Exercises and Solutions - Analysis of High-Dimensional Data

Nicolas Städler

2022-03-27

Exercises

Prostate cancer data and linear regression

We explore the prostate cancer data set described in the book @elements. The data is available at github. The aim is to investigate the relationship between the level of prostate-specific antigen (`lpsa`) and several clinical covariates. The covariates are log cancer volume (`lcavol`), log prostate weight (`lweight`), `age`, log of the amount of benign prostatic hyperplasia (`lbph`), seminal vesical invasion (`svi`), log of capsular penetration (`lcp`), Gleason score (`gleason`), and the percent of Gleason scores 4 or 5 (`pgg45`). Further there is a variable `train` indicating training and test sets.

1. Read the prostate cancer data set and make a histogram for the response variable `lpsa`.
2. Create a scatterplot matrix for all variables in the data set. Use `pairs` or `ggpairs`.
3. Standardize the predictors to have unit variance and divide the data into training and test sets (see variable `train`). How many test and training samples do you count?
4. Run a univariate regression model with `lcavol` as covariate. Study the `summary` output.
    - How do you interpret the regression coefficients for `lcavol`?
    - What is the meaning of the *multiple R-squared*?
    - What is the *residual standard error*?
    - Generate a scatter plot of `lcavol` against `lpsa` and add the regression line with confidence band (use `geom_smooth`, `method="lm"`).
    - Draw the Tukey Anscombe plot and the QQ plot (check `?plot.lm`). What are these two plots telling us?
5. Run a multiple regression model using all covariates. Study the `summary` output.
    - What does change in the interpretation of the coefficient for `lcavol`?
    - What do you conclude from the *multiple R-squared*?
    - Create a Tukey Anscombe plot and a QQ plot.
6. Run a multiple regression model including all covariates but `lacvol`. Proceed as in 5.
7. Calculate the RSS for all 3 models. Write down your observation.
8. Compare all 3 models using the `anova` function. What do you conclude?

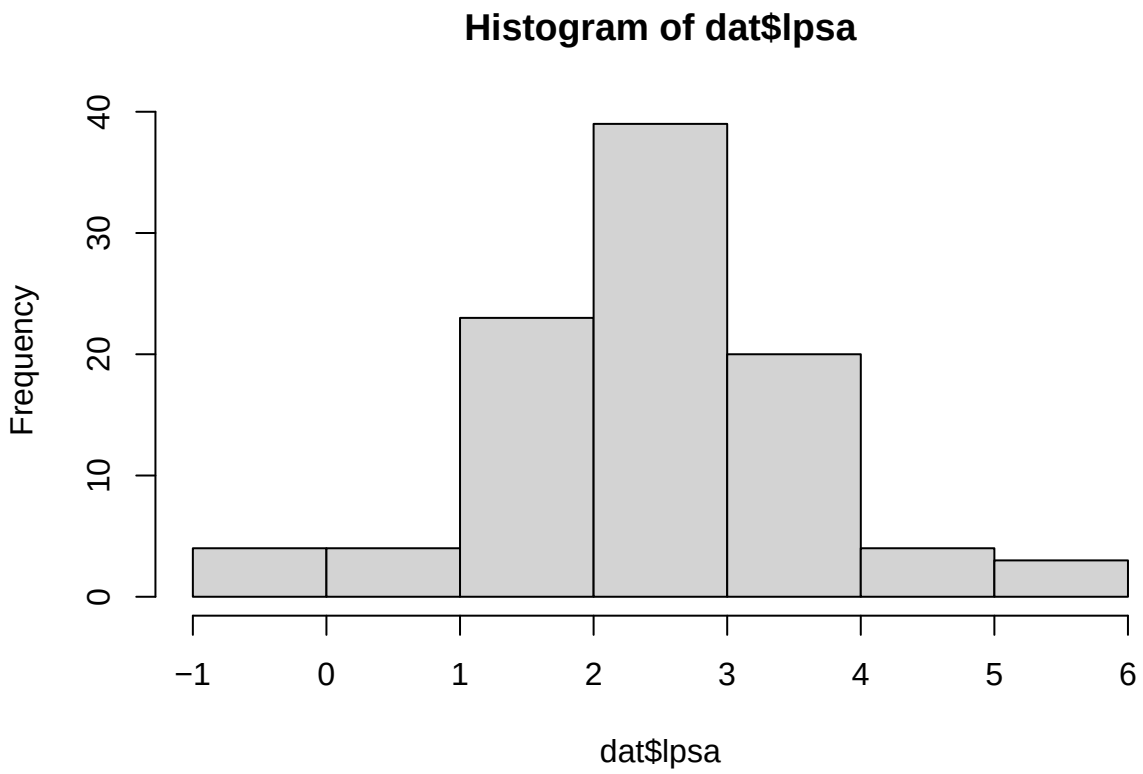
Solution to the exercise.

Read the data set.

```
dat <- read.csv("data/prostate.csv")
```

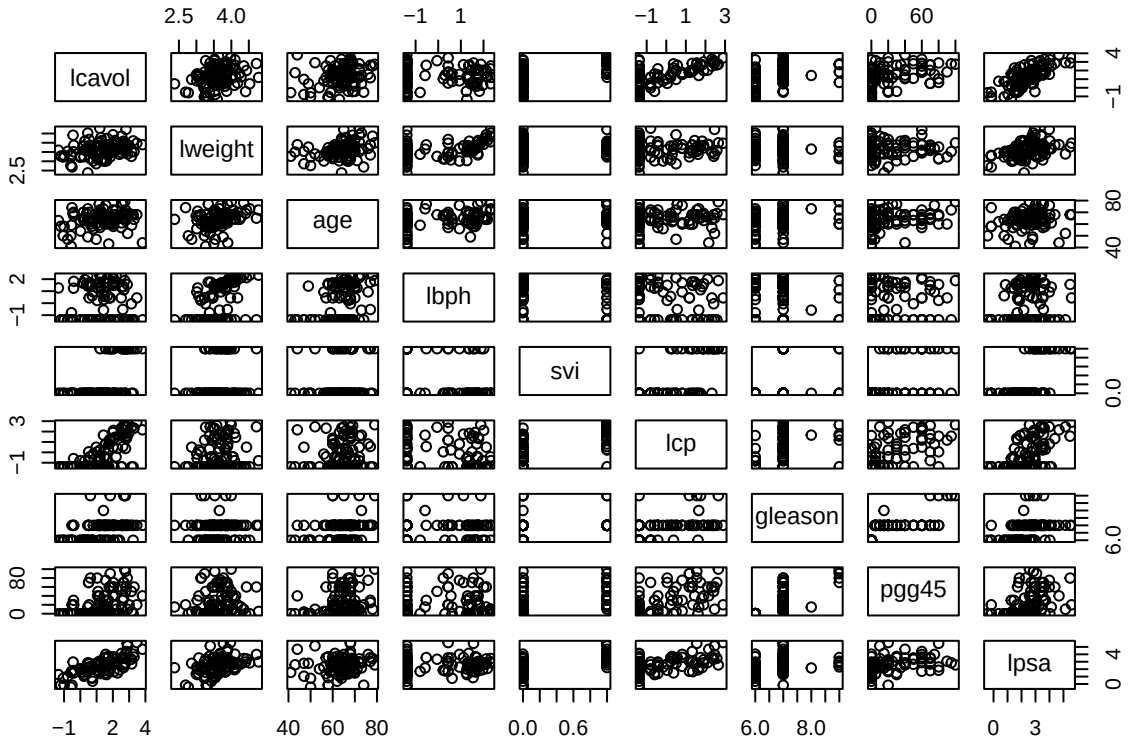
Generate a histogram of `lpsa`.

```
hist(dat$lpsa)
```



Scatterplot matrix of the prostate cancer data.

```
pairs(dat[,-10])
```



Standardize the predictors and create training and test data.

```
dat <- cbind(scale(dat[,-c(9:10)]),dat[,c(9,10)])
dtrain <- dat[dat$train,-10]
dtest <- dat[!dat$train,-10]
nrow(dtrain)
```

```
## [1] 67
```

```
nrow(dtest)
```

```
## [1] 30
```

Fit a univariate regression model.

```
fit1 <- lm(lpsa~lcavol,data=dtrain)
summary(fit1)
```

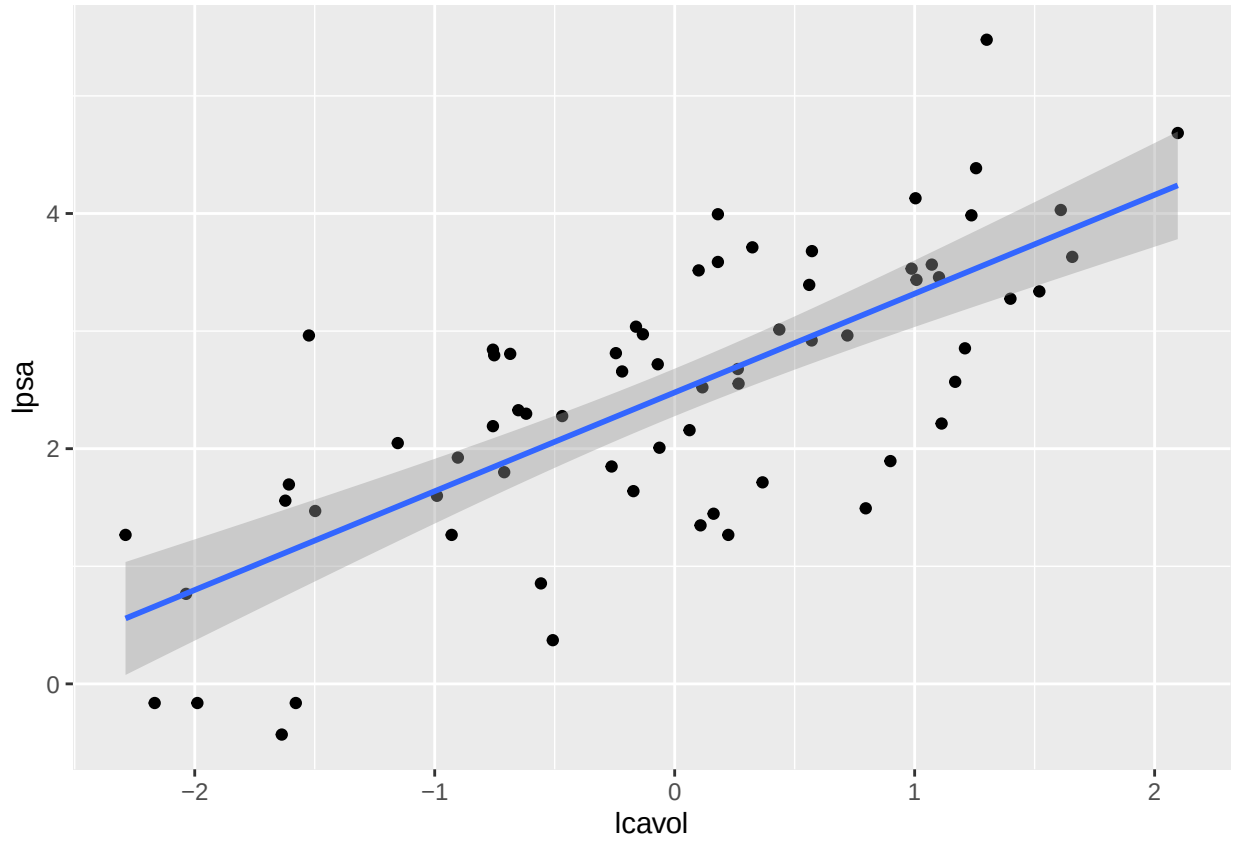
```
##
## Call:
## lm(formula = lpsa ~ lcavol, data = dtrain)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6802 -0.4240  0.1684  0.5392  1.9070
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.47837    0.10116  24.499  < 2e-16
```

```
## lcavol       0.83993    0.09664   8.692 1.73e-12
##
## (Intercept) ***
## lcavol      ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8277 on 65 degrees of freedom
## Multiple R-squared:  0.5375, Adjusted R-squared:  0.5304
## F-statistic: 75.55 on 1 and 65 DF,  p-value: 1.733e-12
```

Scatter plot with regression line.

