = 2, col = "blue",lwd=3)
lines(x, fit.smsp.df30$y, lty = 3, col = "green",lwd=3)
lines(x, fit.smsp.cv$y, lty = 4, col="red",lwd=3)
legend(0.7,1.5,
       lty=1:4,
       lwd=3,
       col=c("black","blue","green","red"),
       legend=c("truth","cubic p=10","cubic p=30","smoothing"))
```



The smoothing spline with $\nu = 30$ (in green) leads to overfitting. The smoothing splines obtained by cross-validation (in red) or by fixing $\nu = 10$ (in blue) are both good approximation of the truth. The corresponding effective degrees of freedom of the cross-validation solution can be retrieved from the model fit.

```
fit.smsp.cv$df
```

```
## [1] 6.458247
```

3.4 Lasso Regression

We have discussed Ridge regression and discussed its properties. Although Ridge regression can deal with high-dimensional data a disadvantage compared to subset- and stepwise regression is that it does not perform variable selection and therefore the interpretation of the final model is more challenging.

In Ridge regression we minimize RSS( ) given constraints on the so-called *L2-norm* of the regression coefficients

$$\|\beta\|_2^2 = \sum_{j=1}^{p} \beta_j^2 \leq c.$$

Another very popular approach in high-dimensional statistics is *Lasso regression* (Lasso=least absolute shrinkage and selection operator). The Lasso works very similarly. The only difference is that constraints are imposed on the *L1-norm* of the coefficients

$$\|\beta\|_1 = \sum_{j=1}^{p} |\beta_j| \leq c.$$

Therefore the Lasso is referred to as L1 regularization. The change in the form of the constraints (L2 vs L1) has important implications. Figure 3.8 illustrates the geometry of the Lasso optimization. Geometrically the Lasso constraint is a diamond with "corners" (the Ridge constraint is a circle). If the sum of squares "hits" one of these corners then the coefficient corresponding to the axis is shrunk to zero. As p increases, the multidimensional diamond has an increasing number of corners, and so it is highly likely that some coefficients will be set to zero. Hence, the Lasso performs not only shrinkage but it also sets some coefficients to zero, in other words the Lasso simultaneously performs variable selection. A disadvantage of the "diamond" geometry is that in general there is no closed form solution for the Lasso (the Lasso optimisation problem is not differentiable at the corners of the diamond).

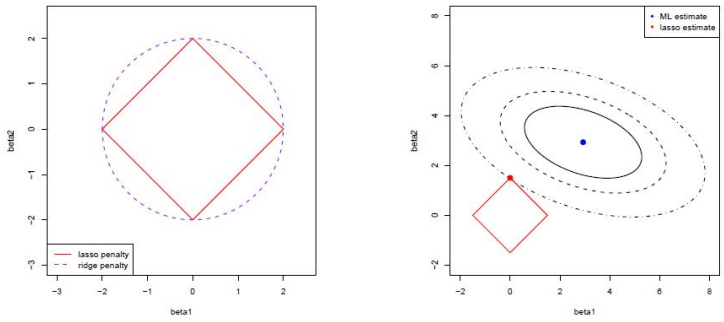


Figure 3.8: Geometry of Lasso regression.

Similar to Ridge regression the Lasso can be formulated as a penalisation problem

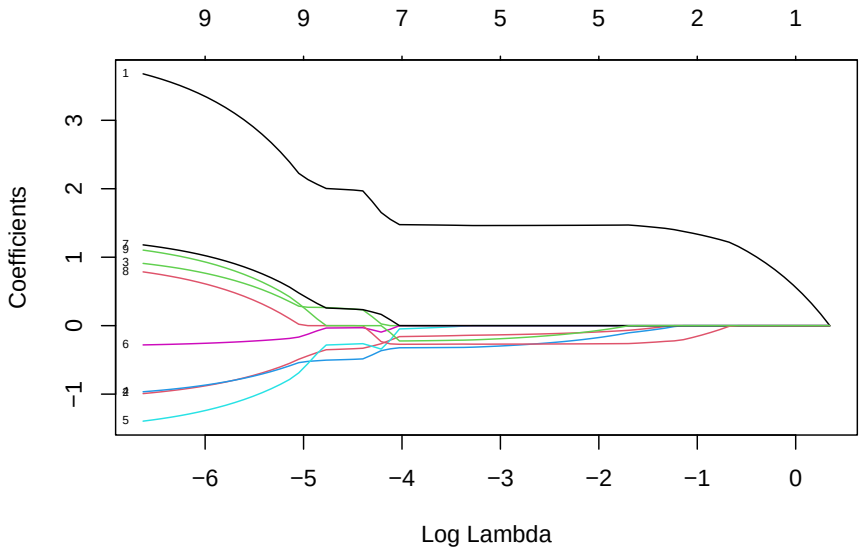
$$\hat{\beta}_{\lambda}^{\text{Lasso}} = \arg\min_{\beta} \ \text{RSS}(\beta) + \lambda \|\beta\|_1.$$

To fit the Lasso we use `glmnet` (with $\alpha = 1$).

```
fit.lasso.glmnet <-glmnet(x=xtrain,y=ytrain,alpha=1)
```

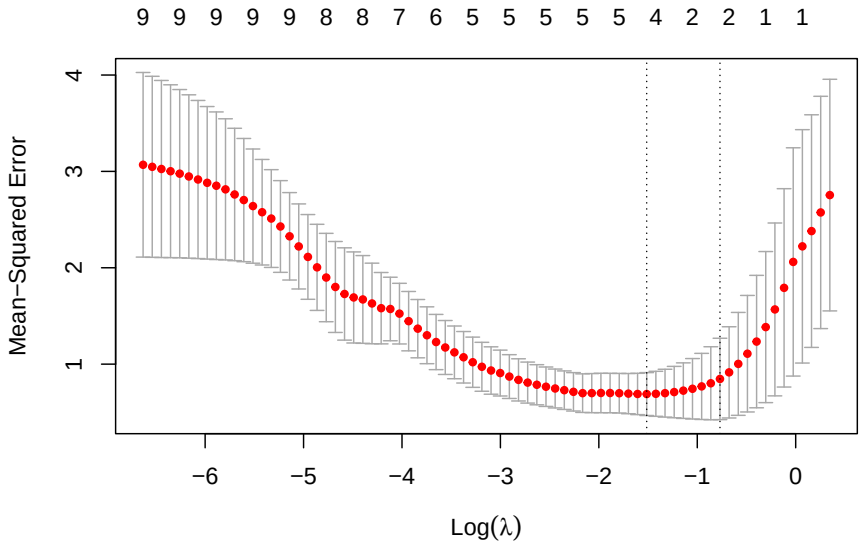
The following figure shows the Lasso solution for a grid of λ values. We note that the Lasso shrinks some coefficients to exactly zero.

```
plot(fit.lasso.glmnet,xvar="lambda",label=TRUE)
```



We choose the optimal tuning parameter λ_{opt} by cross-validation.

```
cv.lasso.glmnet <-cv.glmnet(x=xtrain,y=ytrain,alpha=1)
plot(cv.lasso.glmnet)
```



```
cv.lasso.glmnet$lambda.min
```

```
## [1] 0.2201019
```

The coefficient for the optimal model can be extracted using the `coef` function.

```
beta.lasso <- coef(fit.lasso.glmnet, s = cv.lasso.glmnet$lambda.min)
names(beta.lasso) <- colnames(xtrain)
beta.lasso
```

```
## 10 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  0.08727244
## V1           1.44830414
## V2          -0.04302609
## V3           .
## V4          -0.07325330
## V5           .
## V6           .
## V7           .
## V8          -0.24778236
## V9           .
```

We now discuss some properties of the Lasso.

3.4.1 Numerical optimization and soft thresholding

In general there is no closed-form solution for the Lasso. The optimization has to be performed numerically. An efficient algorithm is implemented in `glmnet` and is referred to as "Pathwise Coordinate Optimization". The algorithm updates one regression coefficient at a time using the so-called soft-thresholding function. This is done iteratively until some convergence criterion is met.

An exception is the case with an orthonormal design matrix $\mathbf{X}$, i.e. $\mathbf{X^T X} = \mathbf{I}$. Under this assumption we have

$$\begin{aligned} \text{RSS}(\beta) &= (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta) \\ &= \mathbf{y}^T\mathbf{y} - 2\beta^T\hat{\beta}^{\text{OLS}} + \beta^T\hat{\beta} \end{aligned}$$

and therefore the Lasso optimization reduces to $j = 1, \ldots, p$ univariate problems

$$\text{minimize } -\hat{\beta}_j^{\text{OLS}}\beta_j + 0.5\beta_j^2 + 0.5\lambda|\beta_j|.$$

In the exercises we will show that the solution is

$$\begin{aligned}\hat{\beta}_{\lambda,j}^{\text{Lasso}} &= \text{sign}(\hat{\beta}_j^{\text{OLS}})\left(|\hat{\beta}_j^{\text{OLS}}| - 0.5\lambda\right)_+ \\ &= \begin{cases} \hat{\beta}_j^{\text{OLS}} - 0.5\lambda & \text{if } \hat{\beta}_j^{\text{OLS}} > 0.5\lambda \\ 0 & \text{if } |\hat{\beta}_j^{\text{OLS}}| \leq 0.5\lambda \\ \hat{\beta}_j^{\text{OLS}} + 0.5\lambda & \text{if } \hat{\beta}_j^{\text{OLS}} < -0.5\lambda \end{cases}\end{aligned}$$

That is, in the orthonormal case, the Lasso is a function of the OLS estimator. This function, depicted in the next figure, is referred to as *soft-thresholding*.

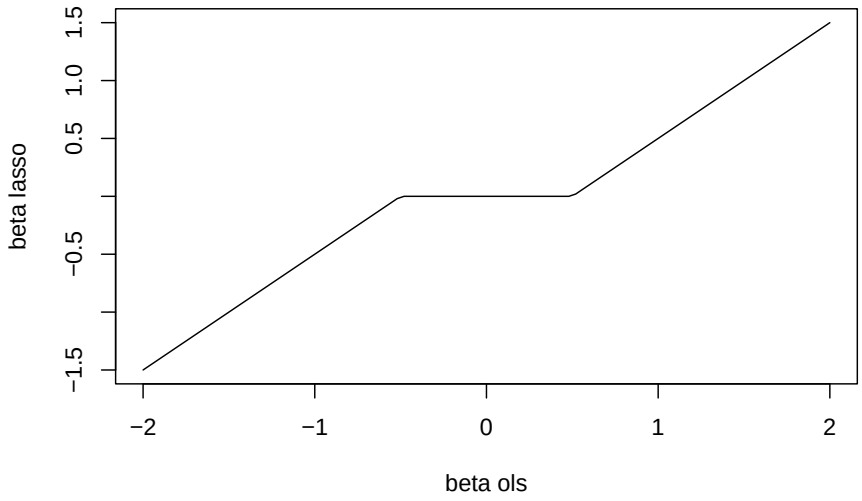


Figure 3.9: Soft-thresholding function.

The soft-thresholding function is not only used for numerical optimization of the Lasso but also plays a role in wavelet thresholding used for signal and image denoising.

3.4.2 Variable selection

We have seen that the Lasso simultaneously shrinks coefficients and sets some of them to zero. Therefore the Lasso performs variable selection which leads to

more interpretabel models (compared to Ridge regression). For the Lasso we can define the set of selected variables

$$\widehat{S}_\lambda^{\text{Lasso}} = \{j \in (1, \dots, p); \widehat{\beta}_{\lambda,j}^{\text{Lasso}} \neq 0\}$$

In our example this set can be obtained as follows.

```
Shat <- rownames(beta.lasso)[which(beta.lasso != 0)]
Shat
```

```
## [1] "(Intercept)" "V1"          "V2"          "V4"          "V8"
```

An interesting question is whether the Lasso does a good or bad job in variable selection. That is, does $\widehat{S}_\lambda^{\text{Lasso}}$ tend to agree with the true set of active variables S_0? Or, does the Lasso typically under- or over-select covariates? These questions are an active field of statistical research.

3.4.3 Elasticnet Regression

We have encountered the L1 and L2 penalty. The Lasso (L1) penalty has the nice property that it leads to sparse solutions, i.e. it simultaneously performs variable selection. A disadvantage is that the Lasso penalty is somewhat indifferent to the choice among a set of strong but correlated variables. The Ridge (L2) penalty, on the other hand, tends to shrink the coefficients of correlated variables toward each other. An attempt to take the best of both worlds is the *elastic net* penalty which has the form

$$\lambda\Big(\alpha\|\beta\|_1 + (1-\alpha)\|\beta\|_2^2\Big).$$

The second term encourages highly correlated features to be averaged, while the first term encourages a sparse solution in the coefficients of these averaged features.

In `glmnet` the elastic net regression is implemented using the mixing parameter α. The default is $\alpha = 1$, i.e. the Lasso.

3.5 Diabetes example

We now review what we have learned with an example. The data that we consider consist of observations on 442 patients, with the response of interest being a quantitative measure of disease progression one year after baseline. There are ten baseline variables — age, sex, body-mass index, average blood

pressure, and six blood serum measurements — plus quadratic terms, giving a total of $p = 64$ features. The task for a statistician is to construct a model that predicts the response Y from the covariates. The two hopes are, that the model would produce accurate baseline predictions of response for future patients, and also that the form of the model would suggest which covariates were important factors in disease progression.

We start by splitting the data into training and test data.

```
diabetes <- readRDS(file="data/diabetes.rds")
data <- as.data.frame(cbind(y=diabetes$y,diabetes$x2))
colnames(data) <- gsub(":",".",colnames(data))
train_ind <- sample(seq(nrow(data)),size=nrow(data)/2)
data_train <- data[train_ind,]
xtrain <- as.matrix(data_train[,-1])
ytrain <- data_train[,1]
data_test <- data[-train_ind,]
xtest <- as.matrix(data_test[,-1])
ytest <- data_test[,1]
```

We perform forward stepwise regression.