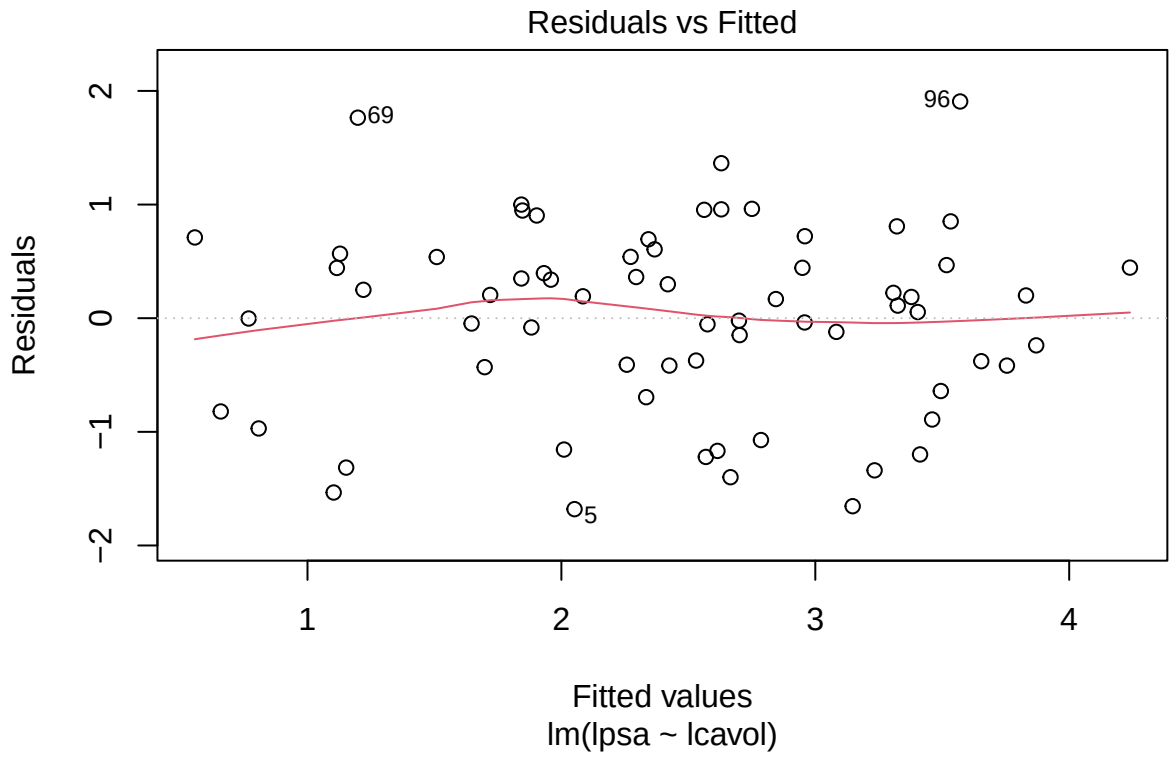
```
dtrain%>%
  ggplot(data=.,aes(x=lcavol,y=lpsa))+
  geom_point()+
  geom_smooth(method="lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



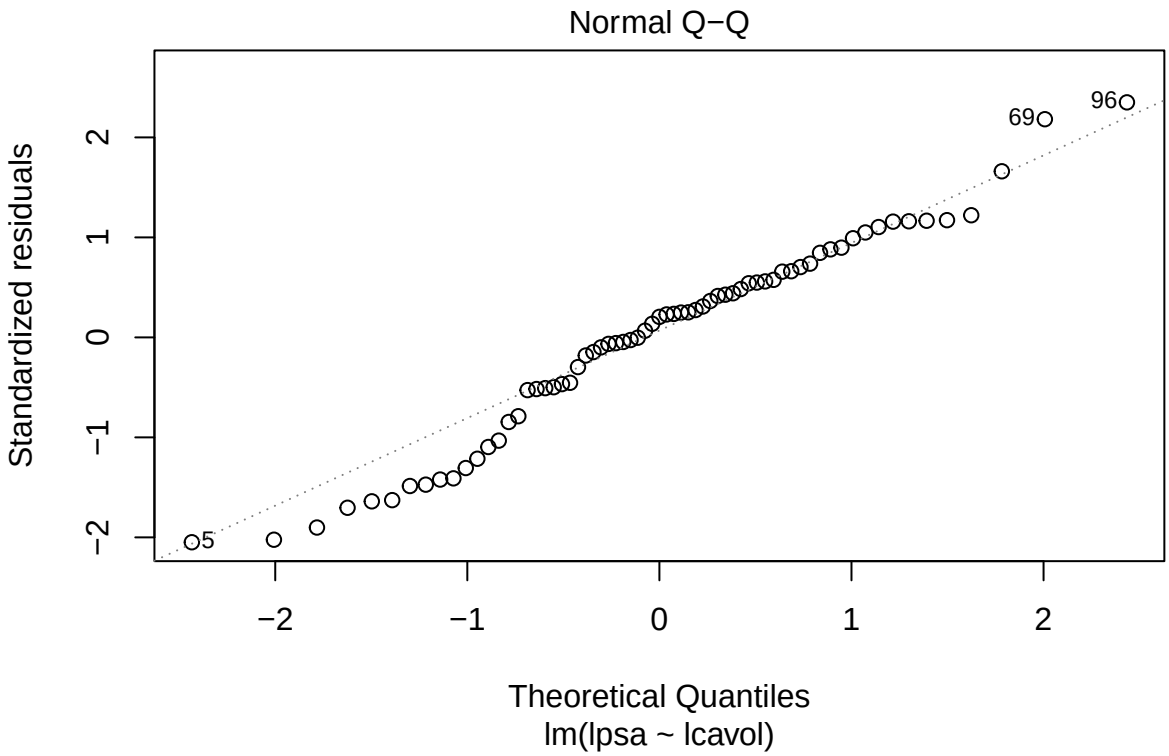
The Tukey Anscombe plot. The residuals scatter around the 0 line and do not show any systematic pattern. This indicates that the residuals are independent and have mean 0.

```
plot(fit1,which=1) # Tukey Anscombe plot
```

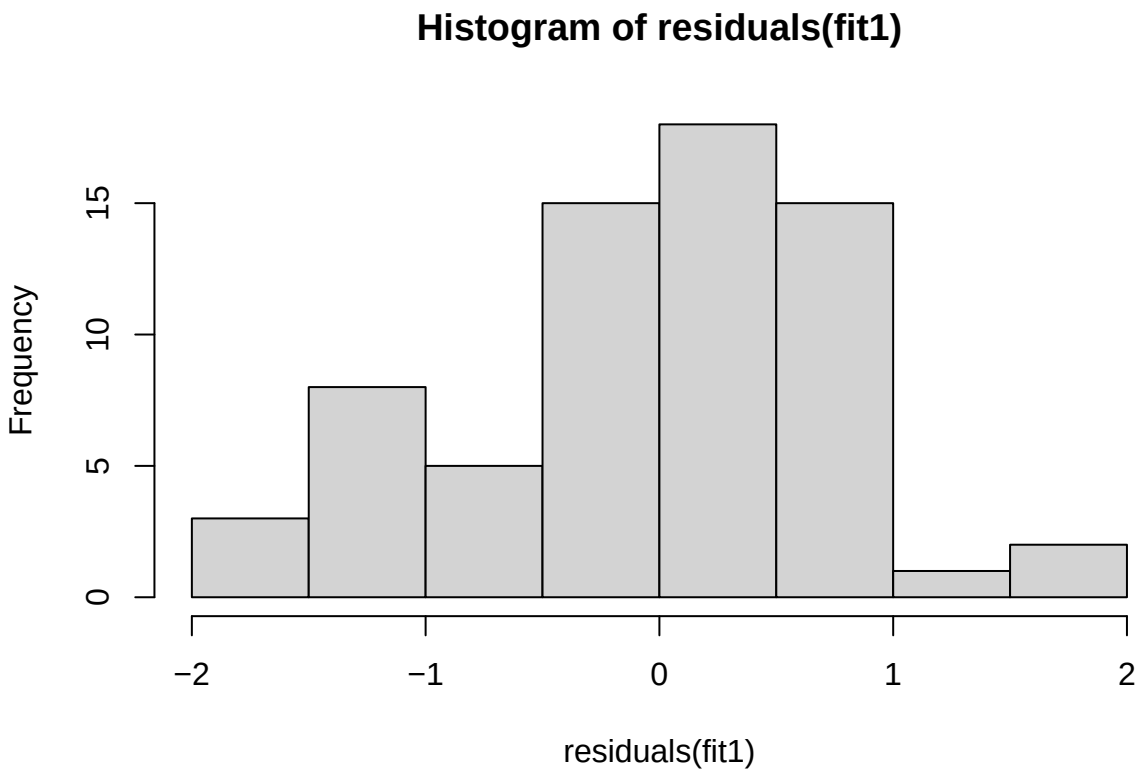


The QQ plot. This plot is used to check the normality assumption of the residuals. The residuals show slight tendency to be left-tailed (see also the histogram).

```
plot(fit1,which=2) # Tukey Anscombe plot
```




```
hist(residuals(fit1))
```



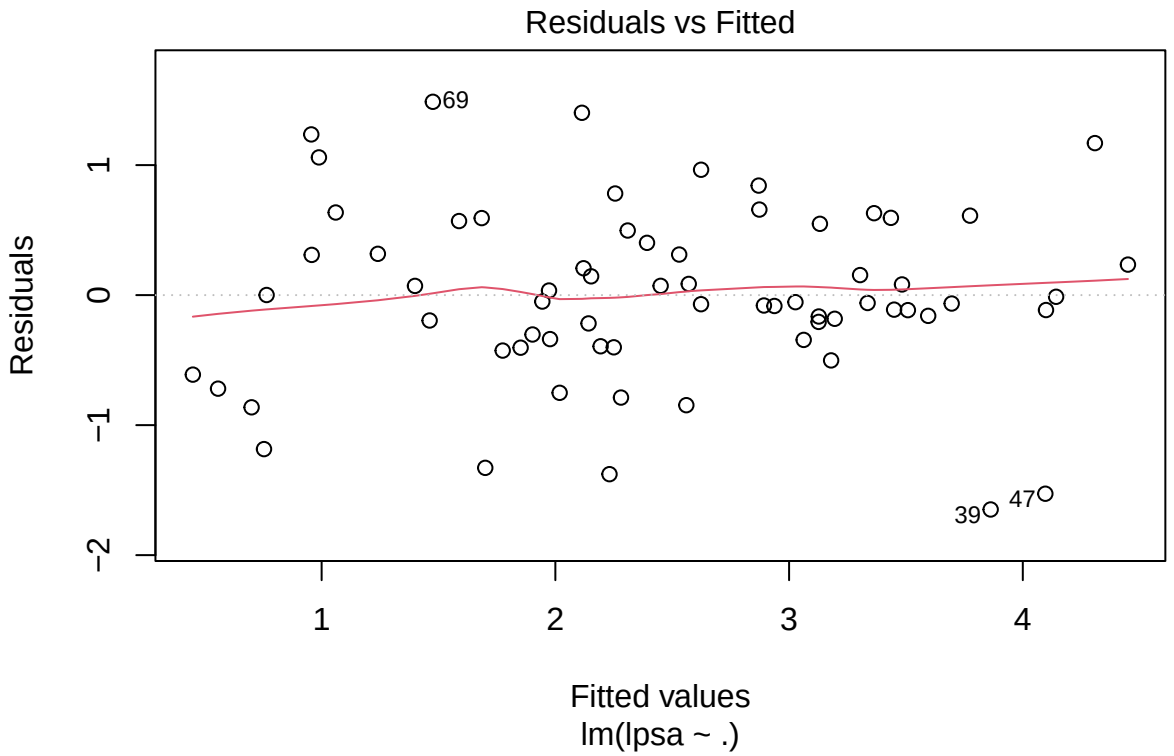
We run the multiple regression model with all covariates. We print the `summary` and create TA and QQ plots.

```
fit2 <- lm(lpsa~.,data=dtrain)
summary(fit2)
```

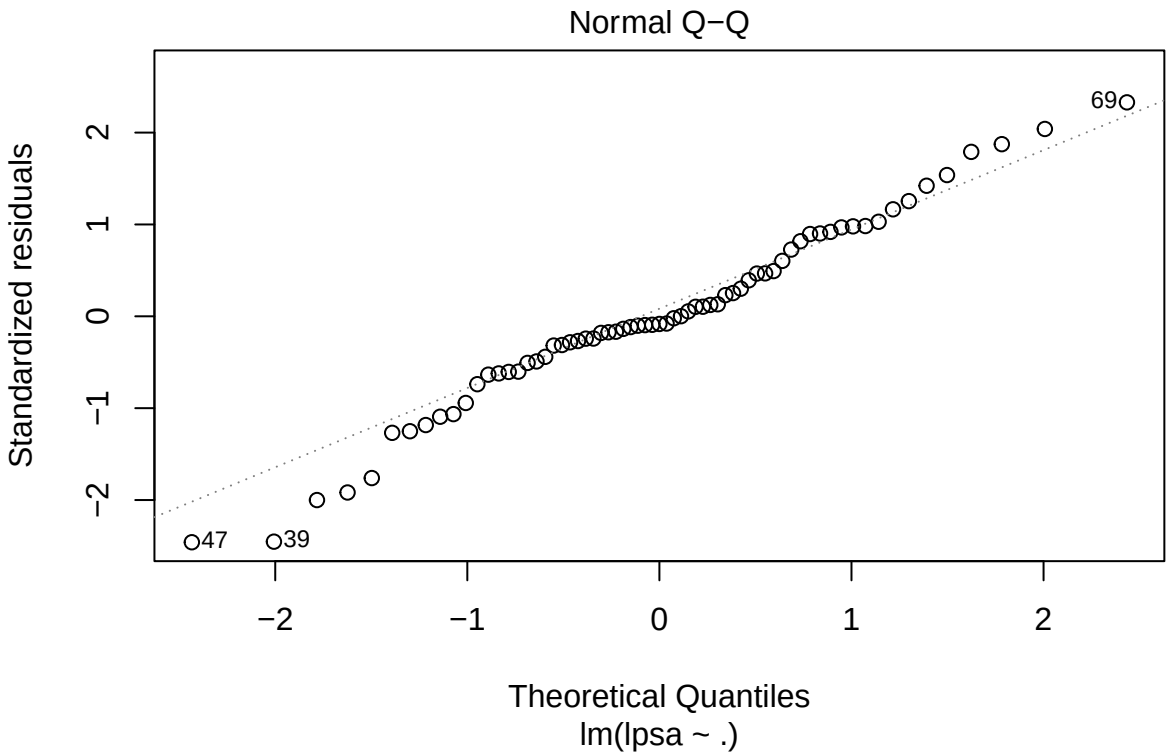
```
##
## Call:
## lm(formula = lpsa ~ ., data = dtrain)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.64870 -0.34147 -0.05424  0.44941  1.48675
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.46493    0.08931  27.598  < 2e-16
## lcavol       0.67953    0.12663   5.366 1.47e-06
## lweight      0.26305    0.09563   2.751  0.00792
## age         -0.14146    0.10134  -1.396  0.16806
## lbph         0.21015    0.10222   2.056  0.04431
## svi          0.30520    0.12360   2.469  0.01651
## lcp         -0.28849    0.15453  -1.867  0.06697
## gleason     -0.02131    0.14525  -0.147  0.88389
## pgg45        0.26696    0.15361   1.738  0.08755
##
## (Intercept) ***
## lcavol      ***
```

```
## lweight     **
## age
## lbph        *
## svi         *
## lcp         .
## gleason
## pgg45       .
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7123 on 58 degrees of freedom
## Multiple R-squared:  0.6944, Adjusted R-squared:  0.6522
## F-statistic: 16.47 on 8 and 58 DF,  p-value: 2.042e-12
```

```
plot(fit2,which=1)
```




```
plot(fit2,which=2)
```



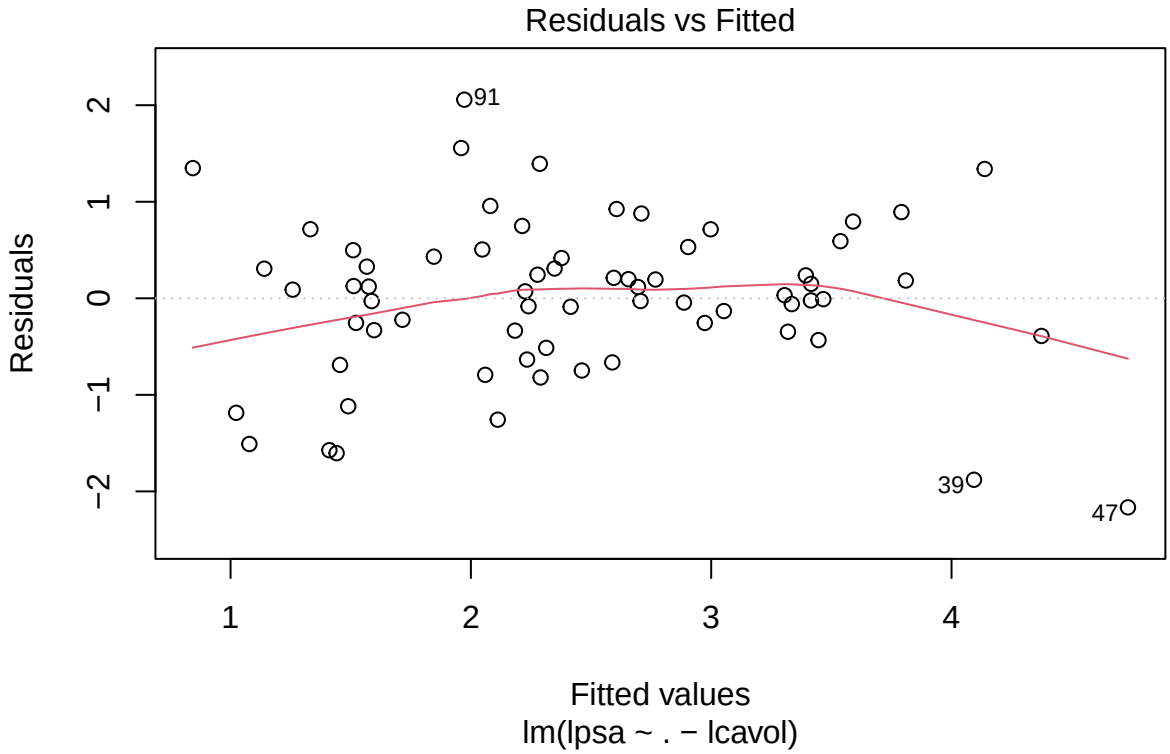
We run the multiple regression model without covariate `lcavol`. We print the `summary` and create TA and QQ plots.

```
fit3 <- lm(lpsa~.-lcavol,data=dtrain)
summary(fit2)
```

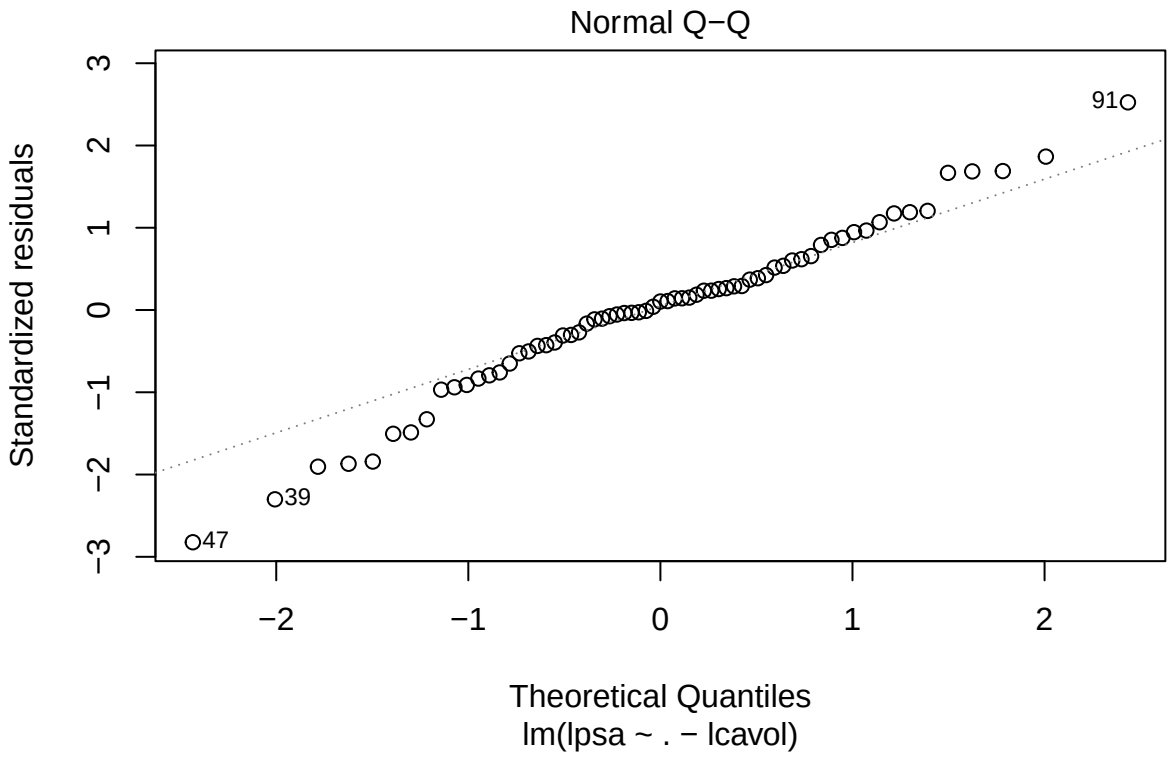
```
##
## Call:
## lm(formula = lpsa ~ ., data = dtrain)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.64870 -0.34147 -0.05424  0.44941  1.48675
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.46493    0.08931  27.598  < 2e-16
## lcavol       0.67953    0.12663   5.366 1.47e-06
## lweight      0.26305    0.09563   2.751  0.00792
## age         -0.14146    0.10134  -1.396  0.16806
## lbph         0.21015    0.10222   2.056  0.04431
## svi          0.30520    0.12360   2.469  0.01651
## lcp         -0.28849    0.15453  -1.867  0.06697
## gleason     -0.02131    0.14525  -0.147  0.88389
## pgg45        0.26696    0.15361   1.738  0.08755
##
## (Intercept) ***
```

```
## lcavol      ***
## lweight     **
## age
## lbph        *
## svi         *
## lcp         .
## gleason
## pgg45       .
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7123 on 58 degrees of freedom
## Multiple R-squared:  0.6944, Adjusted R-squared:  0.6522
## F-statistic: 16.47 on 8 and 58 DF,  p-value: 2.042e-12
```

```
plot(fit3,which=1)
```




```
plot(fit3,which=2)
```



Calculate the RSS.

```
sum(residuals(fit1)^2)
```

```
## [1] 44.52858
```

```
sum(residuals(fit2)^2)
```

```
## [1] 29.42638
```

```
sum(residuals(fit3)^2)
```

```
## [1] 44.03662
```

Compare the 3 models using the `anova` function.

```
anova(fit1,fit2,fit3)
```

```
## Analysis of Variance Table
##
## Model 1: lpsa ~ lcavol
## Model 2: lpsa ~ lcavol + lweight + age + lbph + svi + lcp + gleason +
##     pgg45
## Model 3: lpsa ~ (lcavol + lweight + age + lbph + svi + lcp + gleason +
##     pgg45) - lcavol
##   Res.Df    RSS Df Sum of Sq       F    Pr(>F)
## 1     65 44.529
## 2     58 29.426  7    15.102  4.2524 0.0007548
## 3     59 44.037 -1   -14.610 28.7971 1.469e-06
##
```

```
## 1
## 2 ***
## 3 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Validation of a prognostic model

In the previous section we developed 3 models to predict `lpsa`. In this section we explore the generalizability of the models.

1. Calculated the RMSE on the training data. Which model will perform best on future data?
2. Use the test data and make scatter plots of the observed against predicted outcomes. Use `ggplot` to create one plot per model and add the regression line (`geom_smooth`) and the "y=x" (`geom_abline`) line to the graph. This plot is also called "calibration plot". The model is "well" calibrated if the regression line agrees with the "y=x" line.
3. Repeat the plots in 2. but now focus on `predicted - observed` vs `predicted`.
4. Generate boxplots of `predicted - observed` for the 3 models. What do you conclude?
5. Calculate the generalization error, i.e., the RMSE on the test data.

Solution to the exercise.

We calculate the RMSEs on the training data. RMSE on training tells you how good the model "fits" the data. We cannot make any conclusion about the generalizability of the models based on RMSEs on training data.

```
RMSE(dtrain$lpsa,predict(fit1,newdata=dtrain))
```

```
## [1] 0.8152335
```

```
RMSE(dtrain$lpsa,predict(fit2,newdata=dtrain))
```

```
## [1] 0.6627215
```

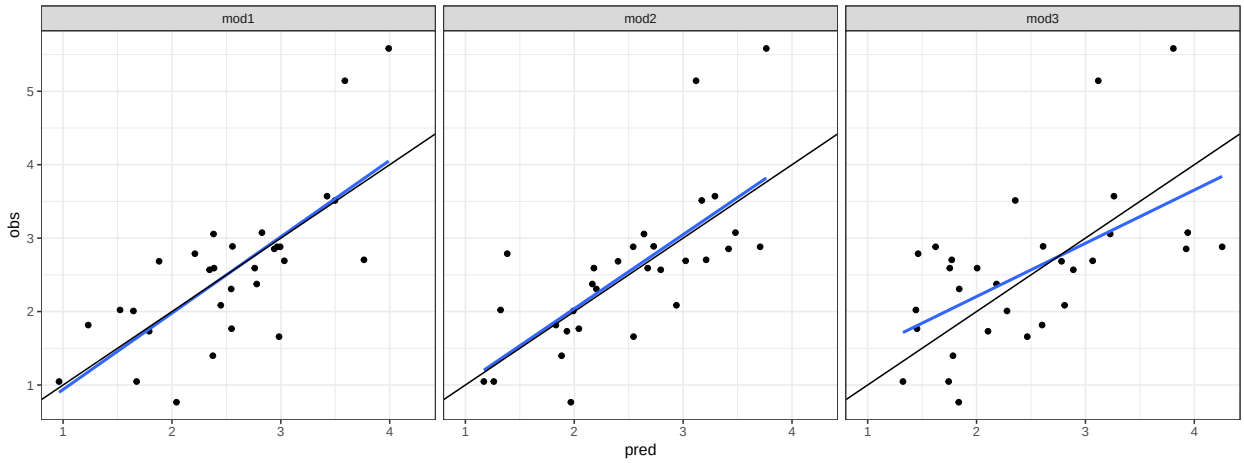
```
RMSE(dtrain$lpsa,predict(fit3,newdata=dtrain))
```

```
## [1] 0.8107176
```

We draw calibration plots for the 3 models. Model 3 does not calibrate well.

```
dd <- data.frame(pred=c(predict(fit1,newdata=dtest),
                        predict(fit2,newdata=dtest),
                        predict(fit3,newdata=dtest)),
                 obs = rep(dtest$lpsa,times=3),
                 model=rep(c("mod1","mod2","mod3"),each=nrow(dtest))
)
dd%>%
  ggplot(.,aes(x=pred,y=obs))+
  geom_point()+
  geom_smooth(se=FALSE,method="lm")+
  geom_abline(slope=1,intercept=0)+
  theme_bw()+
  facet_wrap(~model)
```

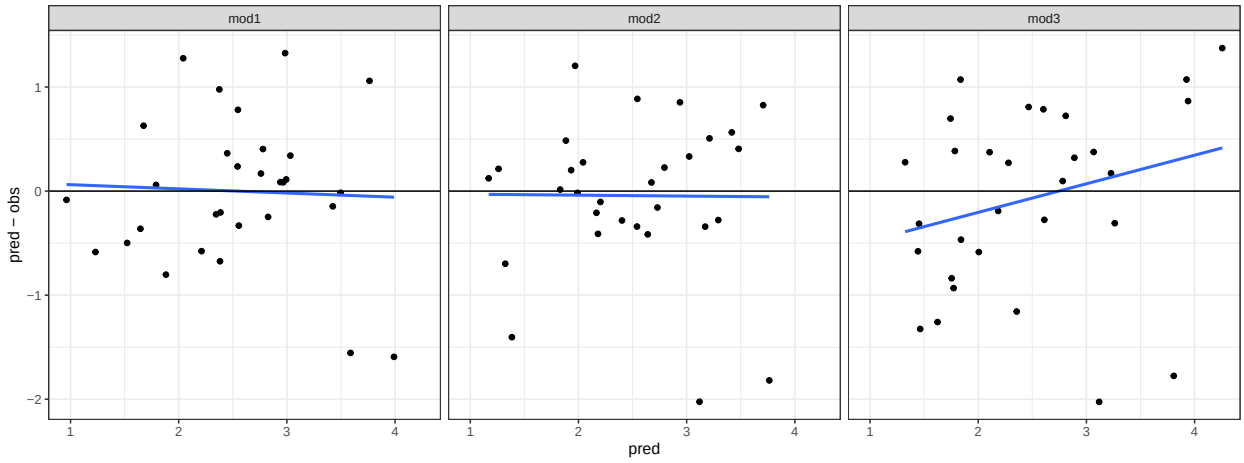
```
## `geom_smooth()` using formula 'y ~ x'
```



We draw Tukey-Anscombe plots.

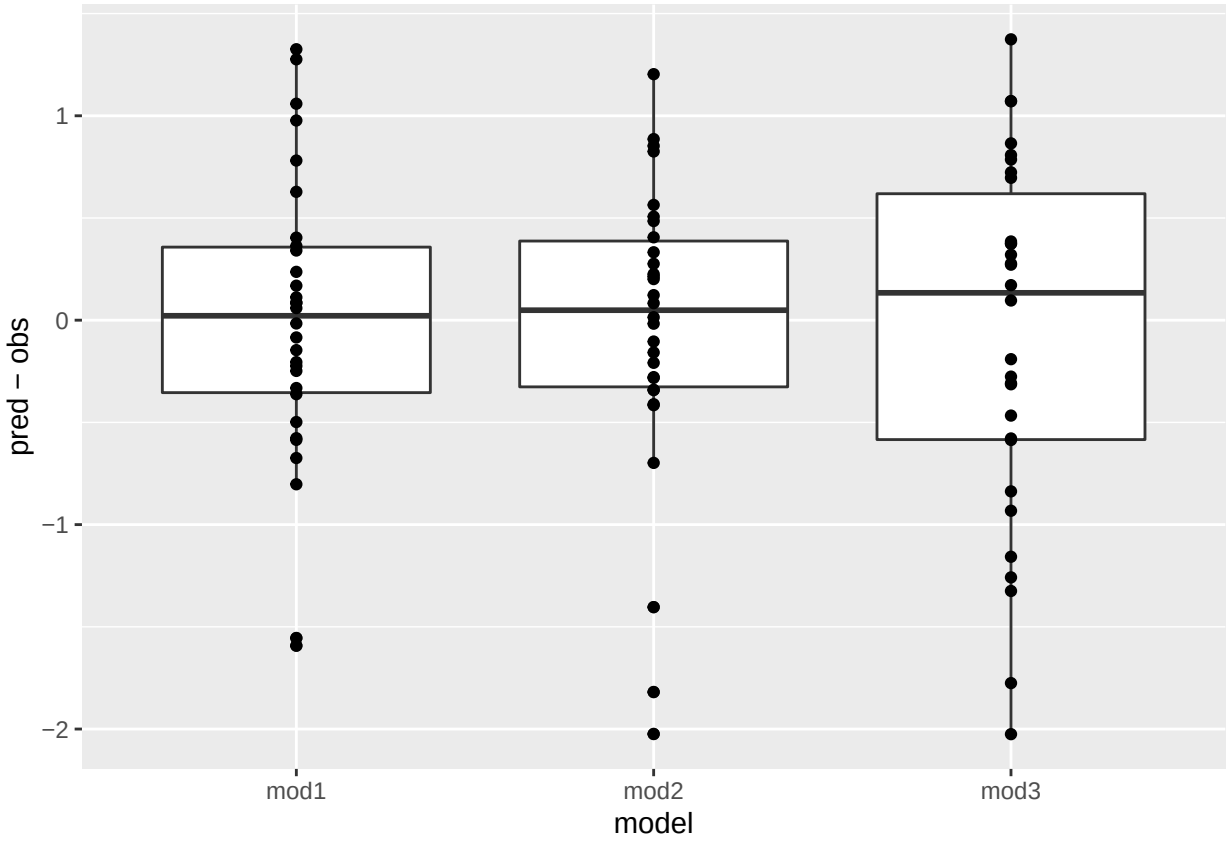
```
dd%>%
  ggplot(.,aes(x=pred,y=pred-obs))+
  geom_point()+
  geom_smooth(se=FALSE,method="lm")+
  geom_abline(slope=0,intercept=0)+
  theme_bw()+
  facet_wrap(~model)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Boxplots of predicted minus observed.