```
# Forward regression
fit0 <- lm(y~1,data=data_train)
up.model <- paste("~",
                  paste(
                    colnames(data_train[,-1]),collapse=" + ")
                  )
fit.fw <- stepAIC(fit0,direction="forward",
                  scope=list(lower=fit0,
                             upper=up.model

                  ),
                  trace = FALSE
)
#summary(fit.fw)
```

The selection process is depicted in the following table.

```
kable(as.data.frame(fit.fw$anova),digits=2,
      booktabs=TRUE)
```

Step	Df	Deviance	Resid. Df	Resid. Dev	AIC
	NA	NA	220	1262297.5	1913.71
+ bmi	1	434735.33	219	827562.1	1822.40
+ ltg	1	155835.95	218	671726.2	1778.30
+ age.sex	1	47106.62	217	624619.6	1764.23
+ map	1	29740.28	216	594879.3	1755.45
+ bmi.glu	1	22952.37	215	571926.9	1748.75
+ hdl	1	19077.03	214	552849.9	1743.25
+ sex	1	15702.72	213	537147.2	1738.89
+ hdl.tch	1	9543.83	212	527603.3	1736.92
+ sex.ldl	1	5735.62	211	521867.7	1736.51
+ tch.ltg	1	6279.00	210	515588.7	1735.83
+ age.map	1	5342.10	209	510246.6	1735.53

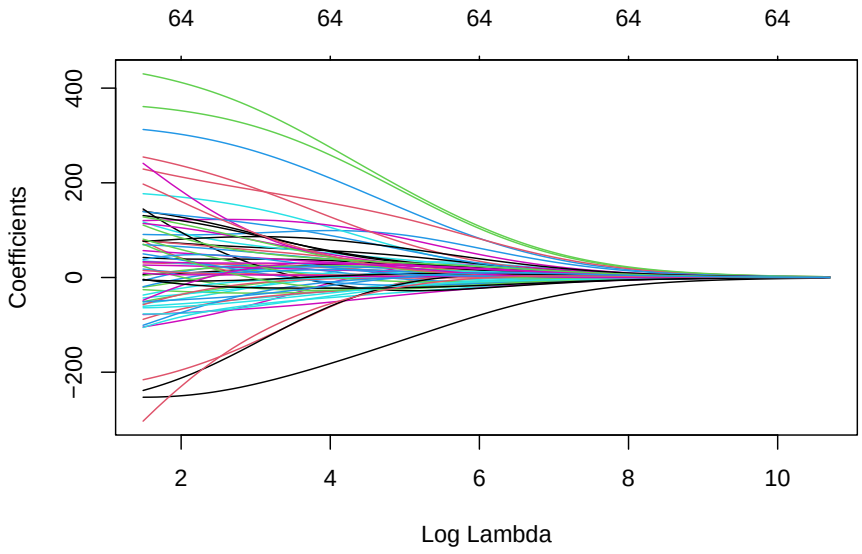
The regression coefficients and the corresponding statistics of the AIC-optimal model are shown next.

```
kable(broom::tidy(fit.fw),digits=2,
      booktabs=TRUE)
```

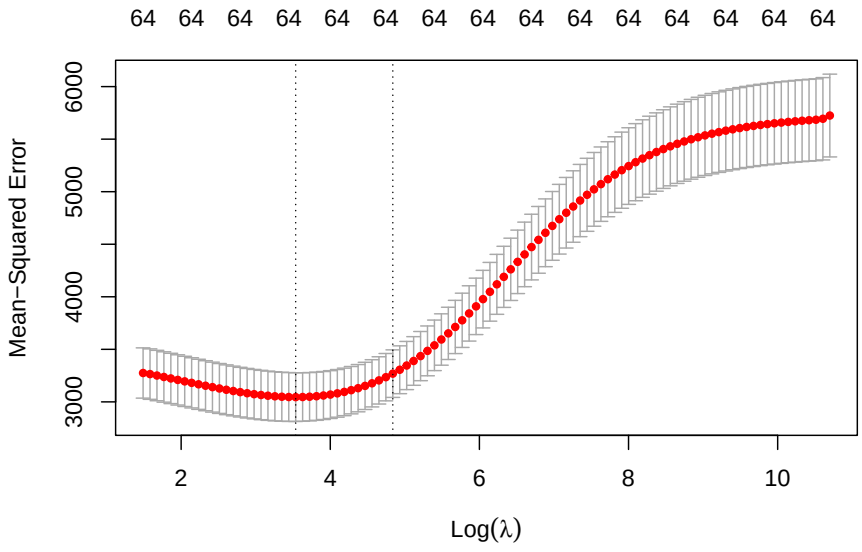
term	estimate	std.error	statistic	p.value
(Intercept)	155.72	3.36	46.29	0.00
bmi	466.07	81.82	5.70	0.00
ltg	497.33	94.05	5.29	0.00
age.sex	274.22	76.35	3.59	0.00
map	315.78	80.98	3.90	0.00
bmi.glu	206.59	74.57	2.77	0.01
hdl	-392.14	94.40	-4.15	0.00
sex	-201.94	80.87	-2.50	0.01
hdl.tch	-210.17	87.81	-2.39	0.02
sex.ldl	118.77	74.81	1.59	0.11
tch.ltg	-146.12	89.83	-1.63	0.11
age.map	119.49	80.78	1.48	0.14

We continue by fitting Ridge regression. We show the trace plot and the cross-validation plot.

```
# Ridge
set.seed(1515)
fit.ridge <- glmnet(xtrain,ytrain,alpha=0)
fit.ridge.cv <- cv.glmnet(xtrain,ytrain,alpha=0)
plot(fit.ridge,xvar="lambda")
```

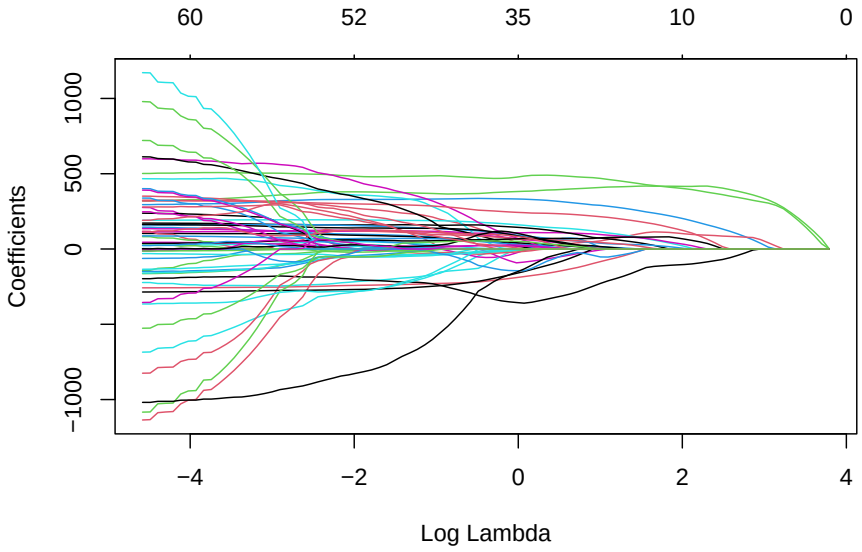



```
plot(fit.ridge.cv)
```

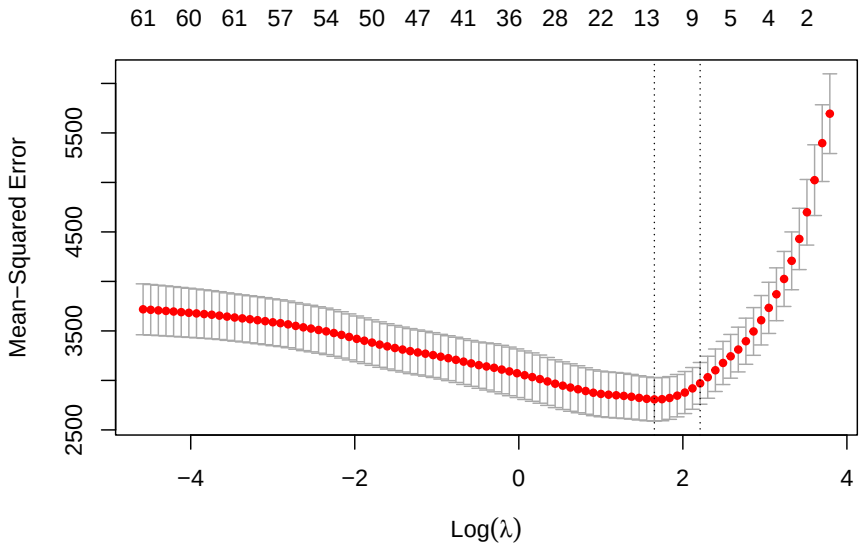


Finally, we run the Lasso approach and show the trace and the cross-validation plots.

```
# Lasso
set.seed(1515)
fit.lasso <- glmnet(xtrain,ytrain,alpha=1)
fit.lasso.cv <- cv.glmnet(xtrain,ytrain,alpha=1)
plot(fit.lasso,xvar="lambda")
```




```
plot(fit.lasso.cv)#fit.lasso.cv$lambda.1se
```

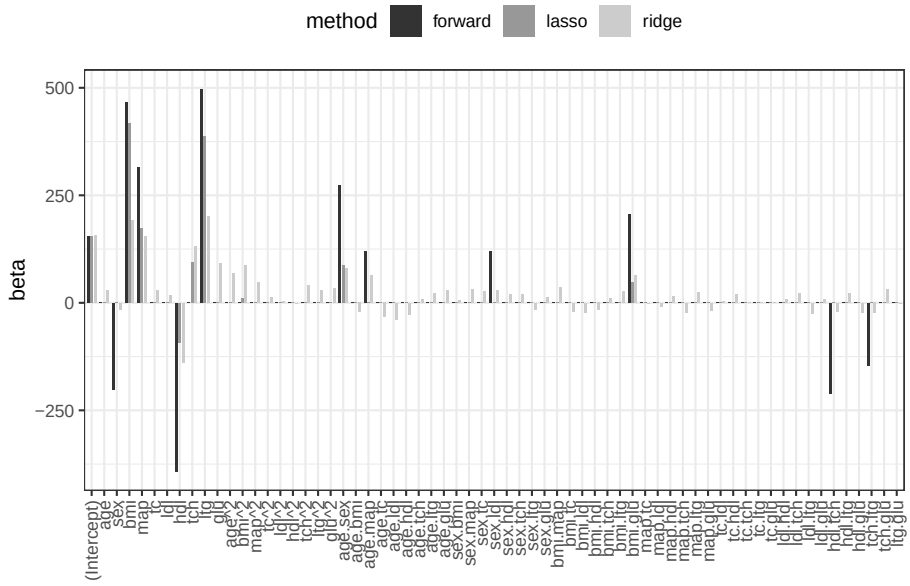


We calculate the root-mean-square errors (RMSE) on the test data and compare with the full model.

```
# Full model
fit.full <- lm(y~.,data=data_train)
# RMSE
pred.full <- predict(fit.full,newdata=data_test)
pred.fw <- predict(fit.fw,newdata=data_test)
pred.ridge <- as.vector(predict(fit.ridge,newx=xtest,s=fit.ridge.cv$lambda.1se))
pred.lasso <- as.vector(predict(fit.lasso,newx=xtest,s=fit.lasso.cv$lambda.1se))
res.rmse <- data.frame(
  method=c("full","forward","ridge","lasso"),
  rmse=c(RMSE(pred.full,ytest),RMSE(pred.fw,ytest),
         RMSE(pred.ridge,ytest),RMSE(pred.lasso,ytest)))
kable(res.rmse,digits = 2,
      booktabs=TRUE)
```

method	rmse
full	84.51
forward	59.89
ridge	62.63
lasso	58.47

The Lasso has the lowest generalization error (RMSE). We plot the regression coefficients for all 3 methods.



We point out that the same analysis can be conducted with the `caret` package. The code to do so is provided next.