```
dd%>%
  ggplot(.,aes(x=model,y=pred-obs))+
  geom_boxplot()+
  geom_point()
```



Calculate RMSEs on test data.

```
RMSE(dtest$lpsa,predict(fit1,newdata=dtest))
```

```
## [1] 0.6926317
```

```
RMSE(dtest$lpsa,predict(fit2,newdata=dtest))
```

```
## [1] 0.7219931
```

```
RMSE(dtest$lpsa,predict(fit3,newdata=dtest))
```

```
## [1] 0.8685875
```

Simulated data and linear regression

In this exercise we explore linear regression using the following simulated data set.

```
set.seed(1)

# training data
x <-  matrix(rnorm(20 * 15), 20, 15)
y <- x[,1:4]%*%c(2,-2,2,-2)+rnorm(20)
dtrain <- data.frame(x,y)

# test data
x <-  matrix(rnorm(20 * 15), 20, 15)
y <- x[,1:4]%*%c(2,-2,2,-2)+rnorm(20)
dtest <- data.frame(x,y)
```

1. Can you describe the model behind the lines of code?
2. Use the training data to fit and explore the following 3 models: a) univariate regression model with covariate X1, b) multiple regression model with X1-X4 as covariates and c) multiple regression model including all covariates.
3. Repeat the calculations from exercise 2. What do you conclude?

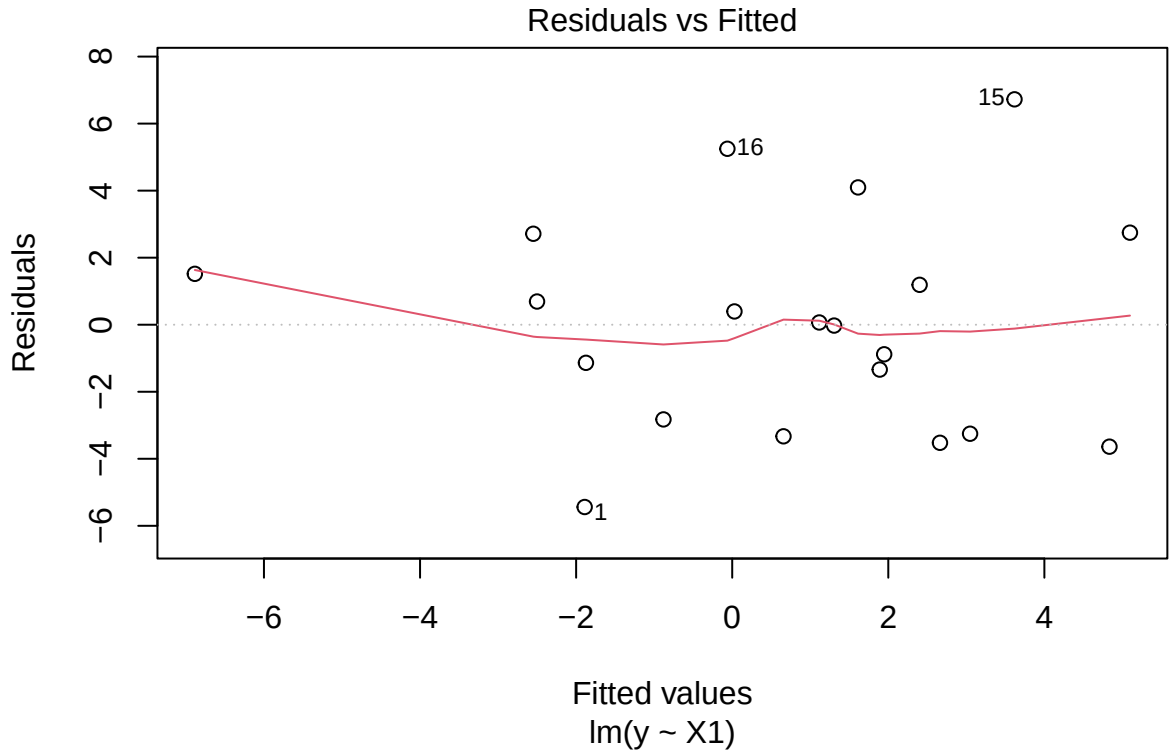
Solution to the exercise.

Fit the 3 models.

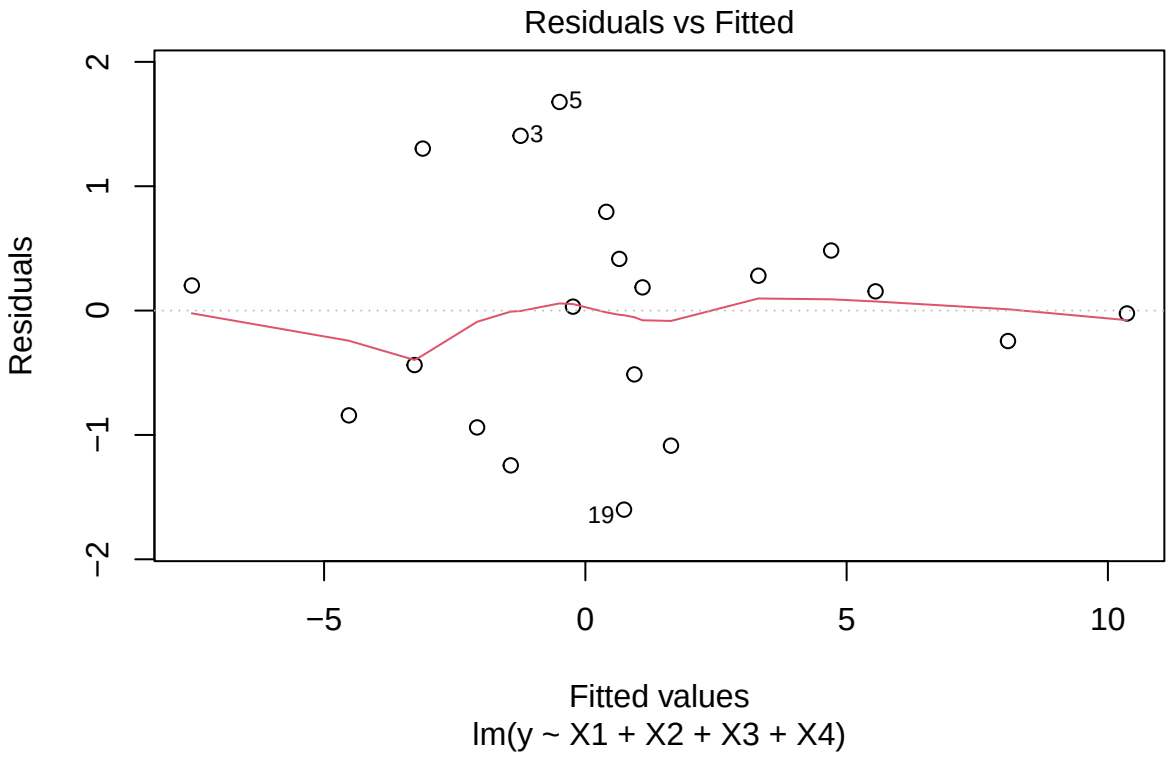
```
fit1 <- lm(y~X1,data=dtrain)
fit2 <- lm(y~X1+X2+X3+X4,data=dtrain)
fit3 <- lm(y~.,data=dtrain)
```

TA plots.

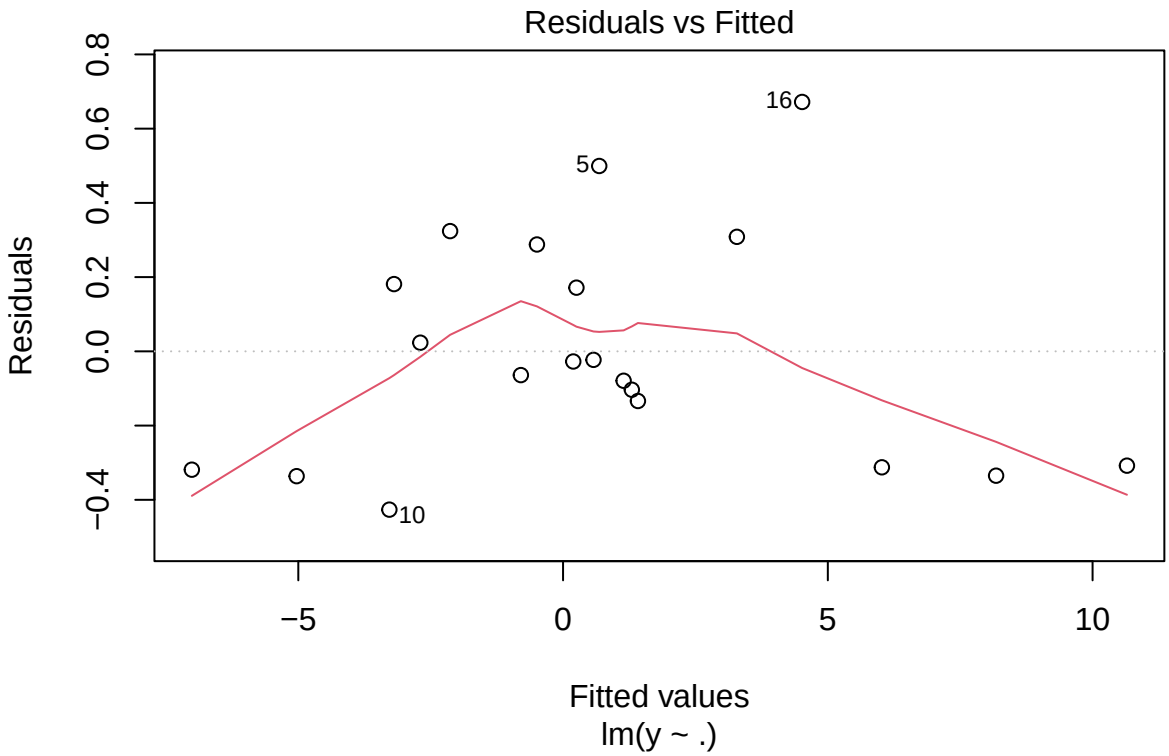
```
plot(fit1,which=1)
```




```
plot(fit2,which=1)
```




```
plot(fit3,which=1)
```



RMSEs on training data.

```
RMSE(dtrain$y,predict(fit1,newdata=dtrain))
```

```
## [1] 3.142926
```

```
RMSE(dtrain$y,predict(fit2,newdata=dtrain))
```

```
## [1] 0.8689516
```

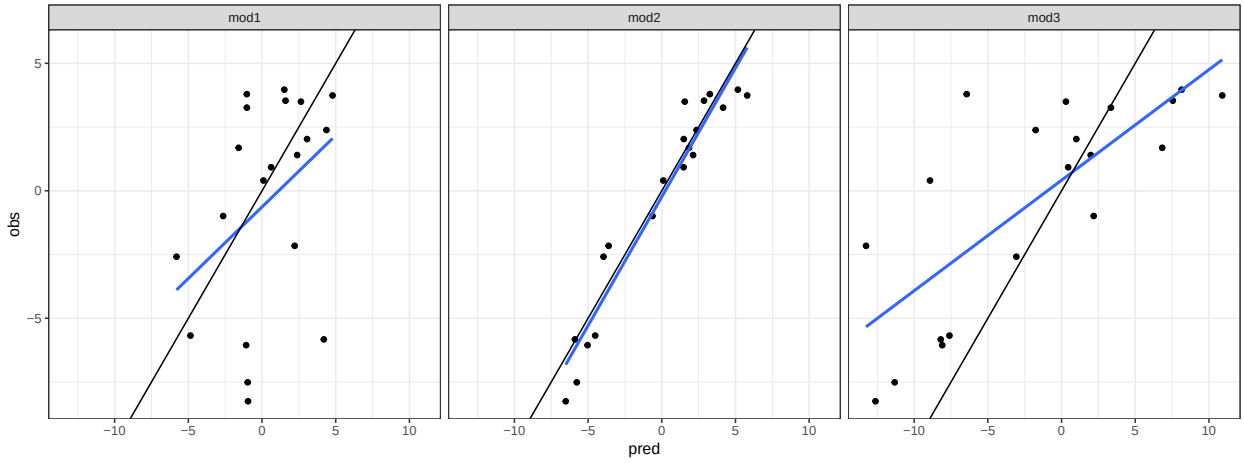
```
RMSE(dtrain$y,predict(fit3,newdata=dtrain))
```

```
## [1] 0.2990234
```

Calibration plots for the 3 models. Only model 2 is well calibrated.

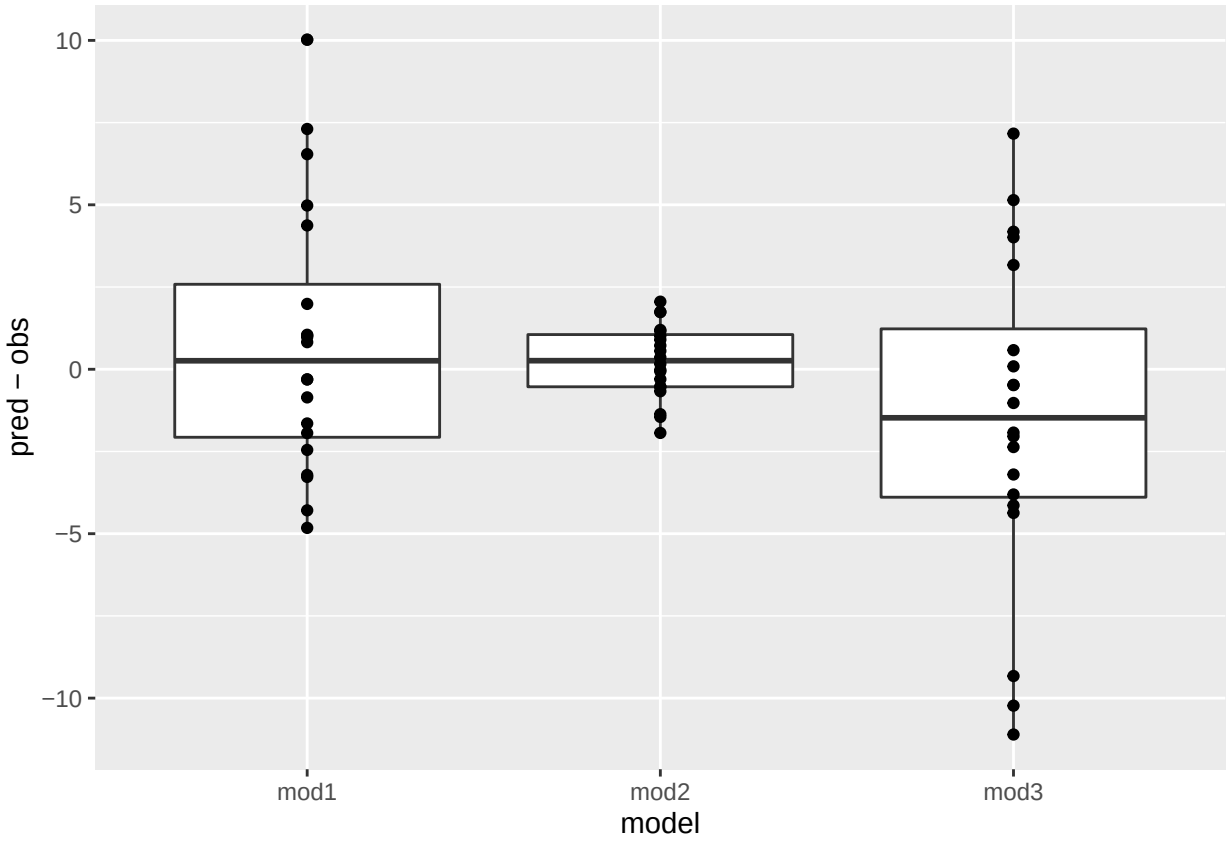
```
dd <- data.frame(pred=c(predict(fit1,newdata=dtest),
                        predict(fit2,newdata=dtest),
                        predict(fit3,newdata=dtest)),
                 obs = rep(dtest$y,times=3),
                 model=rep(c("mod1","mod2","mod3"),each=nrow(dtest))
)
dd%>%
  ggplot(.,aes(x=pred,y=obs))+
  geom_point()+
  geom_smooth(se=FALSE,method="lm")+
  geom_abline(slope=1,intercept=0)+
  theme_bw()+
  facet_wrap(~model)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Boxplots of `predicted - observed`.

```
dd%>%
  ggplot(.,aes(x=model,y=pred-obs))+
  geom_boxplot()+
  geom_point()
```



Calculate the RMSE on test data.

```
RMSE(dtest$y,predict(fit1,newdata=dtest))
```

```
## [1] 4.027371
```

```
RMSE(dtest$y,predict(fit2,newdata=dtest))
```

```
## [1] 1.111998
```

```
RMSE(dtest$y,predict(fit3,newdata=dtest))
```

```
## [1] 5.063148
```

Calculus, optimization and OLS

1. Consider the function $f(x) = 2x^2 + x - 5$. Draw a plot of the function.
2. Use `optimize` to find the minimum of $f(x)$.
3. Obtain the minimum of $f(x)$ by taking the derivative and setting equal to zero.
4. Show that $\|a\|_2^2 = a^T a$.
5. Use the result in 4. and show that $\mathbf{RSS}(\beta) = \mathbf{y^T y} - \mathbf{2yX}\beta + \beta^{\mathbf{T}}\mathbf{X^T X}\beta$.
6. Invoke the result obtained in 4. and show that

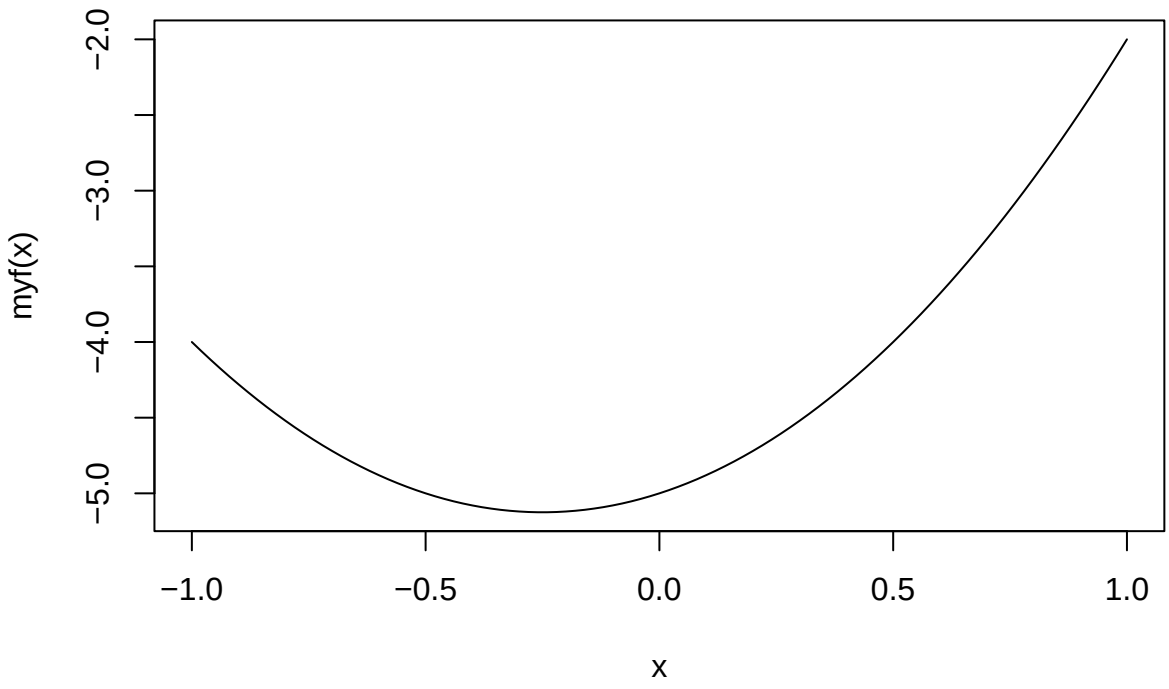
$$\frac{\partial}{\partial \beta}\mathbf{RSS}(\beta) = -\mathbf{2X^T y} + \mathbf{2X^T X}\beta.$$

   Hint: review the "Identities" section of Wikipedia.
7. Do you understand the derivation of the least squares estimator?

Solution to the exercise.

Plot of the function.

```
myf <- function(x){
  2*x^2 + x -5
}
curve(myf,from=-1,to=1)
```




```
optimize(myf,interval=c(-5,5))
```

```
## $minimum
## [1] -0.25
##
## $objective
## [1] -5.125
```

Prostate cancer data and regularization

1. Read the prostate cancer data set.
2. Run OLS, Best Subset selection (use the `leaps` package), Ridge regression (`glmnet` with `alpha=0`) and Lasso regression (`glmnet` with `alpha=1`).
3. Print the coefficients as a table.
4. Calculate for each method the test error and compare the results with Table 3.3. in @elements.

The solution to this exercise.

Read the data set and create training and test data.

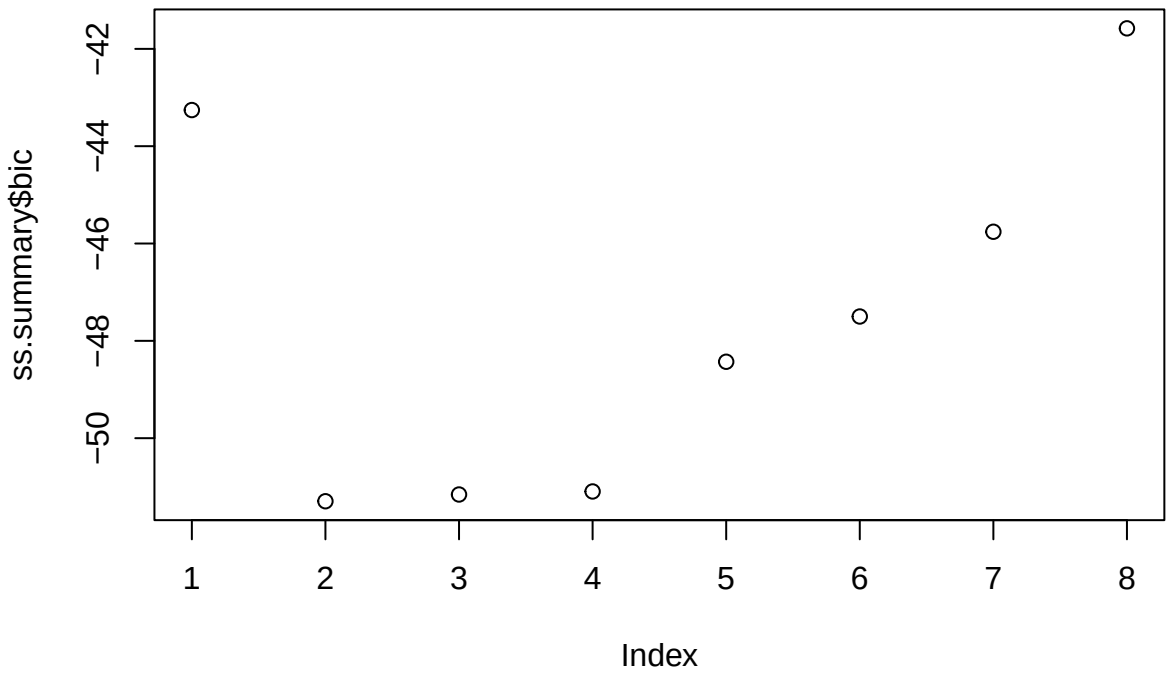
```
dat <- read.csv("data/prostate.csv")
dat <- cbind(scale(dat[,-c(9:10)]),dat[,c(9,10)])
dtrain <- dat[dat$train,-10]
xtrain <- data.matrix(dtrain[,-9])
ytrain <- dtrain$lpsa
dtest <- dat[!dat$train,-10]
xtest <- data.matrix(dtest[,-9])
ytest <- dtest$lpsa
```

Perform OLS regression, Best Subset selection, Ridge regression and Lasso regression.

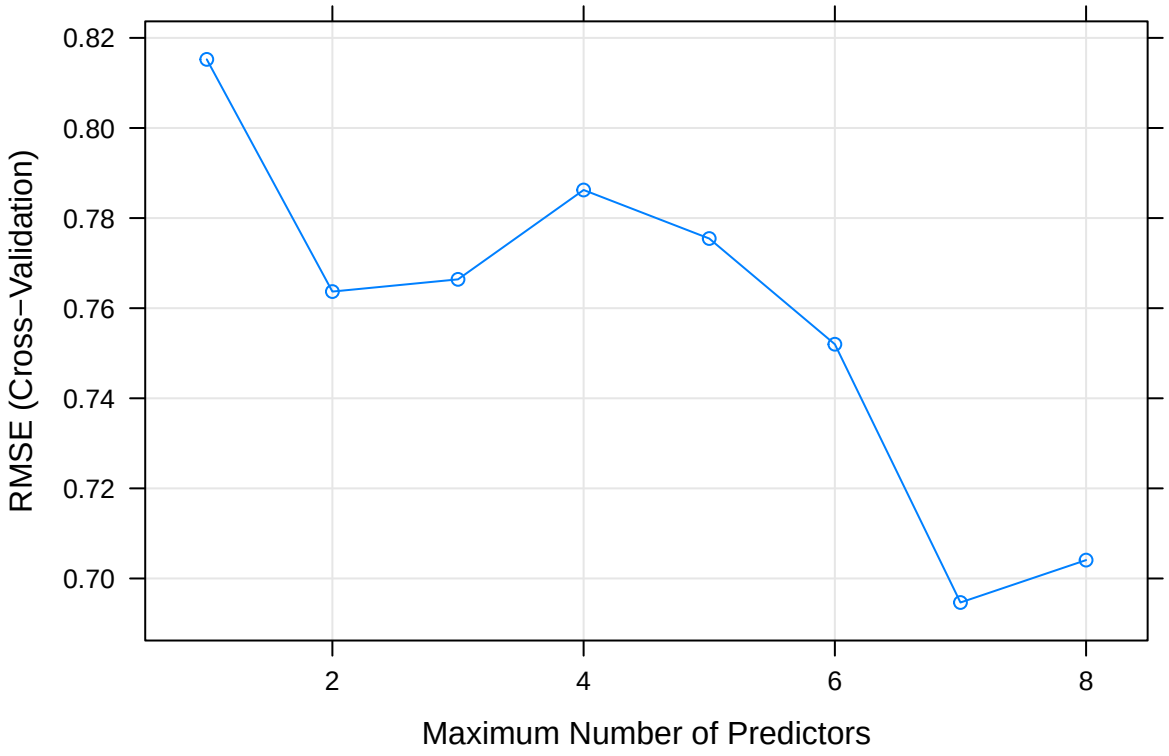
```
# OLS regression
fit.lm <- lm(lpsa~.,data=dtrain)
coef.lm <- coef(fit.lm)
terr.lm <- mean((predict(fit.lm,newdata = dtest)-dtest$lpsa)^2)

# best subset selection with leaps
library(leaps)
subset.leaps<-regsubsets(lpsa~.,data=dtrain,method="forward")
ss.summary <- summary(subset.leaps)
coef.ss.bic <- coef(subset.leaps, which.min(ss.summary$bic))
fit.ss.bic <- lm(lpsa~.,data=dtrain[,c("lpsa",names(coef.ss.bic)[-1])])
terr.ss.bic <- mean((predict(fit.ss.bic,newdata = dtest)-dtest$lpsa)^2)

# best subset selection with caret package (10-fold cv)
library(caret)
set.seed(1) # set seed for reproducibility
train.control <- trainControl(method = "cv", number = 10)# set up repeated k-fold
subset.caret <- train(lpsa ~., data = dtrain,
                      method = "leapBackward",
                      tuneGrid = data.frame(nvmax = 1:8),
                      trControl = train.control
)
#subset.caret$results
#summary(subset.caret$finalModel)
coef.ss.cv <- coef(subset.caret$finalModel, subset.caret$bestTune$nvmax)
fit.ss.cv <- lm(lpsa~.,data=dtrain[,c("lpsa",names(coef.ss.cv)[-1])])
terr.ss.cv <- mean((predict(fit.ss.cv,newdata = dtest)-dtest$lpsa)^2)

# Ridge regression
set.seed(1)
library(glmnet)
fit.ridge <- glmnet(xtrain,ytrain,alpha=0)
fit.ridge.cv <- cv.glmnet(xtrain,ytrain,alpha=0)
coef.ridge <- coef(fit.ridge,s=fit.ridge.cv$lambda.min)
terr.ridge <- mean((as.vector(predict(fit.ridge,newx=xtest,s=fit.ridge.cv$lambda.1se))-ytest)^2)

# Lasso regression
set.seed(1)
fit.lasso <- glmnet(xtrain,ytrain,alpha=1)
fit.lasso.cv <- cv.glmnet(xtrain,ytrain,alpha=1)
coef.lasso <- coef(fit.lasso,s=fit.ridge.cv$lambda.min)
terr.lasso <- mean((as.vector(predict(fit.lasso,newx=xtest,s=fit.lasso.cv$lambda.1se))-ytest)^2)
```