```
## Setup trainControl: 10-fold cross-validation
tc <- trainControl(method = "cv", number = 10)

## Ridge
lambda.grid <- fit.ridge.cv$lambda
fit.ridge.caret<-train(x=xtrain,
                       y=ytrain,
                       method = "glmnet",
                       tuneGrid = expand.grid(alpha = 0,
                                              lambda=lambda.grid),
                       trControl = tc
)

# CV curve
plot(fit.ridge.caret)
# Best lambda
fit.ridge.caret$bestTune$lambda
# Model coefficients
coef(fit.ridge.caret$finalModel,fit.ridge.cv$lambda.1se)%>%head
# Make predictions
fit.ridge.caret %>% predict(xtest,s=fit.ridge.cv$lambda.1se)%>%head
```

```
## Lasso
lambda.grid <- fit.lasso.cv$lambda
fit.lasso.caret<-train(x=xtrain,
                       y=ytrain,
                       method = "glmnet",
                       tuneGrid = expand.grid(alpha = 1,
                                              lambda=lambda.grid),
                       trControl = tc
)

# CV curve
plot(fit.lasso.caret)
# Best lambda
fit.lasso.caret$bestTune$lambda
# Model coefficients
coef(fit.lasso.caret$finalModel,
     fit.lasso.caret$bestTune$lambda)%>%head
# Make predictions
fit.lasso.caret%>%predict(xtest,
                          s=fit.ridge.cv$lambda.1se)%>%head

## Compare Ridge and Lasso
models <- list(ridge= fit.ridge.caret,lasso = fit.lasso.caret)
resamples(models) %>% summary( metric = "RMSE")
```

Chapter 4

Classification

Our high-dimensional considerations so far focused on the linear regression model. We now extend this to classification.

4.1 Logistic Regression

We start with standard logistic regression where the response Y takes values 0 and 1, and the aim is to do prediction based on covariates $X = (X_1, \dots, X_p)$. We model the probability of success (i.e., $Y = 1$)

$$p(x; \beta) = P(Y = 1 | X = x; \beta)$$

assuming a binomial distribution with logit link function

$$\text{logit}(x; \beta) = \log\Big(\frac{p(x; \beta)}{1 - p(x; \beta)}\Big) = X^T \beta.$$

In logistic regression we estimate the regression parameter β by maximizing the log-likelihood

$$\begin{aligned} \ell(\beta | \mathbf{y}, \mathbf{X}) &= \sum_{i=1}^{n} (1 - y_i) \log(1 - p(x_i; \beta)) + y_i \log p(x_i; \beta) \\ &= \sum_{i=1}^{n} y_i x_i^T \beta - \log(1 + \exp(x_i^T \beta)). \end{aligned}$$

Given an estimate $\widehat{\beta}$ (from training data), prediction based on new input data X_{new} can be obtained via the predicted probability of success $p(X_{\text{new}}; \widehat{\beta})$, e.g. the class labels corresponding to the maximum probability

$$\hat{Y} \equiv \hat{G}(X_{\text{new}}) \equiv \begin{cases} 1, & \text{if } p(X_{\text{new}}; \hat{\beta}) > 0.5. \\ 0, & \text{otherwise.} \end{cases}$$

There are different measures to judge the quality of the predictions. We focus on the misclassification error which is simply the fraction of misclassified test samples. Another important measure used in the context of classification is the receiver operating characteristic (ROC).

We illustrate logistic regression using an example taken from the book by Hastie et al. (2001). The data shown in Figure 4.1 are a subset of the Coronary Risk-Factor Study (CORIS) baseline survey, carried out in three rural areas of the Western Cape, South Africa. The aim of the study was to establish the intensity of ischemic heart disease risk factors in that high-incidence region. The data represent white males between 15 and 64, and the response variable is the presence or absence of myocardial infarction (MI) at the time of the survey (the overall prevalence of MI was 5.1% in this region). There are 160 cases in our data set, and a sample of 302 controls. The variables are:

- sbp: systolic blood pressure
- tobacco: cumulative tobacco (kg)
- ldl: low densiity lipoprotein cholesterol adiposity
- famhist: family history of heart disease (Present, Absent)
- obesity
- alcohol: current alcohol consumption
- age: age at onset
- chd: response, coronary heart disease.

```
dat <- readRDS(file="data/sahd.rds")
pairs(data.matrix(dat[,-1]),
      col=ifelse(dat$chd==1,"red","blue"),
      pch=ifelse(dat$chd==1,1,2))
```

We first fit a logistic regression model using the function `glm`.

```
fit.logistic <- glm(chd~sbp+tobacco+ldl+famhist+obesity+alcohol+age,
                    data=dat,
                    family="binomial")
kable(broom::tidy(fit.logistic),digits=3,booktabs=TRUE)
```

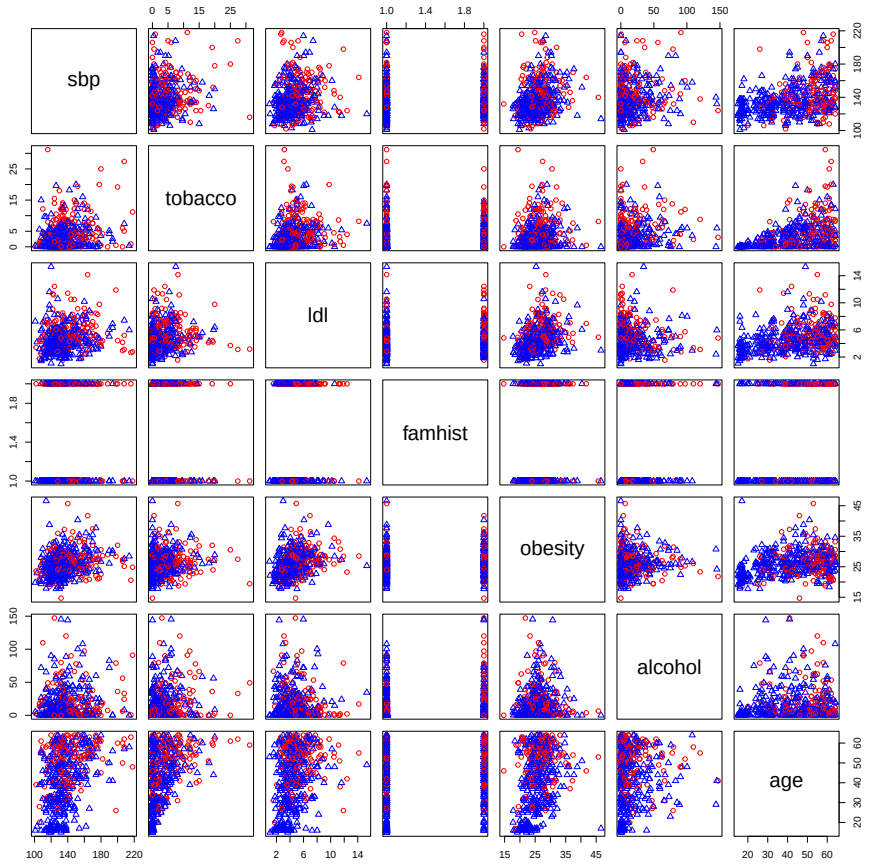


Figure 4.1: Pairs plot of South African Heart Disease Data (red circles: cases, blue triangles: controls).

term	estimate	std.error	statistic	p.value
(Intercept)	-4.130	0.964	-4.283	0.000
sbp	0.006	0.006	1.023	0.306
tobacco	0.080	0.026	3.034	0.002
ldl	0.185	0.057	3.219	0.001
famhistPresent	0.939	0.225	4.177	0.000
obesity	-0.035	0.029	-1.187	0.235
alcohol	0.001	0.004	0.136	0.892
age	0.043	0.010	4.181	0.000

There are some surprises in this table of coefficients, which must be interpreted with caution. Systolic blood pressure (sbp) is not significant! Nor is obesity, and its sign is negative. This confusion is a result of the correlation between the set of predictors. On their own, both sbp and obesity are significant, and with positive sign. However, in the presence of many other correlated variables, they are no longer needed (and can even get a negative sign).

How does one interpret a coefficient of 0.080 (Std. Error = 0.026) for tobacco, for example? Tobacco is measured in total lifetime usage in kilograms, with a median of 1.0kg for the controls and 4.1kg for the cases. Thus an increase of 1kg in lifetime tobacco usage accounts for an increase in the odds of coronary heart disease of exp(0.080)=1.083 or 8.3%. Incorporating the standard error we get an approximate 95% confidence interval of exp($0.081 \pm 2 \times 0.026$)=(1.035,1.133).

4.2 Regularized Logistic Regression

Similar as for linear regression, in the high-dimensional setting where n is small compared to p, the maximum likelihood estimator does lead to overfitting and a poor generalisation error. In the context of linear regression we introduced regularization by imposing constraints on the regression coefficients. It is easy to generalize these approaches to logistic regression. In `R` subset- and stepwise logistic regression is implemented in `stepAIC` and elastic net regularization in `glmnet` (with argument `family="binomial"`). In the latter case the algorithm optimizes the negative log-likelihood penalized with the elastic net term:

$$\widehat{\beta}^{\text{EN}}_{\alpha,\lambda} = \arg\min_{\beta} \; -\frac{1}{n}\ell(\beta|\mathbf{y},\mathbf{X}) + \lambda(\alpha\|\beta\|_1 + (1-\alpha)\|\beta\|_2^2/2).$$

With $\alpha = 0$ and $\alpha = 1$ we obtain the Ridge and the Lasso solution, respectively.

We turn back to the heart disease example and perform backward stepwise logistic regression.

```
fit.bw <- stepAIC(fit.logistic,direction = "backward",trace=FALSE)
```

The terms removed in each step are provided in the next table.

```
kable(as.data.frame(fit.bw$anova),digits=3,booktabs=TRUE)
```

Step	Df	Deviance	Resid. Df	Resid. Dev	AIC
	NA	NA	454	483.174	499.174
- alcohol	1	0.019	455	483.193	497.193
- sbp	1	1.104	456	484.297	496.297
- obesity	1	1.147	457	485.444	495.444

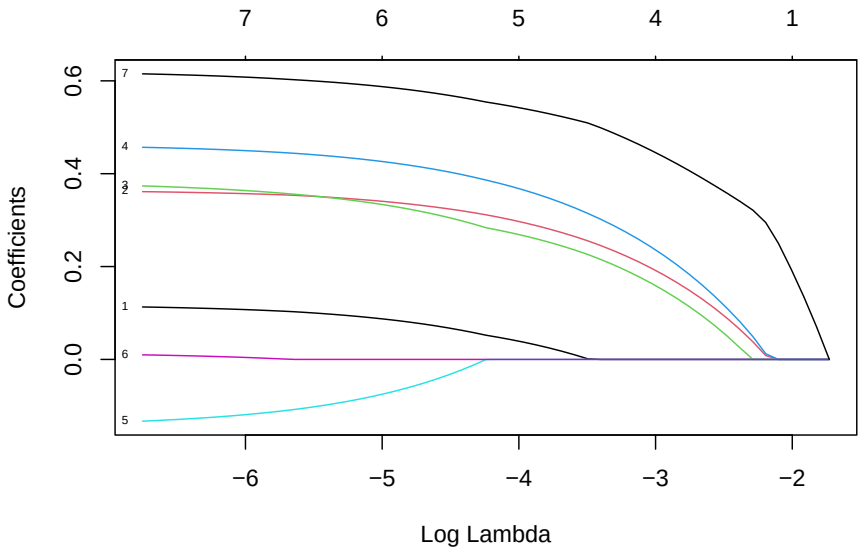
The regression coefficients of the final model are shown below.

```
kable(broom::tidy(fit.bw),digits=3,booktabs=TRUE)
```

term	estimate	std.error	statistic	p.value
(Intercept)	-4.204	0.498	-8.437	0.000
tobacco	0.081	0.026	3.163	0.002
ldl	0.168	0.054	3.093	0.002
famhistPresent	0.924	0.223	4.141	0.000
age	0.044	0.010	4.521	0.000

We continue with the Lasso approach and show the trace plot.

```
x <- scale(data.matrix(dat[,-1]))
y <- dat$chd
fit.lasso <- glmnet(x=x,y=y,family="binomial")
plot(fit.lasso,xvar = "lambda",label=TRUE)
```

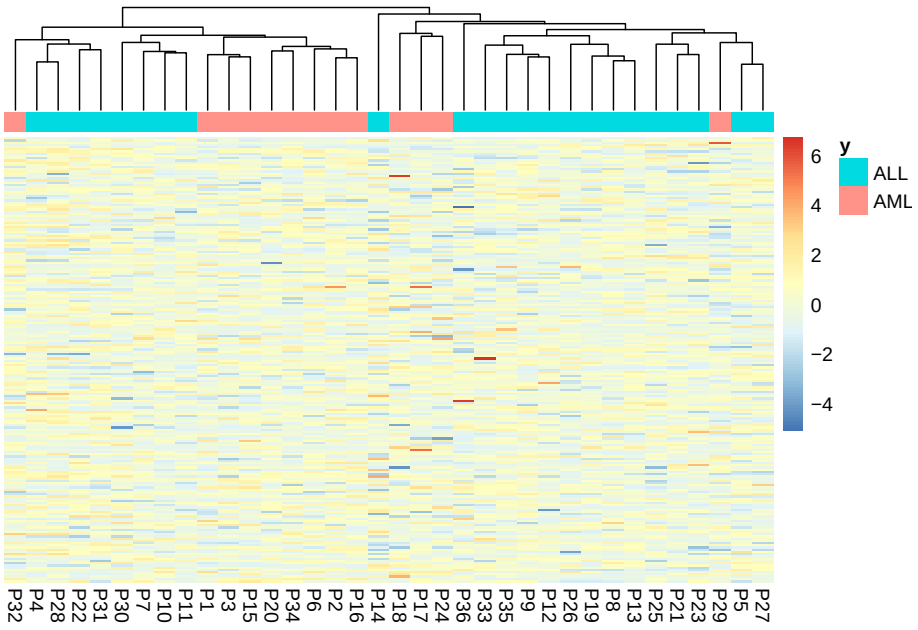


The first coefficient which is shrunk to zero is alcohol followed by sbp and obesity. This is in line with the results from backward selection.

We now turn to a truly high-dimensional example. The data consists of expression levels recorded for 3′571 genes in 72 patients with leukemia. The binary outcome encodes the disease subtype: acute lymphobastic leukemia (ALL) or acute myeloid leukemia (AML). The data are represented as a 72 x 3,571 matrix $\mathbf{X}$ of gene expression values, and a vector $\mathbf{y}$ of 72 binary disease outcomes. We first create training and test data.

```
# set seed
set.seed(15)

# get leukemia data
leukemia <- readRDS(file="data/leukemia.rds")
x <- leukemia$x
y <- leukemia$y

# test/train
ind_train <- sample(1:length(y),size=length(y)/2)
xtrain <- x[ind_train,]
ytrain <- y[ind_train]
xtest <- x[-ind_train,]
ytest <- y[-ind_train]
```

The following heatmap illustrates the gene expression values for the different patients.

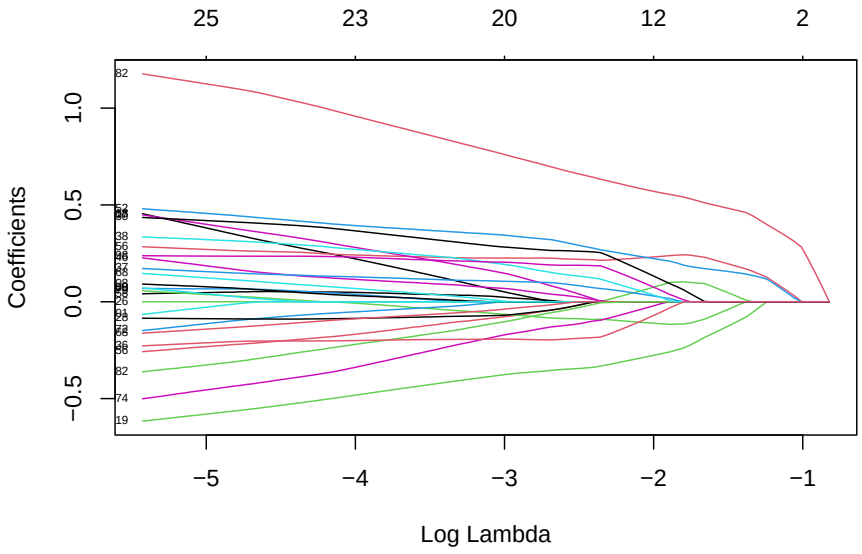


We now run the elastic net logistic regression approach.

```
# run glmnet
alpha  <- 0.95                  # elastic net mixing parameter.
fit.glmnet <-glmnet(xtrain,ytrain,family = "binomial",alpha=alpha)
```

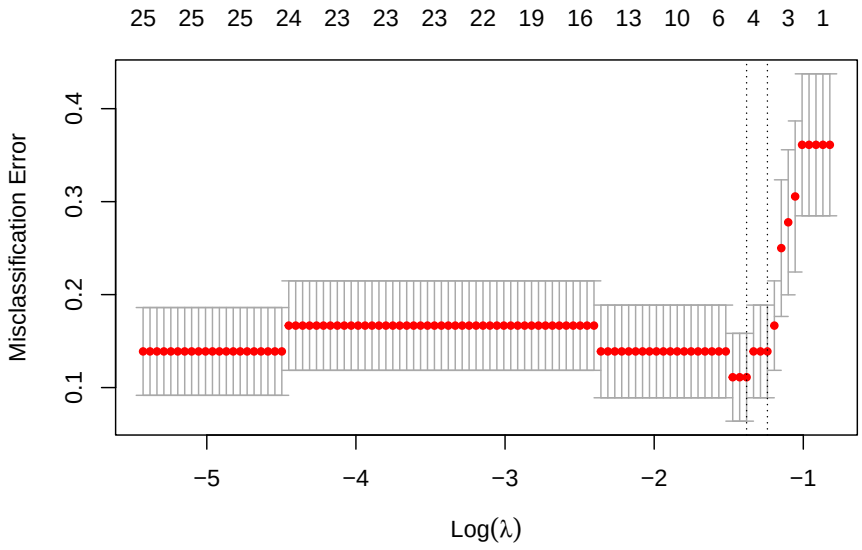
The following Figure shows the trace plot.

```
plot(fit.glmnet,xvar="lambda",label=TRUE)
```



We run 10-fold cross-validation and show the misclassification error.

```
set.seed(118)
# run cv.glmnet
nfolds <- 10 # number of cross-validation folds.
cv.glmnet <- cv.glmnet(xtrain,ytrain,
                       family = "binomial",type.measure = "class",
                       alpha = alpha,nfolds = nfolds)
plot(cv.glmnet)
```



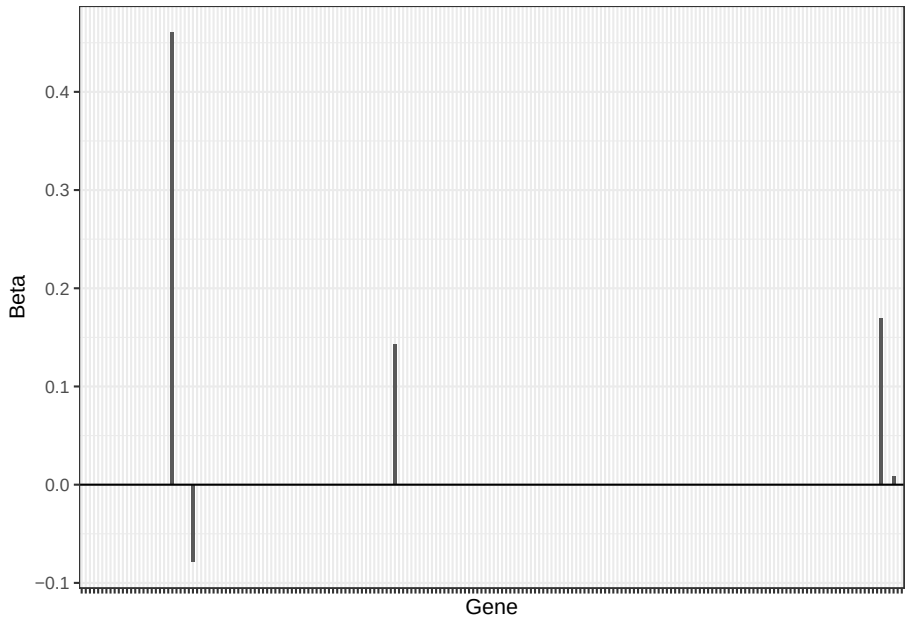
We take `lambda.1se` as the optimal tuning parameter.

```
#sum(coef(fit.glmnet, s = cv.glmnet$lambda.1se)!=0)
#sum(coef(fit.glmnet, s = cv.glmnet$lambda.min)!=0)
(lambda.opt <- cv.glmnet$lambda.1se)
```

```
## [1] 0.2891844
```

We extract the coefficients and plot them as a barplot.

```
beta.glmnet <- coef(fit.glmnet, s = cv.glmnet$lambda.min)
df <- data.frame(Gene=paste0("G",1:(length(beta.glmnet)-1)),
                 Beta=as.numeric(beta.glmnet)[-1])
df%>%
  arrange(desc(abs(Beta)))%>%
  slice(1:200)%>%
  ggplot(.,aes(x=Gene,y=Beta))+
  geom_bar(stat="identity")+
  theme_bw()+
  theme(axis.text.x = element_blank())+
  geom_hline(yintercept = 0)
```



Finally, we predict the disease outcome of the test samples using the fitted model and compare against the observed outcomes of the test samples.

```
pred <- c(predict(fit.glmnet,xtest,s = lambda.opt,type = "class"))
print(table(true = factor(ytest),pred = factor(pred)))
```

```
##     pred
## true  0  1
##    0 24  0
##    1  3  9
```

The misclassification error on the test data can be calculated as follows.

```
round(mean(pred!=ytest),3)
```

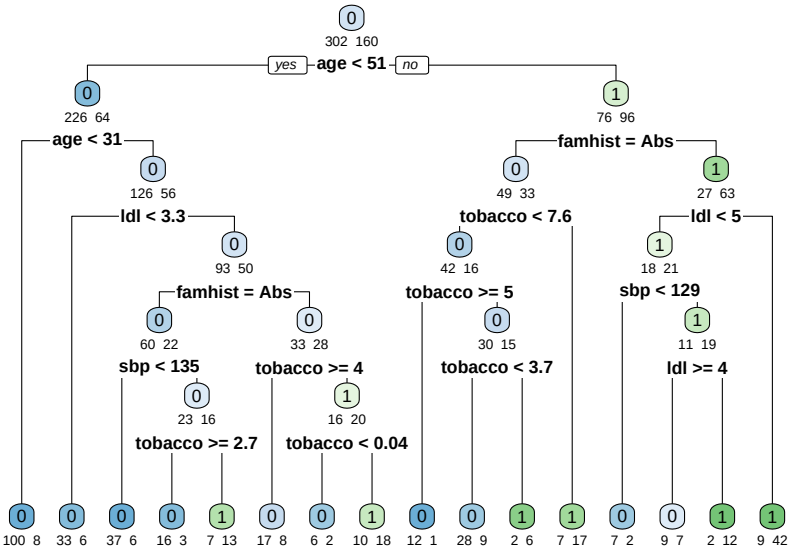
```
## [1] 0.083
```

4.3 Classification Trees and Machine Learning

Classification is a frequent task in data mining and besides logistic regression there is a variety of other methods developed for this task. We first introduce

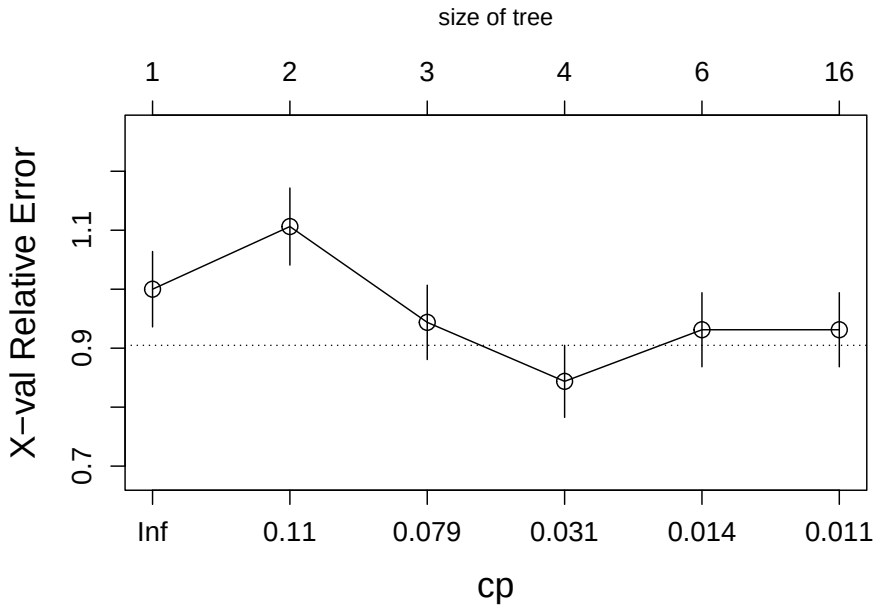
classification trees which learn a binary tree where *leaves* represent class labels and *branches* represent conjunctions of features that lead to those class labels. The package `rpart` can be used to learn classification trees.

```
# load packages
library(rpart)
library(rpart.plot)

# read south african heart disease data
dat <- readRDS(file="data/sahd.rds")

# grow a classification tree
fit.tree <- rpart(chd~.,data=dat,method="class")
# plot(fit.tree, uniform=TRUE)
# text(fit.tree, use.n=TRUE, all=TRUE, cex=.8)
rpart.plot(fit.tree,extra=1,under=TRUE,tweak = 1.2,faclen=3)
```



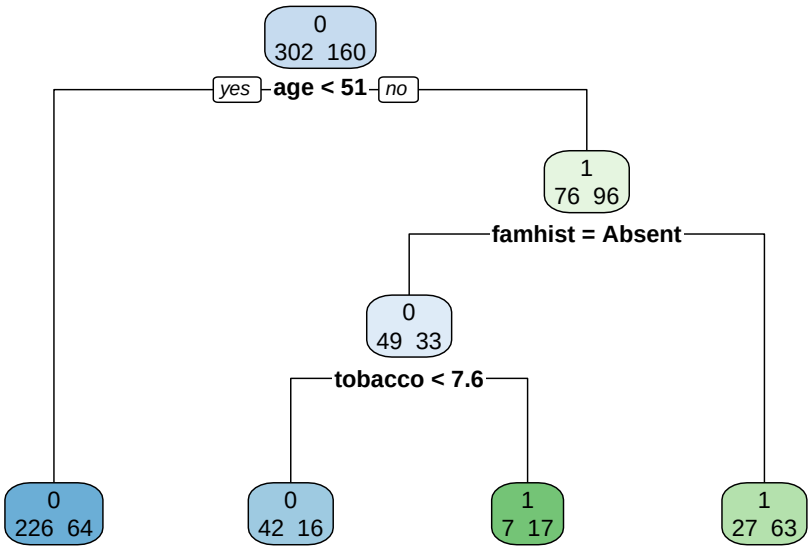
The algorithm starts by growing a typically too large tree which overfits the data. The next step is to "prune" the tree to obtain a good trade-off between goodness of fit and complexity. The following plot shows the relative cross-validation error (relative to the trivial tree consisting of only the root node) as a function of the complexity parameter.

```
plotcp(fit.tree,cex.lab=1.5,cex.axis=1.2,cex=1.5)
```



The optimally pruned tree has size 4 (i.e., 4 leave nodes).