The next two figures show BIC and CV error for the Best Subset selection approach (tuning parameter are the different subset sizes).

```
plot(ss.summary$bic)
```

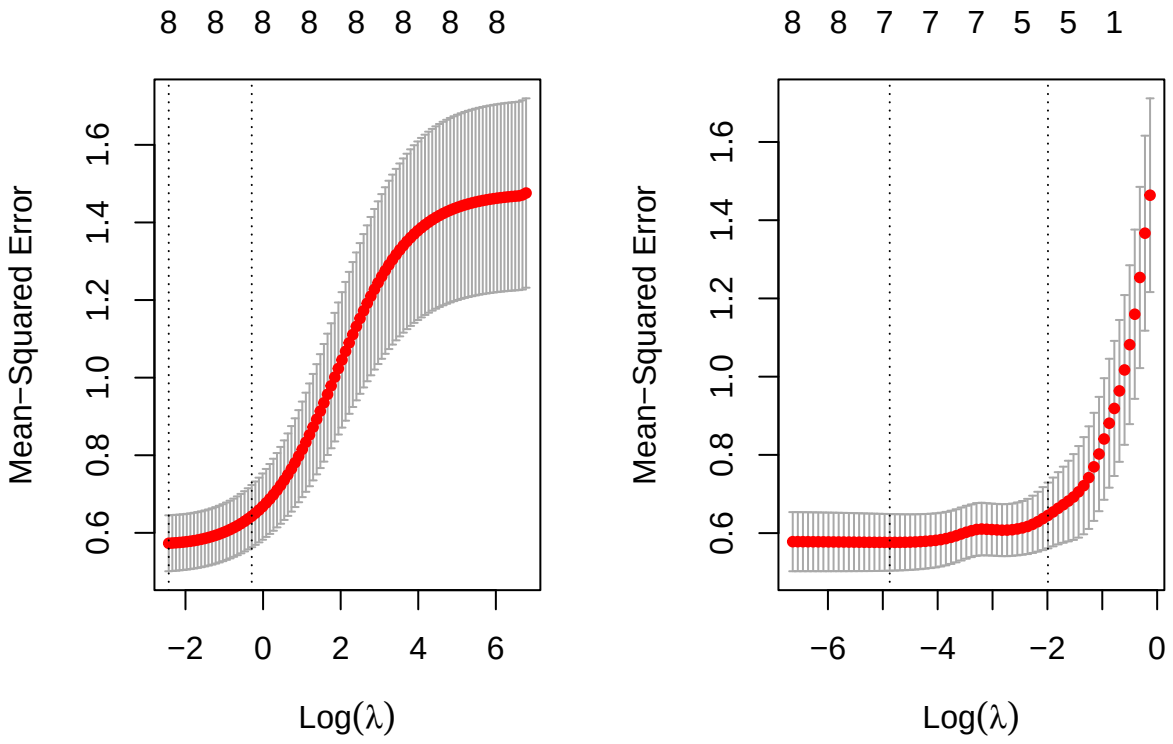



```
plot(subset.caret)
```



The next two figures show the CV error for Ridge and Lasso regression.

```
par(mfrow=c(1,2))
plot(fit.ridge.cv)
plot(fit.lasso.cv)
```



Next we show the table of regression coefficients obtained using the different methods.

```
res <- data.frame(OLS=coef.lm,SSBIC=0,SSCV=0,RIDGE=0,LASSO=0)
res[names(coef.ss.bic),"SSBIC"] <- coef.ss.bic
res[names(coef.ss.cv),"SSCV"] <- coef.ss.cv
res[,"RIDGE"] <- as.numeric(coef.ridge)
res[,"LASSO"] <- as.numeric(coef.lasso)
kable(res,digits = 3,booktabs=TRUE)
```

	OLS	SSBIC	SSCV	RIDGE	LASSO
(Intercept)	2.465	2.477	2.467	2.467	2.465
lcavol	0.680	0.740	0.676	0.581	0.547
lweight	0.263	0.316	0.265	0.258	0.211
age	-0.141	0.000	-0.145	-0.110	0.000
lbph	0.210	0.000	0.210	0.200	0.116
svi	0.305	0.000	0.307	0.281	0.178
lcp	-0.288	0.000	-0.287	-0.163	0.000
gleason	-0.021	0.000	0.000	0.012	0.000
pgg45	0.267	0.000	0.252	0.200	0.070

Finally we show the test errors of the different methods.

```
res <- data.frame(OLS=terr.lm,
                  SSBIC=terr.ss.bic,SSCV=terr.ss.cv,
                  RIDGE=terr.ridge,LASSO=terr.lasso)
kable(res,digits = 3,booktabs=TRUE)
```

OLS	SSBIC	SSCV	RIDGE	LASSO
0.521	0.492	0.517	0.508	0.454

Closed form solution for Ridge regression

1. Show that the Ridge optimization problem has the closed form solution

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

Hint: calculate the gradient of the loss function $\ell_{\text{Ridge}}(\beta|\mathbf{y}, \mathbf{X}) = \text{RSS}(\beta) + \lambda\|\beta\|_2^2$, set equal to zero and solve for β.

2. Use the code below to generate simulated data. Use the formula from the script to calculate the Ridge coefficients for $\lambda = 35$. Compare the coefficients with those obtained using `glmnet`. Hint: Read the following blog on how to scale the λ.

```
set.seed(1)

# simulate data
n <- 20
p <- 15
x <-  matrix(rnorm(n * p), n, p)
y <- x[,1:4]%*%c(2,-2,2,-2)+rnorm(n)
```

Solution to the exercise.

Closed form solution for Ridge regression

$$\hat{\beta}_c^{\text{Ridge}} = \underset{\|\beta\|_2^2 \le c}{\operatorname{argmin}} \; \|y - X\beta\|_2^2$$

Lagrange multiplier

$\Leftrightarrow$ minimize $\underbrace{\|y - X\beta\|_2^2 + \lambda \|\beta\|_2^2}_{\equiv \ell(\beta)}$

$$\Leftrightarrow \quad \frac{\partial \ell(\beta)}{\partial \beta} \overset{!}{=} 0$$

$$\|y - X\beta\|_2^2 = (y - X\beta)^T (y - X\beta) = y^T y - y^T X\beta - \beta^T X^T y + \beta^T X^T X \beta$$

$\frac{\partial}{\partial \beta}$: 0 ; $X^T y$; $X^T y$; $2X^T X\beta$

$$\Rightarrow \frac{\partial}{\partial \beta} \|y - X\beta\|_2^2 = -2X^T y + 2X^T X\beta$$

$$\|\beta\|_2^2 = \sum_{j=1}^{p} \beta_j^2 \quad \Rightarrow \frac{\partial}{\partial \beta} \|\beta\|_2^2 = 2\beta$$

$$\Rightarrow \frac{\partial \ell(\beta)}{\partial \beta} = -2X^T y + 2X^T X\beta + \lambda 2\beta \overset{!}{=} 0$$

$$\Leftrightarrow \quad X^T X\beta + \lambda\beta \overset{!}{=} X^T y$$

$$\Leftrightarrow \quad (X^T X + \lambda I)\beta \overset{!}{=} X^T y$$

$$\Leftrightarrow \quad \hat{\beta} = (X^T X + \lambda I)^{-1} X^T y$$

□

We obtain the Ridge coefficients using `glmnet`.

```
my.lambda <- 35
fit.ridge <- glmnet(x,y,alpha=0,lambda=my.lambda,
                    intercept=FALSE,standardize = FALSE,thresh = 1e-20,exact=TRUE)
coef.ridge <- as.vector(coef(fit.ridge))[-1]
head(coef.ridge)
```

```
## [1]  0.27330465 -0.24799766  0.19686435
## [4] -0.21942808  0.05302251 -0.02458886
```

Next we calculate the coefficients based on the formula from the script. Note that we need to re-scale the lambda.

```
sd_y <- sqrt(var(y)*(n-1)/n)[1,1]
my.lambda2 <- n*my.lambda/sd_y
coef.ridge2 <- solve(t(x)%*%x+my.lambda2*diag(nrow=ncol(x)))%*%t(x)%*%y
head(coef.ridge2)[,1]
```

```
## [1]  0.27031012 -0.24528806  0.19469189
## [4] -0.21696366  0.05244081 -0.02429057
```

Bayesian interpretation of Ridge regression (difficult)

1. Write down the log-likelihood of the linear regression model. Note: $Y_i = X_i^T\beta + \epsilon_i$, where $\epsilon_1, \ldots, \epsilon_n$ iid $N(0, \sigma^2)$ and $\mathbf{X}$ is a fixed $n \times p$ design matrix.
2. Find the expression for the maximum likelihood estimator.
3. Assuming a prior distribution $\beta_1, \ldots, \beta_p$ iid $\sim N(0, \tau^2)$, derive the posterior distribution of β and show that the maximum a posteriori estimator (MAP) coincides with the Ridge estimator.

The solution to this exercise.

Likelihood of linear regression

$$p(y_i | x_i, \beta, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(- \frac{(y_i - x_i^T \beta)^2}{2\sigma^2} \right)$$

$$\log p(y_i | x_i, \beta, \sigma^2) \propto - \log \sigma - \frac{(y_i - x_i^T \beta)^2}{2\sigma^2}$$

$$\log p(y | X, \beta, \sigma^2) = \sum_{i=1}^{n} \log p(y_i | x_i, \beta, \sigma^2) \propto -n \log \sigma - \frac{1}{2\sigma^2} \| Y - X\beta \|_2^2$$

Maximum Likelihood for linear regression

A) $\max_{\beta} \log p(y | X, \beta, \sigma^2) \iff \min_{\beta} \| Y - X\beta \|_2^2 \iff \hat{\beta}^{OLS}$

B) $\max_{\sigma} \log p(y | X, \beta, \sigma^2) \iff \frac{\partial}{\partial \sigma} \log p(y | X, \beta, \sigma^2) \overset{!}{=} 0$

$$\frac{\partial}{\partial \sigma} \log p(y | X, \hat{\beta}^{OLS}, \sigma^2) = \frac{-n}{\sigma} + \frac{1}{\sigma^3} \| y - X\hat{\beta}^{OLS} \|^2 \overset{!}{=} 0$$

$$\hat{\sigma}^2 = \frac{\| y - X\hat{\beta}^{OLS} \|^2}{n}$$

Posteriori distribution

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

$$\log p(\beta | y, X) \propto \log p(y | X, \beta) + \log p(\beta)$$

$$\propto -n \log \sigma - \frac{1}{2\sigma^2} \| Y - X\beta \|_2^2 - p \log \tau - \frac{1}{2\tau^2} \| \beta \|_2^2$$

$$\rightsquigarrow \max_{\beta} \log p(\beta | y, X) \iff \min_{\beta} \| Y - X\beta \|_2^2 + \underbrace{\frac{\sigma^2}{\tau^2}}_{=\lambda} \| \beta \|_2^2$$

Elastic net mixing parameter and cross-validation

1. Load the `hdi` package and read the riboflavin data set (`?riboflavin`).
2. Run the Lasso and generate the trace plot.
3. Run the Elastic net with mixing parameters $\alpha = 0.25, 0.5, 0.75$ and 1 and compare the cross-validation curves. Hint: use the `foldid` argument in `glmnet`.
4. Show the selected genes for the best performing model.

The solution to this exercise.

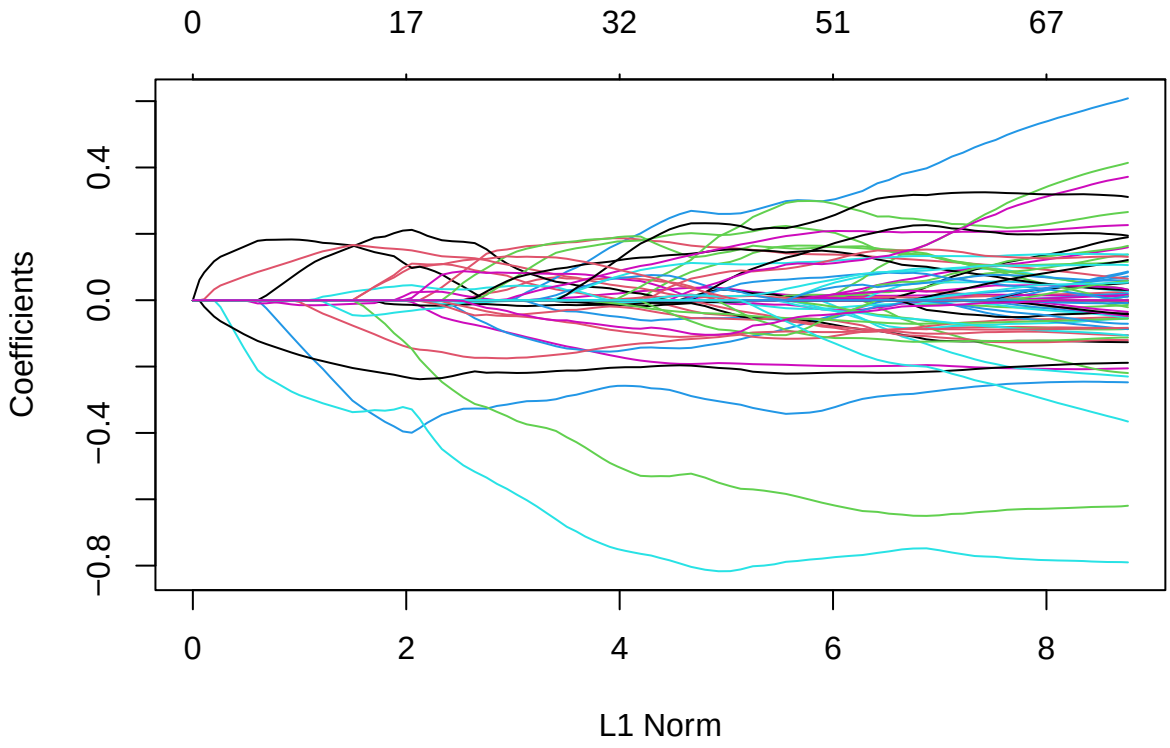
We first load the data and check the data structure.

```
library(hdi)
library(glmnet)
riboflavin <- readRDS(file="data/riboflavin.rds")
str(riboflavin)
```

```
## 'data.frame':    71 obs. of  2 variables:
##  $ y: num  -6.64 -6.95 -7.93 -8.29 -7.31 ...
##  $ x: 'AsIs' num [1:71, 1:4088] 8.49 7.64 8.09 7.89 6.81 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:71] "b_Fbat107PT24.CEL" "b_Fbat107PT30.CEL" "b_Fbat107PT48.CEL" "b_Fbat107PT52.CEL
##   .. ..$ : chr [1:4088] "AADK_at" "AAPA_at" "ABFA_at" "ABH_at" ...
```