```
# prune the tree
fit.prune<- prune(fit.tree,
                  cp=fit.tree$cptable[which.min(fit.tree$cptable[,"xerror"]),"CP"])
rpart.plot(fit.prune,extra=1)
```



Classification trees are the basis of powerful Machine Learning (ML) algorithms, namely *Random Forest* and *AdaBoost*. Both methods share the idea of growing various trees and combining the outputs to obtain a more powerful classification. However, the two approaches differ in the way they grow the trees and in how they do the aggregation. Random Forest works by building trees based on bootstrapped data sets and by aggregating the results using a majority vote (see Figure 4.2). The key idea behind AdaBoost is to sequentially fit a "stump" (i.e., a tree with two leaves) to weighted data with repeatedly modified weights. The weights assure that each stump takes the errors made by the previous stump into account. In that way a sequence of weak classifiers $\{G_m\}_{m=1}^{M}$ is produced and finally a powerful new classifier is obtained by giving more influence towards the more accurate classifiers (see Figure 4.3). More details on these methods are provided in the book by Hastie et al. (2001). Random Forest is implemented in the package `RandomForest` and AdaBoost in the package `gbm`. We will explore examples in the exercises.

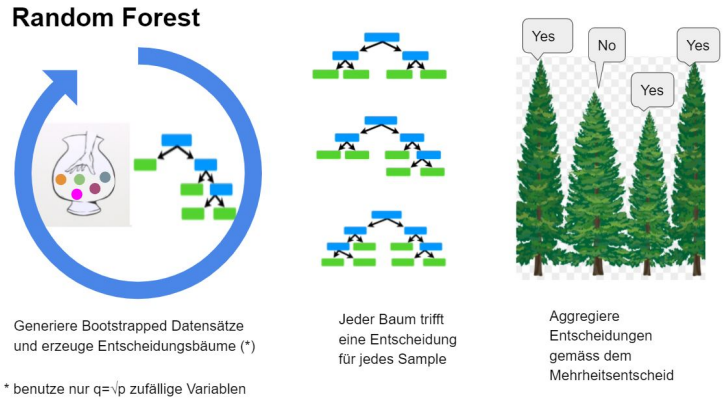


Figure 4.2: The key idea of Random Forest

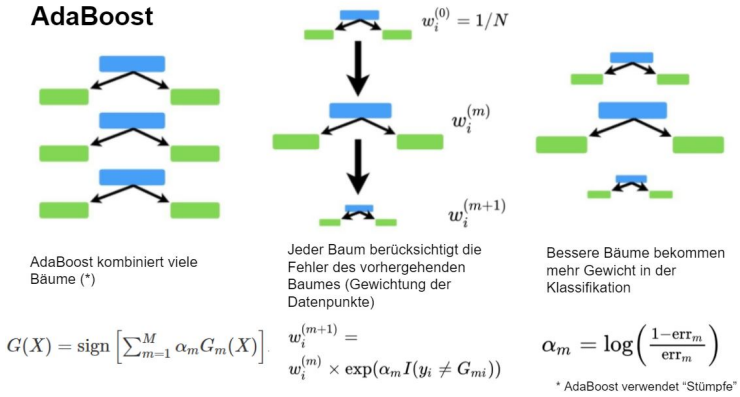


Figure 4.3: The key idea of AdaBoost

Chapter 5

Survival Analysis

We turn our attention to survival analysis which deals with so-called time-to-event endpoints. We will use the *lymphoma* data set to set the scene and explain the basics. In particular, we will discuss elastic net regularization in the context of cox regression, introduce the time-dependent Brier score as a measure of prediction accuracy, and we give an example on how to use the `pec` package to benchmark prediction algorithms.

5.1 Survival Endpoints and Cox Regression

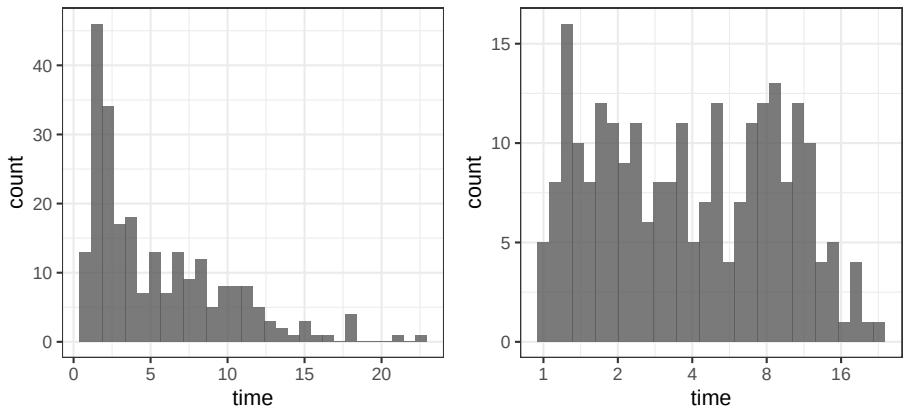
We start by reading the lymphoma data which consists of gene expression data for $p = 7399$ genes measured on $n = 240$ patients, as well as survival data, for these patients.

```
# read gene expression matrix
x <- read.table("data/lymphx.txt")%>%
  as.matrix

# read survival data
y <- read.table("data/lymphtime.txt",header = TRUE)%>%
  as.matrix
```

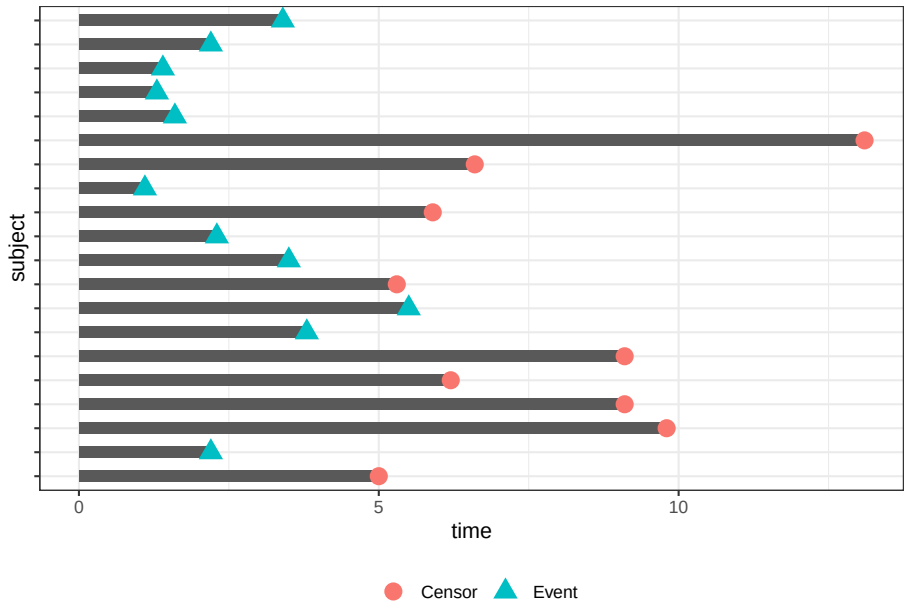
The survival data consists of two variables `time` (the survival time) and `status` (event status, 1 in case of death, 0 in case of censoring).

time	status
5.0	0
5.9	0
6.6	0
13.1	0
1.6	1
1.3	1

The next plots shows the distribution of the survival times on a linear and log-scale.



The distribution on the left is right skewed. However, after a log transformation the distribution looks near-to-symmetric. What makes this endpoint so special? Why can't we just use (regularized) linear regression to predict the (log) survival time based on the gene expression features? Such an approach would be shortsighted the reason being that we so far did not take into account the event status. The following graph shows survival times along side with the event status for a few patients. For patients with an event (blue triangles) the survival time equals the time-to-event. However, for censored patients (red dots) the actual time-to-event is not observed and will be larger than the survival time.



In survival analysis we denote the time-to-event with T. As illustrated above we typically only partially observe T as some subjects may be censored due to:

- Loss to follow-up
- Withdrawal from study
- No event by end of fixed study period.

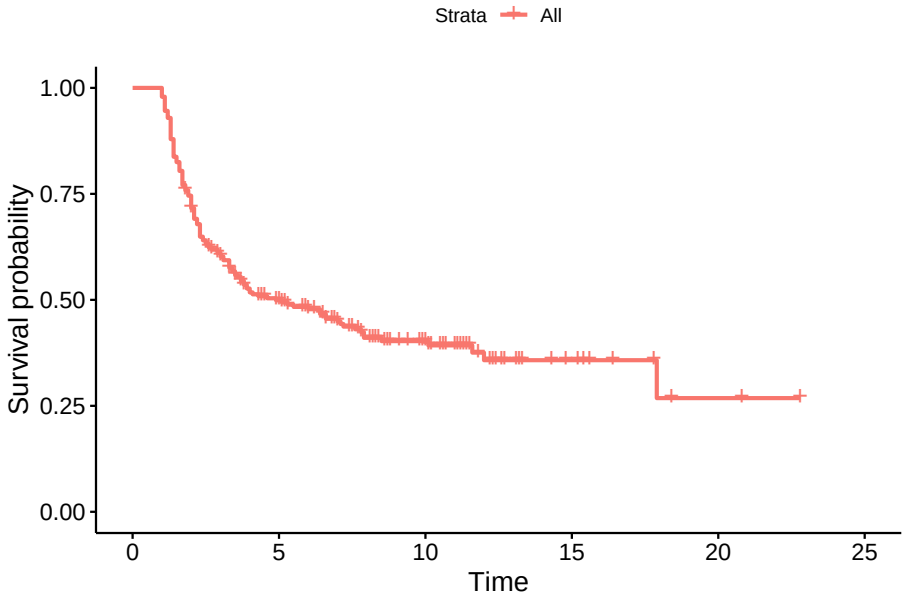
Therefore we observe the survival time Y (which equals the event time or the censoring time whichever occurs earlier) and the event status D ($D = 1$ in case of event, $D = 0$ in case of censoring).

A fundamental quantity in survival analysis is the survival function

$$S(t) = P(T > t) = 1 - F(t)$$

which can be estimated using the Kaplan-Meier method. In R we use `survfit` to invoke Kaplan-Meier and `ggsurvplot` to plot the estimated curve.

```
dat <- data.frame(y)
fit.surv <- survfit(Surv(time, status) ~ 1,
                    data = dat)
ggsurvplot(fit.surv,conf.int=FALSE)
```



More specific information on the estimated survival probabilities can be obtained using the `summary` function.

```
# estimated probability of surviving beyond 10 years
summary(survfit(Surv(time, status) ~ 1, data = dat), times = 10)
```

Now, how do we study the relationship between covariates and survival time? The solution is Cox regression! We introduce the hazard function defined as

$$\begin{aligned} h(t) &= \lim_{dt \to 0} \frac{P(t \leq T < t + dt | T \geq t)}{dt} \\ &= -S'(t)/S(t). \end{aligned}$$

The Cox proportional hazards model then assumes a semi-parametric form for the hazard

$$h(t|X) = h_0(t) \exp(X^T \beta),$$

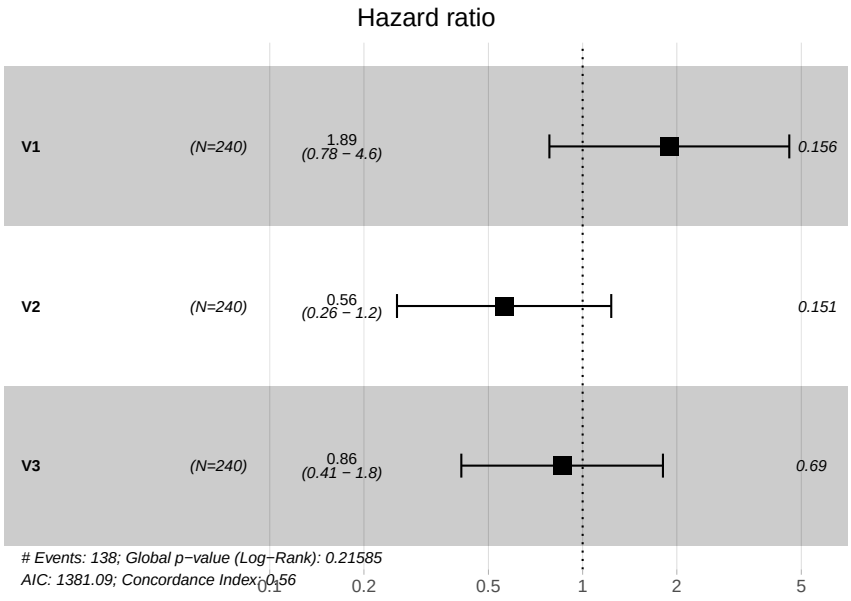
where $h_0(t)$ is the baseline hazard and β are the regression coefficients. Cox regression estimates the regression coefficients by maximizing the so-called partial likelihood function (surprisingly this works without specifying the baseline hazard function). For illustration we fit a Cox regression model using the first 3 genes as predictors.

```
dat <- data.frame(cbind(y,x[,1:3]))
fit <- coxph(Surv(time,status)~.,data=dat)
summary(fit)
```

```
## Call:
## coxph(formula = Surv(time, status) ~ ., data = dat)
##
##   n= 240, number of events= 138
##
##       coef exp(coef) se(coef)      z Pr(>|z|)
## V1  0.6382    1.8931   0.4504  1.417    0.156
## V2 -0.5778    0.5611   0.4023 -1.436    0.151
## V3 -0.1508    0.8600   0.3785 -0.398    0.690
##
##    exp(coef) exp(-coef) lower .95 upper .95
## V1    1.8931     0.5282    0.7831     4.577
## V2    0.5611     1.7822    0.2551     1.234
## V3    0.8600     1.1627    0.4095     1.806
##
## Concordance= 0.559  (se = 0.028 )
## Likelihood ratio test= 4.46  on 3 df,   p=0.2
## Wald test            = 4.66  on 3 df,   p=0.2
## Score (logrank) test = 4.66  on 3 df,   p=0.2
```

The (exponentiated) regression coefficients are interpreted as hazard-ratios. For example a unit change in the 3rd covariate accounts for a risk reduction of $\exp(\beta_3)$=0.86 or 14%. The results of Cox regression are often visualized using a forest plot.