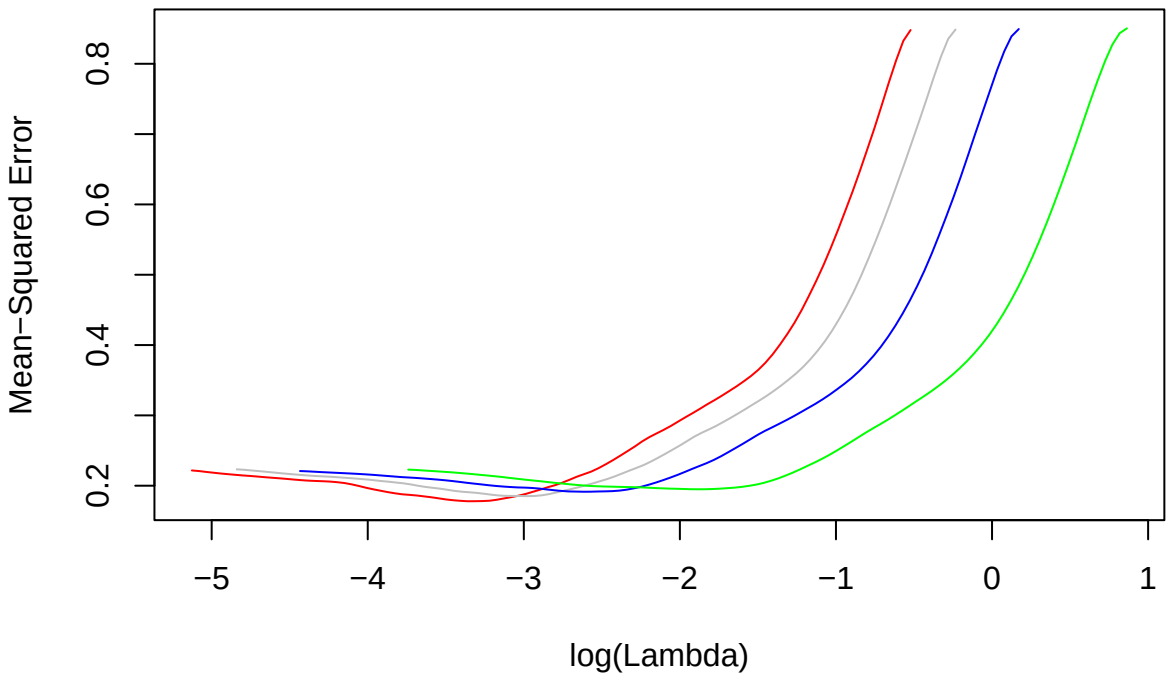
Next we setup the design matrix and the response variable and we run the Lasso.

```
x <- riboflavin[,-1]
y <- riboflavin[,1]
fit <- glmnet(x = x, y = y)
plot(fit)
```



We run 10-fold cross-validation for the different mixing parameters and plot the error curves.

```
set.seed(1)
n.fold <- 10
foldid <- sample(1:n.fold, size = length(y), replace = TRUE)
cv1 <- cv.glmnet(x, y, foldid = foldid, alpha = 1)
cv2  <- cv.glmnet(x, y, foldid = foldid, alpha = 0.75)
cv3  <- cv.glmnet(x, y, foldid = foldid, alpha = 0.5)
cv4  <- cv.glmnet(x, y, foldid = foldid, alpha = 0.25)

t.lambdarange <- range(log(c(cv1$lambda,
                             cv2$lambda,
                             cv3$lambda,
                             cv4$lambda)))
t.crange <- range(c(cv1$cvm,cv2$cvm,cv3$cvm,cv4$cvm))
plot(log(cv1$lambda), cv1$cvm ,
     pch = 19, col = "red",
     xlab = "log(Lambda)",
     ylab=cv1$name,
     type="l",
     xlim=t.lambdarange,
     ylim=t.crange)
lines(log(cv2$lambda), cv2$cvm, pch = 19, col = "grey")
lines(log(cv3$lambda) , cv3$cvm , pch = 19, col = "blue")
lines(log(cv4$lambda) , cv4$cvm , pch = 19, col = "green")
```



Finally, we print the gene names of the non-zero coefficients.

```
## Get selected genes
b <- as.matrix(coef(cv1))
rownames(b)[b!=0]
```

```
##  [1] "(Intercept)" "ARGF_at"     "DNAJ_at"
##  [4] "GAPB_at"     "LYSC_at"     "PKSA_at"
##  [7] "SPOIISA_at"  "SPOVAA_at"   "XHLB_at"
## [10] "XKDS_at"     "XTRA_at"     "YBFI_at"
## [13] "YCDH_at"     "YCGO_at"     "YCKE_at"
## [16] "YCLB_at"     "YCLF_at"     "YDDH_at"
## [19] "YDDK_at"     "YEBC_at"     "YEZB_at"
## [22] "YFHE_r_at"   "YFIR_at"     "YHDS_r_at"
## [25] "YKBA_at"     "YOAB_at"     "YQJU_at"
## [28] "YRVJ_at"     "YTGB_at"     "YURQ_at"
## [31] "YXLD_at"     "YXLE_at"     "YYDA_at"
```

```
## By default, the selected variables are based on the largest value of
## lambda such that the cv-error is within 1 standard error of the minimum
```

Ridge and Lasso for the orthonormal design (difficult)

1. Calculate the Ridge and the Lasso solution for the special case of an orthonormal design matrix.

The solution to this exercise.

Ridge and Lasso for orthonormal design

orthonormal design $\Rightarrow \boxed{X^T X = I}$

$\hat{\beta}^{OLS} = (X^T X)^{-1} X^T y = X^T y$ Note: $\hat{\beta}^{OLS\,T} \hat{\beta}^{OLS} = y^T X X^T y$

$$\| y - X\beta \|_2^2 = (y - X\beta)^T (y - X\beta) =$$

$$= y^T y - y^T X\beta - \beta^T X^T y + \beta^T X^T X \beta$$

$$= y^T y - 2\beta^T \underbrace{X^T y}_{\hat{\beta}^{OLS}} + \beta^T \beta$$

$$\propto -2\beta^T \hat{\beta}^{OLS} + \beta^T \beta$$

Ridge: $\frac{\partial}{\partial \beta} \left(-2\beta_j \hat{\beta}_j^{OLS} + \beta_j^2 + \lambda \beta_j^2 \right) \stackrel{!}{=} 0$

$$-2\hat{\beta}_j^{OLS} + 2\beta_j + 2\lambda\beta_j \stackrel{!}{=} 0$$

$$\Rightarrow (1+\lambda)\beta_j \stackrel{!}{=} \hat{\beta}_j^{OLS}$$

$$\Rightarrow \hat{\beta}_j^{Ridge} \stackrel{!}{=} \frac{1}{1+\lambda} \hat{\beta}_j^{OLS}$$

Lasso: $\frac{\partial}{\partial \beta} \left(-2\beta_j \hat{\beta}_j^{OLS} + \beta_j^2 + \lambda |\beta_j| \right) \stackrel{!}{=} 0$

If $\beta_j \neq 0$ $\quad -\hat{\beta}_j^{OLS} + \beta_j + \frac{\lambda}{2} \operatorname{sgn}(\beta_j) \stackrel{!}{=} 0$

$$\Rightarrow \hat{\beta}_j^{Ridge} = \operatorname{sgn}(\hat{\beta}_j^{OLS}) \left(|\hat{\beta}_j^{OLS}| - 0.5\lambda \right)_+$$

P-values for high-dimensional regression (difficult)

The Lasso does not provide p-values for the regression coefficients. In this exercise we use the riboflavin data set to explore a neat approach to obtain p-values. The research paper which proposed the method is available here.

1. Split the riboflavin data set into two halves ("data splitting").
2. On the first half: run `glmnet` and select the top genes ("screening").
3. On the second half: take the selected top genes as covariates, run ordinary least squares and report the p-values ("p-value calculation").
4. Write a function combining steps 1-3 and re-run these steps 100 times.
5. For a single gene, plot a histogram of the p-values.
6. For each gene calculate the 10th percentile of the p-values ("p-value aggregation").
7. Use the function `hdi` with `method="multi.split"` to calculate p-values.

Solution to the exercise.

Start by splitting the data set into two halves.

```
library(multcomp)
riboflavin <- readRDS(file="data/riboflavin.rds")
str(riboflavin)
```

```
## 'data.frame':    71 obs. of  2 variables:
##  $ y: num  -6.64 -6.95 -7.93 -8.29 -7.31 ...
##  $ x: 'AsIs' num [1:71, 1:4088] 8.49 7.64 8.09 7.89 6.81 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:71] "b_Fbat107PT24.CEL" "b_Fbat107PT30.CEL" "b_Fbat107PT48.CEL" "b_Fbat107PT52.CEL"
##   .. ..$ : chr [1:4088] "AADK_at" "AAPA_at" "ABFA_at" "ABH_at" ...
```

```
x <- riboflavin$x
colnames(x) <- make.names(colnames(x))
y <- riboflavin$y

# split data into two halves
set.seed(1)
n <- length(y)
in_sample <- sample(seq(n),size=ceiling(n/2))
x_in <- x[in_sample,]
y_in <- y[in_sample]
x_out <- x[-in_sample,]
y_out <- y[-in_sample]
```

Perform variable screening based on the first half.

```
# screening
fit.cv <- cv.glmnet(x_in, y_in)
bhat <- as.matrix(coef(fit.cv,s="lambda.min"))[-1,]
hatact <- names(bhat[bhat!=0])
```

Next, calculated p-values based on the second half.

```
# testing
t.dat <- data.frame(cbind(x_out[,hatact,drop=FALSE]))
t.dat$y <- y_out
fit.lm <- lm(y~.,data=t.dat)
fit.glht <- glht(fit.lm)
pvals <- summary(fit.glht, test = adjusted("bonferroni"))$test$pvalues
pvals
```

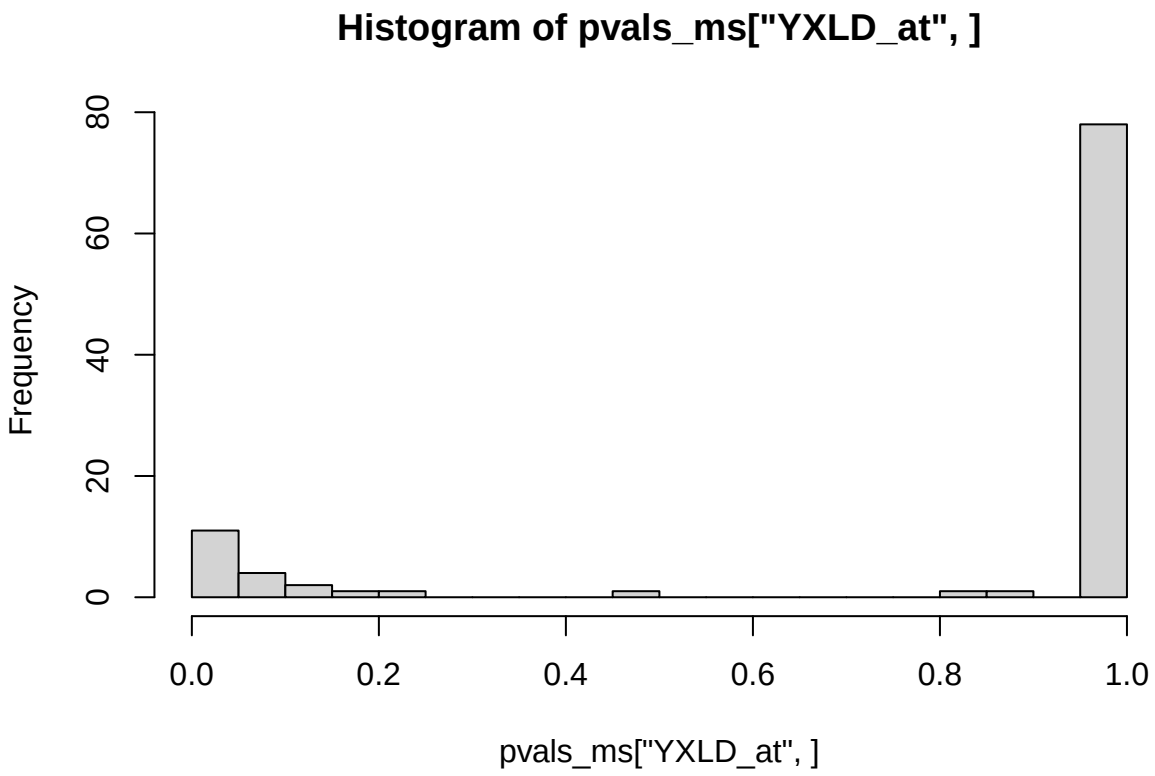
```
## (Intercept)     LYTR_at     XHLA_at     YCKE_at
##   1.0000000   1.0000000   0.1561028   1.0000000
##     YDDM_at     YHCL_at     YJCI_at     YOAE_at
##   1.0000000   1.0000000   1.0000000   1.0000000
##     YXLD_at     YXLG_at
##   1.0000000   1.0000000
```

Write a function in order to re-run the previous steps many times.

```
hdpvalue_singlesplit <- function(x,y){

  # splitting
  n <- length(y)
  in_sample <- sample(seq(n),size=ceiling(n/2))
  x_in <- x[in_sample,]
  y_in <- y[in_sample]
  x_out <- x[-in_sample,]
  y_out <- y[-in_sample]

  # screening
  fit.cv <- cv.glmnet(scale(x_in), y_in,standardize=FALSE)
  bhat <- as.matrix(coef(fit.cv,s="lambda.1se"))[-1,]
  bhato <- sort(abs(bhat[bhat!=0]),decreasing=TRUE)
  if(length(bhato)<length(y_out)-1){
    hatact <- names(bhato)
  }else{
    hatact <- names(bhato)[1:(length(y_out)-2)]
  }

  # testing
  t.dat <- data.frame(cbind(x_out[,hatact[-1]]))
  t.dat$y <- y_out
  fit.lm <- lm(y~.,data=t.dat)
  fit.glht <- glht(fit.lm)
  pvals <- summary(glht(fit.lm), test = adjusted("bonferroni"))$test$pvalues[-1]


  # output
  pvalues <- rep(1,length=ncol(x))
  names(pvalues) <- colnames(x)
  pvalues[names(pvals)] <- pvals

  return(pvalues)
}

pvals_ms <- replicate(100,hdpvalue_singlesplit(x=x,y=y))
```

Histogram of the p-values of all splits for the gene *YXLD_at*.

```
hist(pvals_ms["YXLD_at",],breaks=20)
```



Aggregate the p-values over the different splits.

```
pval.agg <- sort(apply(pvals_ms,1,quantile,probs=0.1),decreasing=FALSE)
head(pval.agg)
```

```
##    YXLD_at    YCKE_at    YOAB_at    XHLA_at
## 0.02471826 0.25095686 0.36085847 0.69123631
##    LYSC_at    AADK_at
## 0.98907351 1.00000000
```

The multi-splitting approach is implemented in the function `hdi` function. (Note that the `hdi` function uses a clever way to aggregate the p-values.)

```
library(hdi)
x <- riboflavin[,-1]
y <- riboflavin[,1]
## Multi-split p-values
set.seed(12)
fit.multi <- hdi(x, y, method = "multi-split", B = 100)
head(sort(fit.multi$pval.corr,decreasing = FALSE))
```

Logistic regression and splines

We explore logistic regression based on the South African heart disease data. Proceed as follows:

1. Fit a univariate logistic regression model with `age` as the covariate. Calculate the odds-ratio and elaborate on the interpretation.