```
ggforest(fit)
```



5.2 Regularized Cox Regression

The lymphoma data consists of $p = 7399$ predictors. A truly high-dimensional example! Similar as for linear - and logistic regression we can build upon the Cox regression model and use subset selection or regularization. The R package `glmnet` implements elastic net penalized cox regression. For illustration we restrict ourselves to the top genes (highest variance) and we scale the features as part of the data preprocessing.

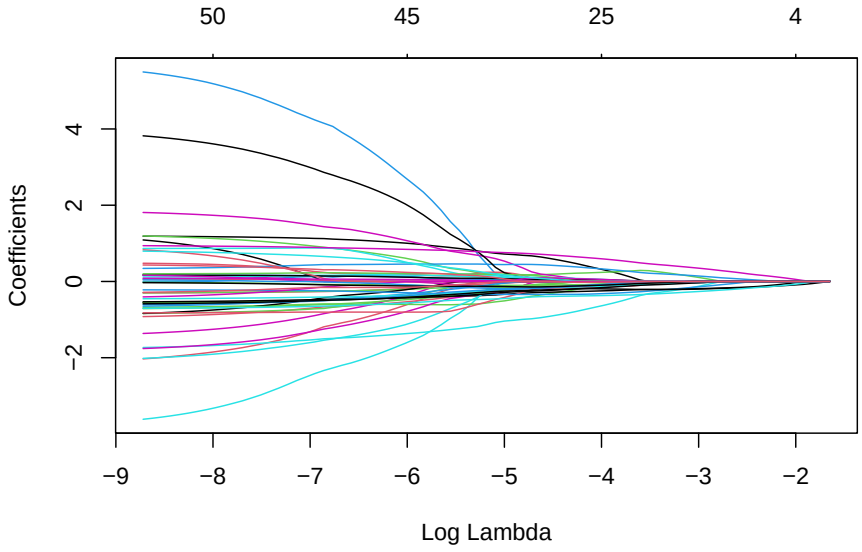
```
# filter for top genes (highest variance) and scale the input matrix
topvar.genes <- order(apply(x,2,var),decreasing=TRUE)[1:50]
x <- scale(x[,topvar.genes])
```

We split the data set into training and test data.

```
set.seed(1234)
train_ind <- sample(1:nrow(x),size=nrow(x)/2)
xtrain <- x[train_ind,]
ytrain <- y[train_ind,]
xtest <- x[-train_ind,]
ytest <- y[-train_ind,]
```

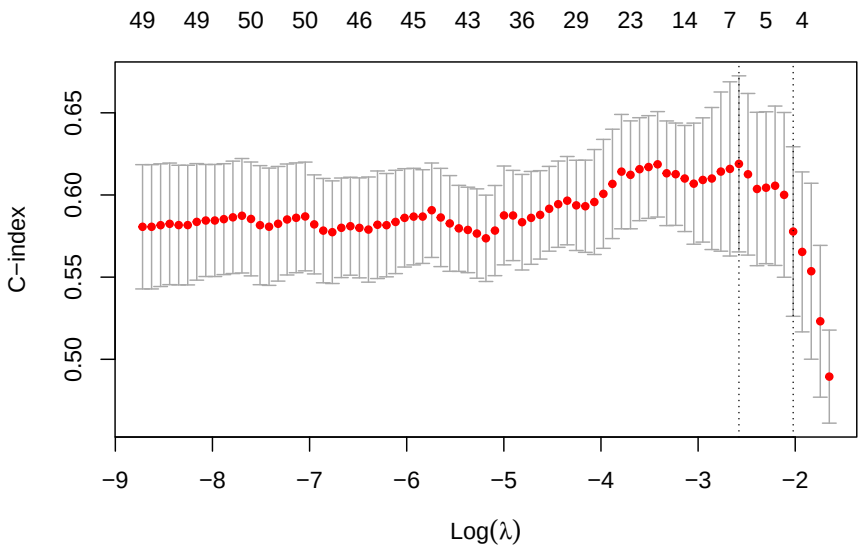
We invoke `glmnet` with argument `family="cox"` and set the mixing parameter to $\alpha = 0.95$.

```
set.seed(1)
ytrain.surv <- Surv(ytrain[,"time"],ytrain[,"status"])
fit.coxnet <- glmnet(xtrain, ytrain.surv, family = "cox",alpha=0.95)
plot(fit.coxnet,xvar="lambda")
```

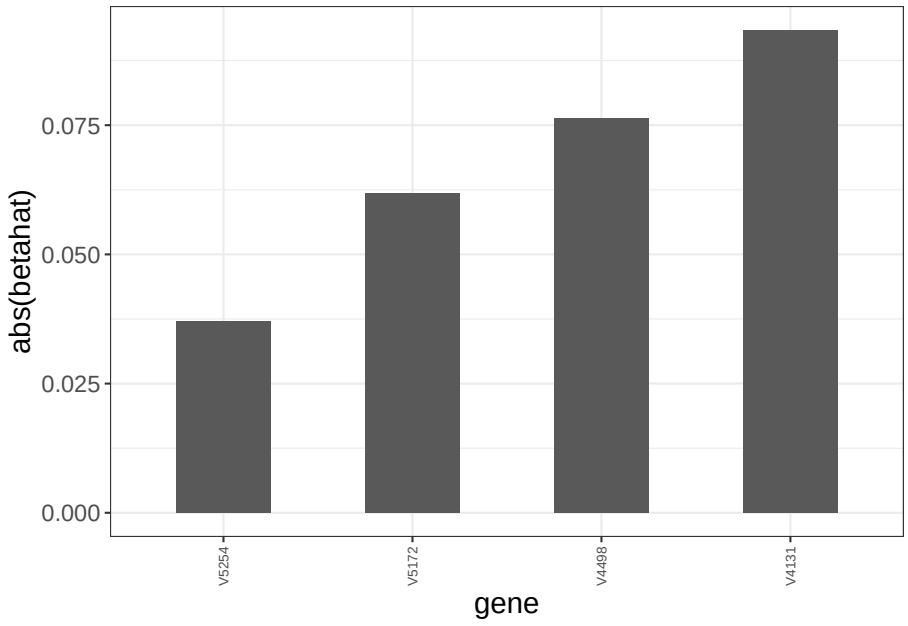


We tune the amount of penalization by using cross-validation and take Harrel's concordance index as a goodness of fit measure.

```
cv.coxnet <- cv.glmnet(xtrain,ytrain.surv,
                       family="cox",
                       type.measure="C",
                       nfolds = 5,
                       alpha=0.95)
plot(cv.coxnet)
```



The C-index ranges from 0.5 to 1. A value of 0.5 indicates that the model is no better at predicting an outcome than random chance. The largest tuning parameter within 1se of the maximum C-index is $\lambda_{\text{opt}} = 0.132$. The next graphic shows the magnitude of the non-zero coefficients (note that we standardized the input covariates).

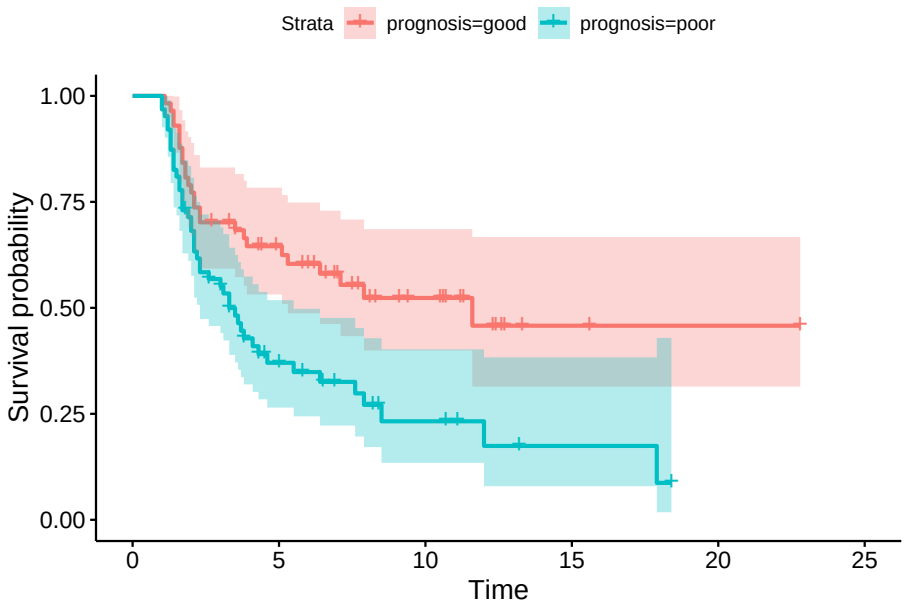


We use the obtained model to make predictions on the test data. In particular we compute the linear predictor

$$\hat{f}(X_{\text{new}}) = X_{\text{new}}^T \hat{\beta}_{\lambda_{\text{opt}}}.$$

We can now classify patients into good and poor prognosis based on thresholding the linear predictor at zero.

```
# linear predictor
lp <- predict(fit.coxnet,
              newx=xtest,
              s=cv.coxnet$lambda.1se,
              type="link")
dat.test <- data.frame(ytest)
dat.test$prognosis <- ifelse(lp>0,"poor","good")
fit.surv <- survfit(Surv(time, status) ~ prognosis,
                    data = dat.test)
ggsurvplot(fit.surv,conf.int = TRUE)
```



The survival curves are reasonably well separated, which suggests we have derived a gene signature which deserves further investigation.

5.3 Brier Score

We have seen how to evaluate the generalization error in the linear regression and classification context. For time-to-event data this is slightly more involved. A popular way to quantify the prediction accuracy is the time-dependent Brier score

$$\mathrm{BS}(t, \hat{S}) = \mathbf{E}[(\Delta_{\mathrm{new}}(t) - \hat{S}(t|X_{\mathrm{new}}))^2]$$

where $\Delta_{\mathrm{new}}(t) = \mathbf{1}(T_{\mathrm{new}} \geq t)$ is the true status of a new test subject and $\hat{S}(t|X_{\mathrm{new}})$ is the predicted survival probability. Calculation of the Brier score is complicated by the fact that we do not always observe the event time T due to censoring. The R package `pec` estimates the Brier score using a technique called *Inverse Probability of Censoring Weighting (IPCW)*.

We use forward selection on the training data to obtain a prediction model.

```
dtrain <- data.frame(cbind(ytrain,xtrain))
dtest <- data.frame(cbind(ytest,xtest))
fit.lo <- coxph(Surv(time,status)~1,data=dtrain,
```

```
              x=TRUE,y=TRUE)
up <- as.formula(paste("~",
                       paste(colnames(xtrain),
                             collapse="+")))
fit.fw <- stepAIC(fit.lo,
                  scope=list(lower=fit.lo,
                             upper=up),
                  direction="both",
                  trace=FALSE)
```

The following table summarizes the variables added in each step of the forward selection approach.

```
kable(as.data.frame(fit.fw$anova),digits=3,booktabs=TRUE)
```

Step	Df	Deviance	Resid. Df	Resid. Dev	AIC
	NA	NA	67	587.326	587.326
+ V4131	1	7.554	66	579.771	581.771
+ V4498	1	7.405	65	572.366	576.366
+ V5172	1	5.272	64	567.094	573.094
+ V5254	1	7.388	63	559.706	567.706
+ V5223	1	3.802	62	555.903	565.903
+ V4356	1	3.766	61	552.138	564.138
+ V4341	1	3.311	60	548.826	562.826

We further run a Cox regression model based on the predictors selected by `glmnet`.

```
beta.1se <- coef(fit.coxnet,s=cv.coxnet$lambda.1se)
vars.1se <- rownames(beta.1se)[as.numeric(beta.1se)!=0]
fm.1se <- as.formula(paste0("Surv(time,status)~",
                            paste0(vars.1se,collapse="+")))
fit.1se <- coxph(fm.1se,data=dtrain,x=TRUE,y=TRUE)
```

Finally we use the `pec` package to calculate Brier scores for both models on the training and test data.

```
library(pec)
fit.pec.train <- pec::pec(
  object=list("cox.fw"=fit.fw,
              "cox.1se"=fit.1se),
  data = dtrain,
  formula = Surv(time, status) ~ 1,
  splitMethod = "none")
```

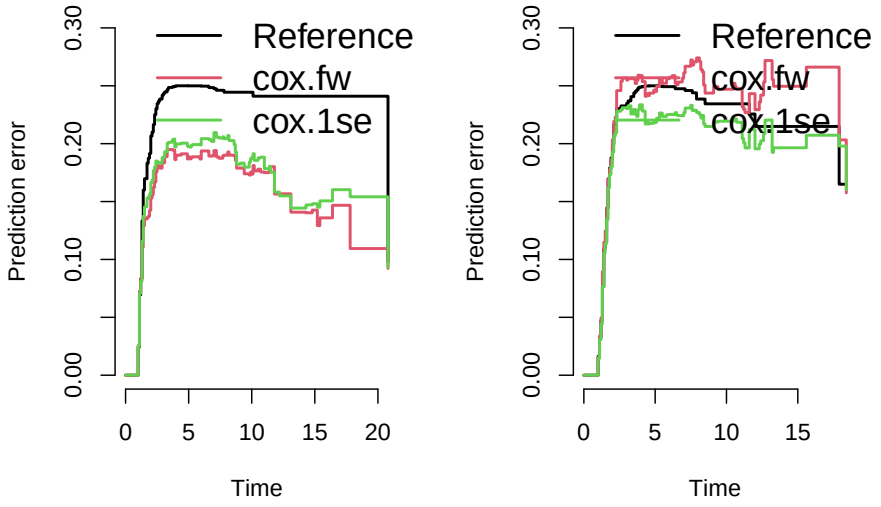
```
## Warning in .recacheSubclasses(def@className, def, env): undefined subclass
## "packedMatrix" of class "replValueSp"; definition not updated
```

```
## Warning in .recacheSubclasses(def@className, def, env): undefined subclass
## "packedMatrix" of class "mMatrix"; definition not updated
```

```
fit.pec.test <- pec::pec(
  object=list("cox.fw"=fit.fw,
              "cox.1se"=fit.1se),
  data = dtest,
  formula = Surv(time, status) ~ 1,
  splitMethod = "none")
```

The following figure shows the Brier scores evaluated on training and test data.

```
par(mfrow=c(1,2))
plot(fit.pec.train,main="training data")
plot(fit.pec.test,main="test data")
```



The plot on the right shows the Brier score on the test data and indicates that the glmnet selected model performs slightly better than the reference model (no covariates, Kaplan-Meier estimate only).

The pec package can also be used to benchmark different prediction models. We illustrate this based on random forest and forward selection. In this illustration we do not split the data into training and test. Instead we use cross-validation to compare the two prediction approaches.

We start by writing a small wrapper function to use forward selection in pec. (A detailed description on the pec package and on how to set up wrapper functions is provided here.)

```
selectCoxfw <- function(formula,data,steps=100,direction="both")
{
  require(prodlim)
  fmlo <- reformulate("1",formula[[2]])
  fitlo <- coxph(fmlo,data=data,x=TRUE,y=TRUE)
  fwfit <- stepAIC(fitlo,
                   scope=list(lower=fitlo,
                              upper=formula),
                   direction=direction,
                   steps=steps,
                   trace=FALSE)
  if (fwfit$formula[[3]]==1){
    newform <- reformulate("1",formula[[2]])
    newfit <- prodlim(newform,
                      data=data)
  }else{
    newform <-fwfit$formula
    newfit <- coxph(newform,data=data,x=TRUE,y=TRUE)
  }
  out <- list(fit=newfit,
              In=attr(terms(newfit$formula),which = "term.labels"))
  out$call <-match.call()
  class(out) <- "selectCoxfw"

  out
}

predictSurvProb.selectCoxfw <- function(object,newdata,times,...){
  predictSurvProb(object[[1]],newdata=newdata,times=times,...)
}
```

We run forward selection.

```
dat <- data.frame(cbind(y,x))
fm <- as.formula(paste("Surv(time, status) ~ ",
                       paste(colnames(dat[,-(1:2)]),
                             collapse="+")))
```

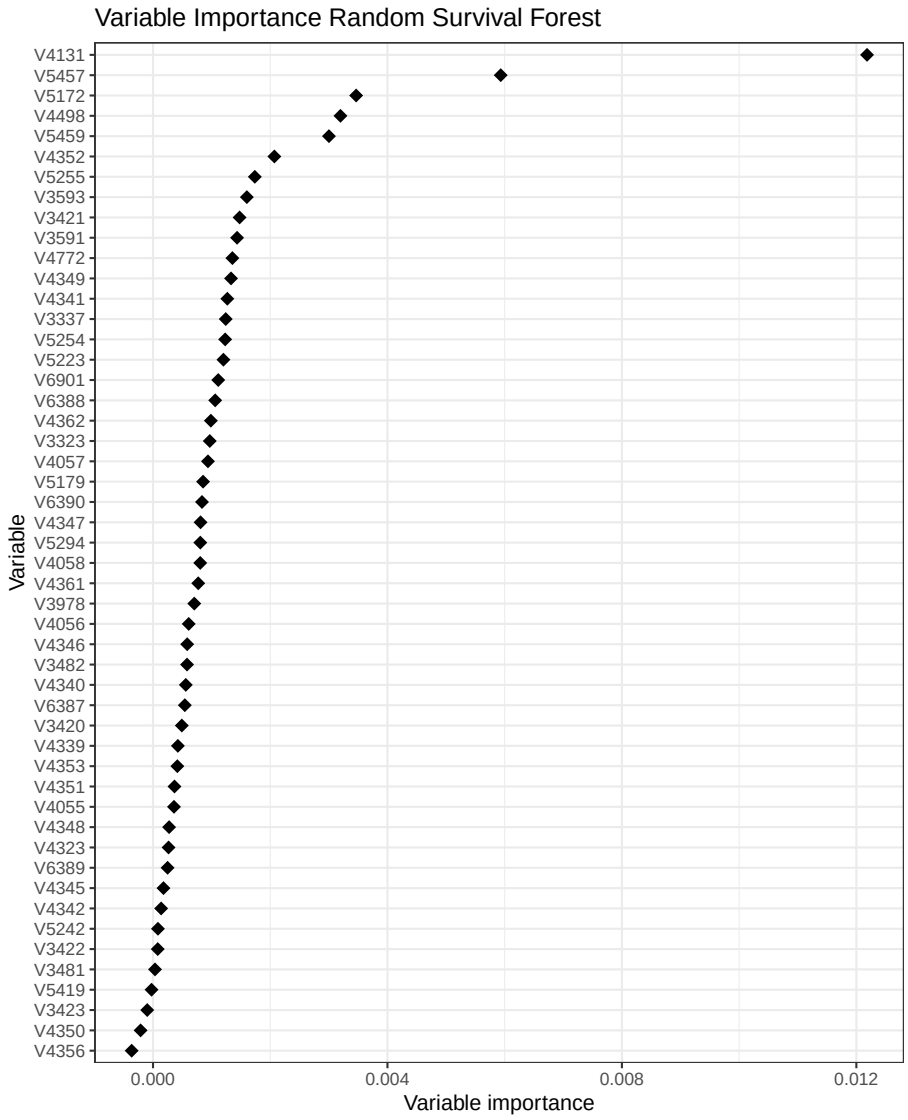
```
fit.coxfw <- selectCoxfw(fm,data=dat,
                         direction="forward")
```

We fit a random forest using `cforest` from the `party` package.

```
fit.cforest <- pec::pecCforest(fm, data =dat,
                               control = party::cforest_classical(ntree = 100))
```

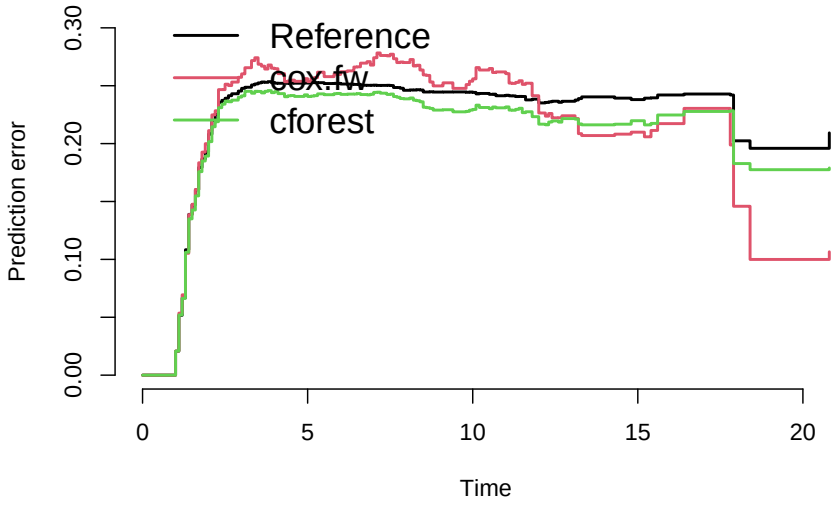
We random forest we can obtain a measure of variable importance using the function `varimp`.

```
##
## Variable importance for survival forests; this feature is _experimental_
```



Finally we compare the two approaches using the cross-validated Brier score.

```
pec.cv <- pec::pec(
  object=list("cox.fw"=fit.coxfw,"cforest"=fit.cforest),
  data = dat,
  formula = Surv(time, status) ~ 1,
  splitMethod = "cv5")
plot(pec.cv)
```



We conclude that forward selection and random forest do not outperform the reference model.

Chapter 6

High-Dimensional Feature Assessment

A frequent task is to find features which differ with respect to one or more experimental factors. We illustrate this type of analysis using a gene expression experiment ($p = 15923$ genes) performed with 12 randomly selected mice from two strains. The features are the $p = 15923$ genes and the strain (A vs B) is the experimental factor.

A commonly used format for gene expression data is the `ExpressionSet` class from the `Biobase` package. The actual expressions are retrieved using the function `exprs`. Information on the phenotypes is obtained using `pData` and with `fData` we get more information on the genes ("features").

We load the `ExpressionSet`.

```
esetmouse <- readRDS(file="data/esetmouse.rds")
class(esetmouse)
```

```
## [1] "ExpressionSet"
## attr(,"package")
## [1] "Biobase"
```

```
dim(esetmouse)
```

```
## Features  Samples
##    15923       24
```

We can look at the expression values of the first sample and the first 6 genes.

```
exprs(esetmouse)[1:6,1]
```

```
## 1367452_at 1367453_at 1367454_at 1367455_at 1367456_at 1367457_at
##  10.051651  10.163334  10.211724  10.334899  10.889349   9.666755
```

An overview on the phenotype data can be obtained using the following commands.

```
table(pData(esetmouse)$strain)
```

```
##
##  A  B
## 12 12
```

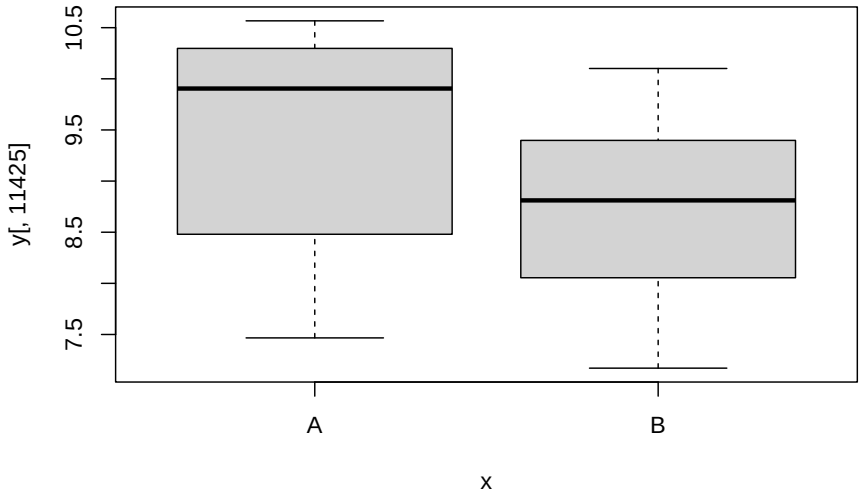
6.1 Gene-wise Two-sample Comparison

We are interested in comparing gene expression between the mice strains A and B.

```
x <- esetmouse$strain # strain information
y <- t(exprs(esetmouse)) # gene expressions matrix (columns refer to genes)
```

We start by visualizing the expression of gene $j = 11425$.

```
boxplot(y[,11425]~x)
```



This gene seems to be higher expressed in A. To nail down this observeration we can do a more formal hypothesis test. We build the ordinary t-statistic

$$t_j = \frac{\overline{y}_j^B - \overline{y}_j^A}{s_j \sqrt{\frac{1}{n_A} + \frac{1}{n_B}}}.$$

We can calculate the two-sided p-value

$$q_j = 2\left(1 - F(|t_j|, \nu = n_A + n_B - 2)\right).$$

In R we can perform a two-sample t-test using the function `t.test`.

```
ttest <- t.test(y[,11425]~x,var.equal=TRUE)
ttest$statistic #tscore
```

```
##        t
## 1.774198
```

```
ttest$p.value
```

```
## [1] 0.0898726
```

We obtain q_{11425}=0.09 and based on that we would not reject the null-hypothesis for this specific gene at the $\alpha = 0.05$ level. What about the other genes? We continue by repeating the analysis for all $p = 15923$ genes. We save the results in a data frame.

```
pvals <- apply(y,2,FUN=
                 function(y){
                   t.test(y~x,var.equal=TRUE)$p.value
                 })
tscore <- apply(y,2,FUN=
                  function(y){
                    t.test(y~x,var.equal=TRUE)$statistic
                  })
res.de <- data.frame(p.value=pvals,
                     t.score=tscore,
                     geneid=names(tscore))
```

Next we count the number of significant genes.

```
sum(res.de$p.value<0.05)
```

```
## [1] 2908
```

According to this analysis 2908 genes are differentially expressed between strains A and B. This is 18.3% of all genes. In the next section we will explain that this analysis misses an important point, namely it neglects the issue of multiple testing.

6.2 Multiple Testing

To illustrate the multiple testing problem we create an artificial gene expression data set where we are certain that none of the genes is differentially expressed.

```
set.seed(1)
p <- ncol(y)
n <- nrow(y)
ysim <- matrix(rnorm(n*p),n,p)
```

Now we repeat the gene-wise two-sample comparisons for the artificial data set.

```
pvals.sim <- apply(ysim,2,FUN=
                 function(y){
                   t.test(y~x,var.equal=TRUE)$p.value
                 })
tscore.sim <- apply(ysim,2,FUN=
                  function(y){
                    t.test(y~x,var.equal=TRUE)$statistic
                  })
res.de.sim <- data.frame(p.value=pvals.sim,t.score=tscore.sim)
```

We count the number of significant genes.

```
sum(res.de.sim$p.value<0.05)
```

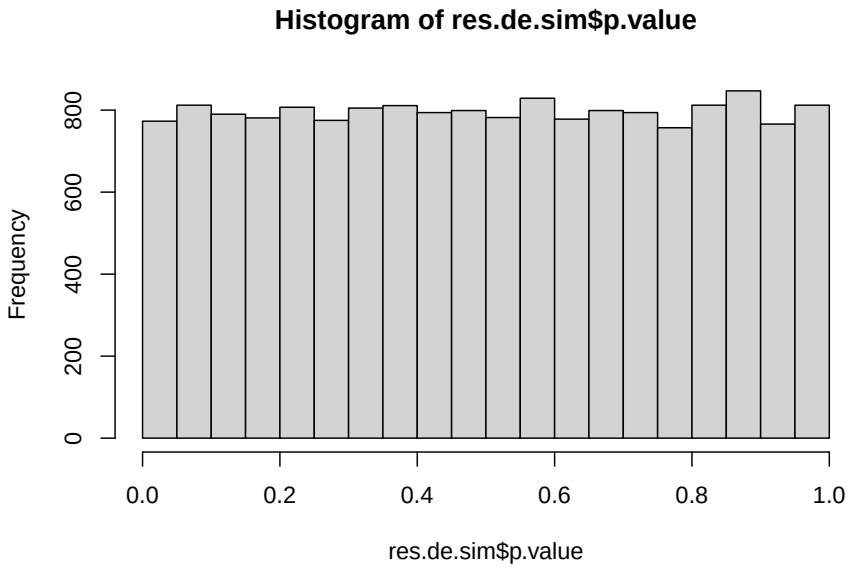
```
## [1] 773
```

This is a surprise! According to the analysis 773 genes are differentially expressed. However, we know that this cannot be true. What did we miss? The reason for the large number of falsely declared significant genes is that we performed multiple significance tests simultaneously. Each test is associated with an error which accumulate over the various test. In particular, we re-call that the probability of falsely rejecting the null-hypothesis (=Type-I error) is

$$\text{Prob}(q_j < \alpha) \leq \alpha.$$

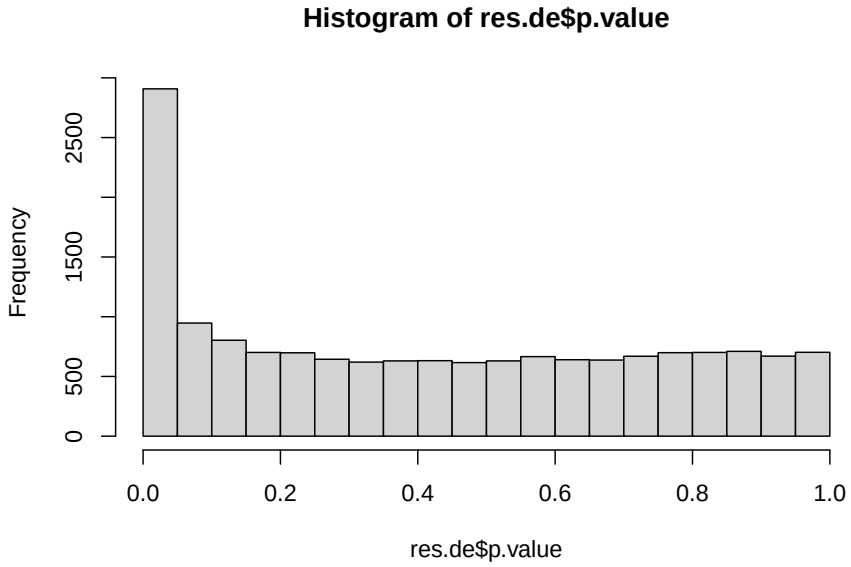
We performed a significance test for each gene which makes the expected number of falsely rejected null-hypotheses $p \times \alpha$=796.15. Under the null hypothesis we would expect the p-values to follow a uniform distribution. Indeed, that is what we observe in our simulation example.

```
hist(res.de.sim$p.value)
```



The distribution of p-values obtained from the real example has a peak near zero which indicates that some genes are truly differentially expressed between strains A and B.

```
hist(res.de$p.value)
```



In the next section we will discuss *p-value adjustment* which is a method to counteract the issue of multiple testing.

6.3 P-value Adjustment

Our previous consideration suggest that we could adjust the p-values by multiplying with the number p of performed tests, i.e.

$$q_j^{\text{adjust}} = p \times q_j.$$

This adjustment method is known as the Bonferroni correction. The method has the property that it controls the so-called family-wise-error rate (FWER). Let's assume that p_0 is the number of *true* null hypotheses (unknown to the researcher), then we can show

$$\begin{aligned}
\text{FWER} &= \text{Prob}(\text{at least one false positive}) \\
&= \text{Prob}(\min_{j=1..p_0} q_j^{\text{adjust}} \leq \alpha) \\
&= \text{Prob}(\min_{j=1..p_0} q_j \leq \alpha/p) \\
&\leq \sum_{j=1}^{p_0} \text{Prob}(q_j \leq \alpha/p) \\
&= p_0 \frac{\alpha}{p} \leq \alpha.
\end{aligned}$$

In our example we calculate the Bonferroni adjusted p-values.

```
res.de$p.value.bf <- p*res.de$p.value
res.de.sim$p.value.bf <- p*res.de.sim$p.value
```

The number of significant genes in the real and simulated data are provided next. Note that none of the genes is significant in the simulated data which is in line with our expectations.

```
sum(res.de$p.value.bf<0.05)
```

```
## [1] 82
```

```
sum(res.de.sim$p.value.bf<0.05)
```

```
## [1] 0
```

The R function `p.adjust` offers various adjustment procedures. The different methods are based on different assumptions and/or they control a different error measure. The Bonferroni correction is the most conservative approach and often leads to too few significant result (loss of statistical power). Less conservative is the so-called FDR approach which controls the False Discovery Rate (instead of FWER). We calculate the FDR adjusted p-values and print the number of significant genes.

```
res.de$p.value.fdr <- p.adjust(res.de$p.value,method="fdr")
res.de.sim$p.value.fdr <- p.adjust(res.de.sim$p.value,method="fdr")
sum(res.de$p.value.fdr<0.05)
```

```
## [1] 1123
```

```
sum(res.de.sim$p.value.fdr<0.05)
```

```
## [1] 0
```

6.4 Volcano Plot

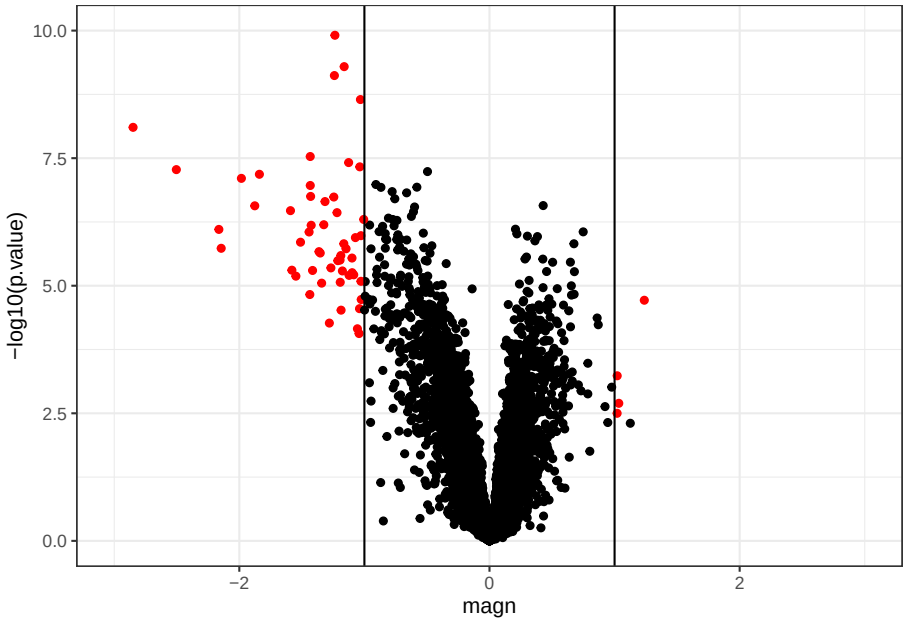
It is important to effectively display statistical results obtained from high-dimensional data. We have discussed how to calculate p-values and how to adjust them for multiplicity. However, the p-value is often not the only quantity of interest. In differential gene expression analysis we are also interested in the magnitude of change in expression.

```
magn<- apply(y,2,FUN=
             function(y){
               mba <- tapply(y,x,mean)
               return(mba[2]-mba[1])
             })
magn.sim <- apply(ysim,2,FUN=
                  function(y){
                    mba <- tapply(y,x,mean)
                    return(mba[2]-mba[1])
                  })
res.de$magn <- magn
res.de.sim$magn <- magn.sim
```

A frequently used display is the volcano plot which shows on the y-axis the $-\log_{10}$ p-values and on the x-axis the magnitude of change. By using $-\log_{10}$, the "highly significant" features appear at the top of the plot. Using log also permits us to better distinguish between small and very small p-values. We can further highlight the "top genes" as those with adjusted p-value <0.05 and magnitude of change > 1.

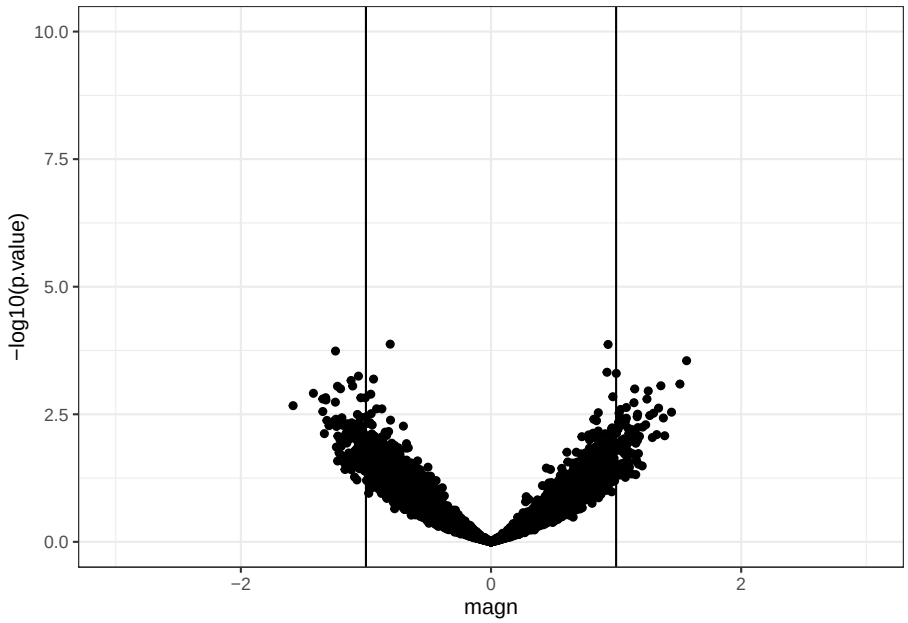
```
res.de%>%
  dplyr::mutate(topgene=ifelse(p.value.fdr<0.05&abs(magn)>1,
                               "top",
                               "other")
                )%>%
  ggplot(.,aes(x=magn,y=-log10(p.value),col=topgene))+
  geom_point()+
  scale_color_manual(values = c("top"="red","other"="black"))+
  theme_bw()+
  theme(legend.position = "none")+
```

```
  xlim(-3,3)+ylim(0,10)+
  geom_vline(xintercept = 1)+
  geom_vline(xintercept = -1)
```



We repeat the same plot with the simulated data.

```
res.de.sim%>%
  dplyr::mutate(topgene=ifelse(p.value.fdr<0.05&abs(magn)>1,
                               "top",
                               "other")
                )%>%
  ggplot(.,aes(x=magn,y=-log10(p.value),col=topgene))+
  geom_point()+
  scale_color_manual(values = c("top"="red","other"="black"))+
  theme_bw()+
  theme(legend.position = "none")+
  xlim(-3,3)+ylim(0,10)+
  geom_vline(xintercept = 1)+
  geom_vline(xintercept = -1)
```



6.5 Variance Shrinkage and Empirical Bayes

The basis of the statistical analyses are the t-statistics

$$t_j = \frac{\overline{y}_j^B - \overline{y}_j^A}{s_j\sqrt{\frac{1}{n_A} + \frac{1}{n_B}}}.$$

In a small sample size setting the estimated standard deviations exhibit high variability which can lead to large t-statistics. Extensive statistical methodology has been developed to counteract this challenge. The key idea of those methods is to *shrink* the gene-wise variances s_j^2 towards a common variance s_0^2 (s_0^2 is estimated from the data)

$$\tilde{s}_j^2 = \frac{d_0 s_0^2 + d s_j^2}{d_0 + d}.$$

A so-called *moderated* t-statistic is obtained by replacing in the denominator s_j with the "shrunken" $\tilde{s}_j$. The *moderated* t-statistic has favourable statistical properties in the small n setting. The statistical methodology behind the approach is referred to as empirical Bayes and is implemented in the function

`eBayes` of the `limma` package. Limma starts with running gene-wise linear regression using the `lmFit` function.

```
library(limma)
```

```
##
## Attaching package: 'limma'
```

```
## The following object is masked from 'package:BiocGenerics':
##
##     plotMA
```

```
# first argument: gene expression matrix with genes in rows and sample in columns
# second argument: design matrix
fit <- lmFit(t(y), design=model.matrix(~ x))
head(coef(fit))
```

```
##             (Intercept)           xB
## 1367452_at   10.027453  0.092544985
## 1367453_at   10.173732  0.026867630
## 1367454_at   10.275137  0.003017421
## 1367455_at   10.371786 -0.101727288
## 1367456_at   10.815641 -0.006899555
## 1367457_at    9.607297  0.038318498
```

We can compare it with a standard `lm` fit.

```
coef(lm(y[,1]~x)) # gene 1
```

```
## (Intercept)          xB
## 10.02745295  0.09254499
```

```
coef(lm(y[,2]~x)) # gene 2
```

```
## (Intercept)          xB
## 10.17373179  0.02686763
```

Next, we use the `eBayes` function to calculate the moderated t statistics and p-values.

```
ebfit <- eBayes(fit)
head(ebfit$t) # moderated t statistics
```

```
##             (Intercept)          xB
## 1367452_at    182.4496  1.19066640
## 1367453_at    221.0103  0.41271149
## 1367454_at    184.0507  0.03821825
## 1367455_at    195.0270 -1.35258223
## 1367456_at    257.5965 -0.11619667
## 1367457_at    196.7628  0.55492613
```

```
head(ebfit$p.value) # p.values based on moderated t statistics
```

```
##              (Intercept)        xB
## 1367452_at 3.871319e-43 0.2441871
## 1367453_at 2.236672e-45 0.6830894
## 1367454_at 3.061059e-43 0.9697959
## 1367455_at 6.452172e-44 0.1874515
## 1367456_at 3.639411e-47 0.9083599
## 1367457_at 5.084786e-44 0.5835310
```

We can also retrieve the "shrunken" standard deviations

```
head(sqrt(ebfit$s2.post)) # shrunken standard deviations
```

```
## 1367452_at 1367453_at 1367454_at 1367455_at 1367456_at 1367457_at
##  0.1903875  0.1594624  0.1933930  0.1842254  0.1454464  0.1691410
```

Bibliography

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA.

R Core Team (2022). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.