2. Predict the probability of heart disease at age 65.

3. Fit a logistic regression model including all covariates. Run stepwise backward selection. Which variables are excluded? What is the AIC value of the final model?
4. Fit a logistic regression model using four natural cubic spline bases for each term in the model. Run backward selection and summarise the final model. Plot the natural cubic spline functions for the age term (use `termplot`). What does the plot tell you?

The solution to this exercise.

We load the data and fit a logistic regression with age as the covariate.

```
sahd <- readRDS(file="data/sahd.rds")
fit <- glm(chd~age, data=sahd, family=binomial )
summary(fit)
```

```
##
## Call:
## glm(formula = chd ~ age, family = binomial, data = sahd)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4321  -0.9215  -0.5392   1.0952   2.2433
##
## Coefficients:
##              Estimate Std. Error z value
## (Intercept) -3.521710   0.416031  -8.465
## age          0.064108   0.008532   7.513
##             Pr(>|z|)
## (Intercept)  < 2e-16 ***
## age         5.76e-14 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 596.11  on 461  degrees of freedom
## Residual deviance: 525.56  on 460  degrees of freedom
## AIC: 529.56
##
## Number of Fisher Scoring iterations: 4
```

The odds ratio for age is given next.

```
exp(coef(fit)[2])
```

```
##      age
## 1.066208
```

This means that an increase of 1 year in age leads to a 6.6% increase in the odds of having a heart disease.

The estimated probability of having a heart disease at age 65 can be calculated using the `predict` function.

```
predict(fit,newdata=data.frame(age=65),type="response")
```

```
##         1
## 0.6559532
```

Alternatively, we can use the inverse logit formula.

```
lp <- coef(fit)[1]+coef(fit)[2]*65
exp(lp)/(exp(lp)+1)
```

```
## (Intercept)
##   0.6559532
```

We fit a logistic regression model including all covariates. Then we perform stepwise backward selection using `stepAIC`.

```
fit.full <- glm(chd~sbp+tobacco+ldl+famhist+obesity+alcohol+age,
                    data=sahd,
                    family="binomial")
fit.bw <- stepAIC(fit.full,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 variables alcohol, sbp and obesity are excluded from the model. The AIC values are provided in the table above. We can also re-calculate the AIC for the final model.

```
AIC(fit.bw)
```

```
## [1] 495.4439
```

We fit a logistic regression model using natural splines.

```
# Computes the logistic regression model using natural splines (note famhist is included as a factor):
form <-  "chd ~ ns(sbp,df=4) + ns(tobacco,df=4) + ns(ldl,df=4) + famhist + ns(obesity,df=4)+ ns(alcohol
form <-  formula(form)
fit <-  glm( form, data=sahd, family=binomial )

# stepwise backward selection
fit.bw <- stepAIC(fit,direction="backward",trace = 0)
kable(as.data.frame(fit.bw$anova),digits=3,booktabs=TRUE)
```

Step	Df	Deviance	Resid. Df	Resid. Dev	AIC
	NA	NA	436	457.632	509.632
- ns(alcohol, df = 4)	4	0.456	440	458.088	502.088

The summary of the final model is provided next.

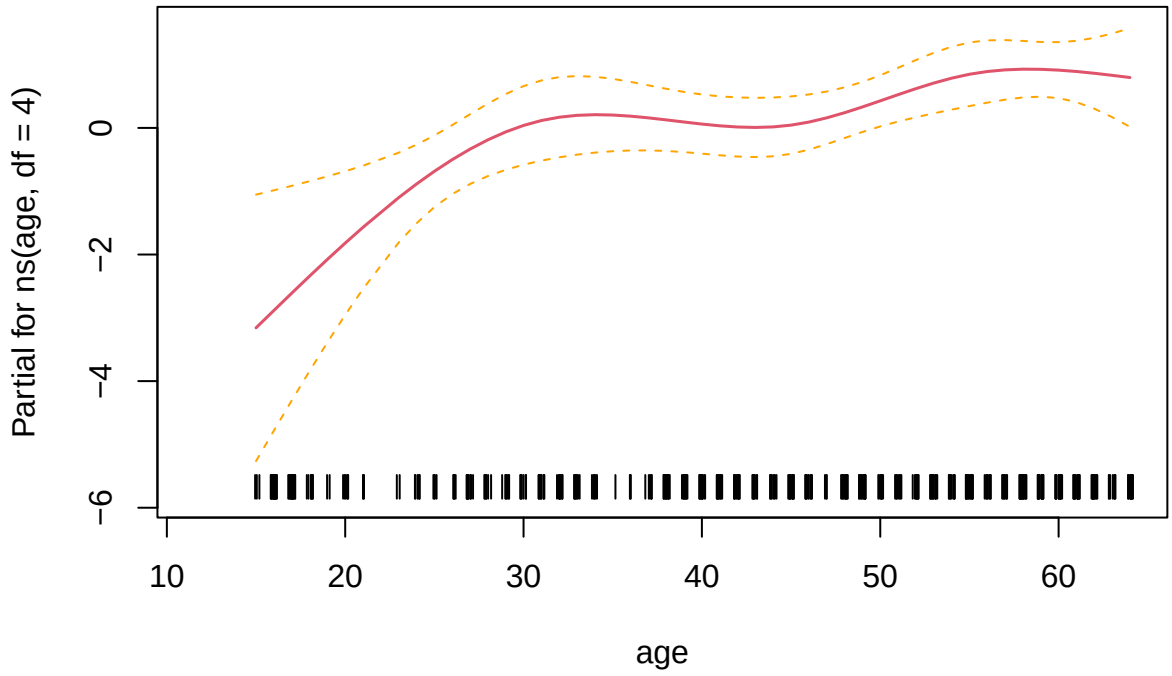
```
kable(as.data.frame(drop1(fit.bw, test="Chisq" )),digits=2,booktabs=TRUE)
```

	Df	Deviance	AIC	LRT	Pr(>Chi)
	NA	458.09	502.09	NA	NA
ns(sbp, df = 4)	4	467.16	503.16	9.08	0.06

	Df	Deviance	AIC	LRT	Pr(>Chi)
ns(tobacco, df = 4)	4	470.48	506.48	12.39	0.01
ns(ldl, df = 4)	4	472.39	508.39	14.31	0.01
famhist	1	479.44	521.44	21.36	0.00
ns(obesity, df = 4)	4	466.24	502.24	8.15	0.09
ns(age, df = 4)	4	481.86	517.86	23.77	0.00

We can plot the natural spline function for the first term as follows.

```
termplot(fit.bw,se=TRUE,rug=TRUE,term=6)
```



The plot shows how the log-odds change with age (keeping the other variables fixed). We observe a slight deviation from linearity, i.e. the log-odds increase more strongly for age <35 than for age >35.

Decision trees, Random Forest and AdaBoost

In this exercise we explore decision trees based on the South African heart disease data.

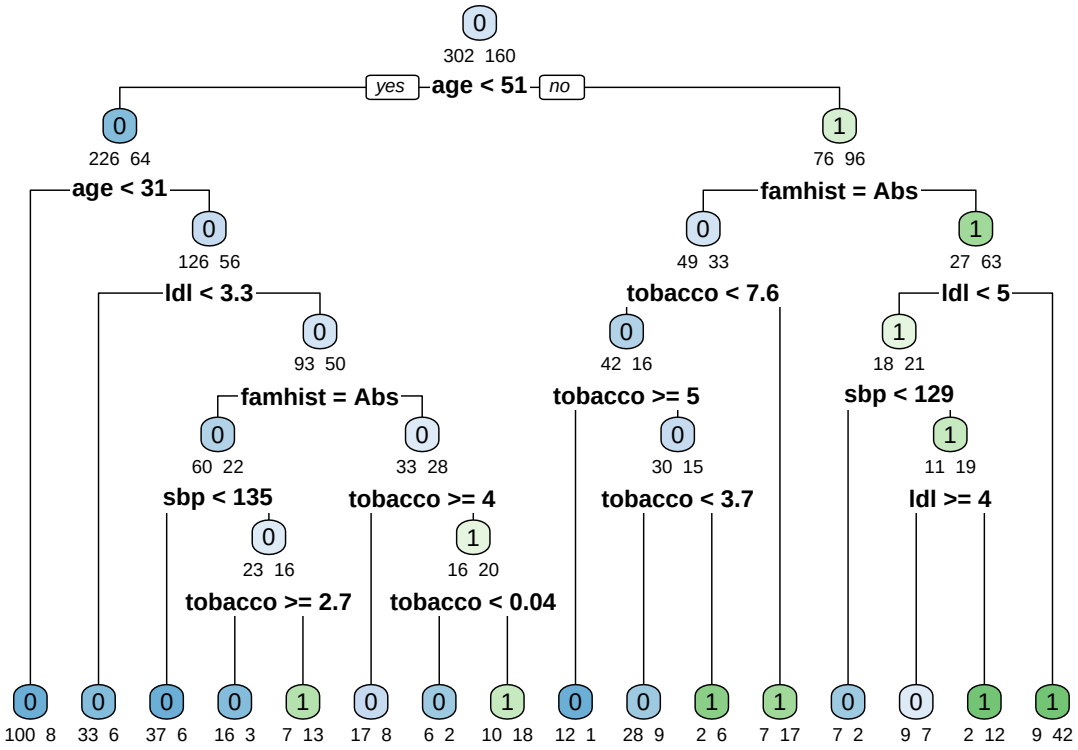
1. Load the South African heart disease data and grow a decision tree using `rpart`. Visualize the tree using `rpart.plot`. How many leave nodes has the tree?
2. Re-grow the tree but now relax the "default" control parameters (choose `rpart.control(cp=0, minsplit=50)`). How many leaves has the tree now?
3. Plot the cross-validation error against the complexity parameter α. What is the tree size of the optimal model?
4. Prune the tree using `prune` and by choosing cp ($=\alpha$) to achieve minimal cross-validation error.
5. Calculate the confusion matrix and the misclassification error.

6. Generate a bootstrap sample. Grow a tree and calculate the out-of-bag error.
7. Fit a random forest using `randomForest`. Plot the fitted object. What is this plot telling us? Calculate the variable importance. Which are the most important variables?
8. Run AdaBoost using `gbm`. What is the prediction for a patient with covariates sbp=100, tobacco=0, ldl=5, famhist="Present", obesity=25, alcohol=10 and age=50. Compute the variable importance.

The solution to this exercise.

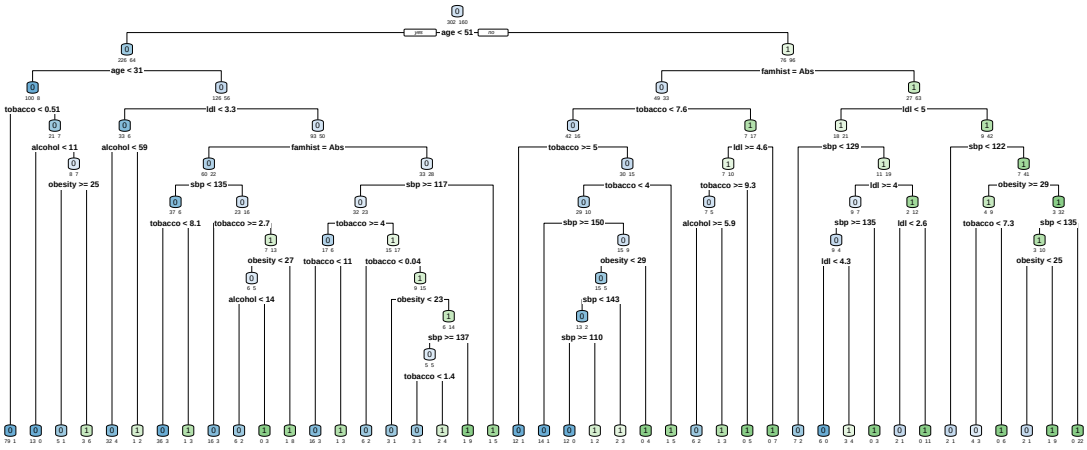
First we read the data and grow the tree.

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

# grow a classification tree
fit.tree <- rpart(chd~.,data=sahd,method="class")
rpart.plot(fit.tree,extra=1,under=TRUE,tweak = 1.2,faclen=3)
```



We re-grow the tree using different control parameters.

```
# controlling the growth of the tree with rpart.control
# cp: improvement in each split needs to be > cp
# minsplit: minimal number of samples in a node inorder to do a split
fit.tree2 <- rpart(chd~.,data=sahd,method="class",
                   control = rpart.control(cp = 0,minsplit=10)

)
rpart.plot(fit.tree2,extra=1,under=TRUE,tweak = 1.2,faclen=3)
```



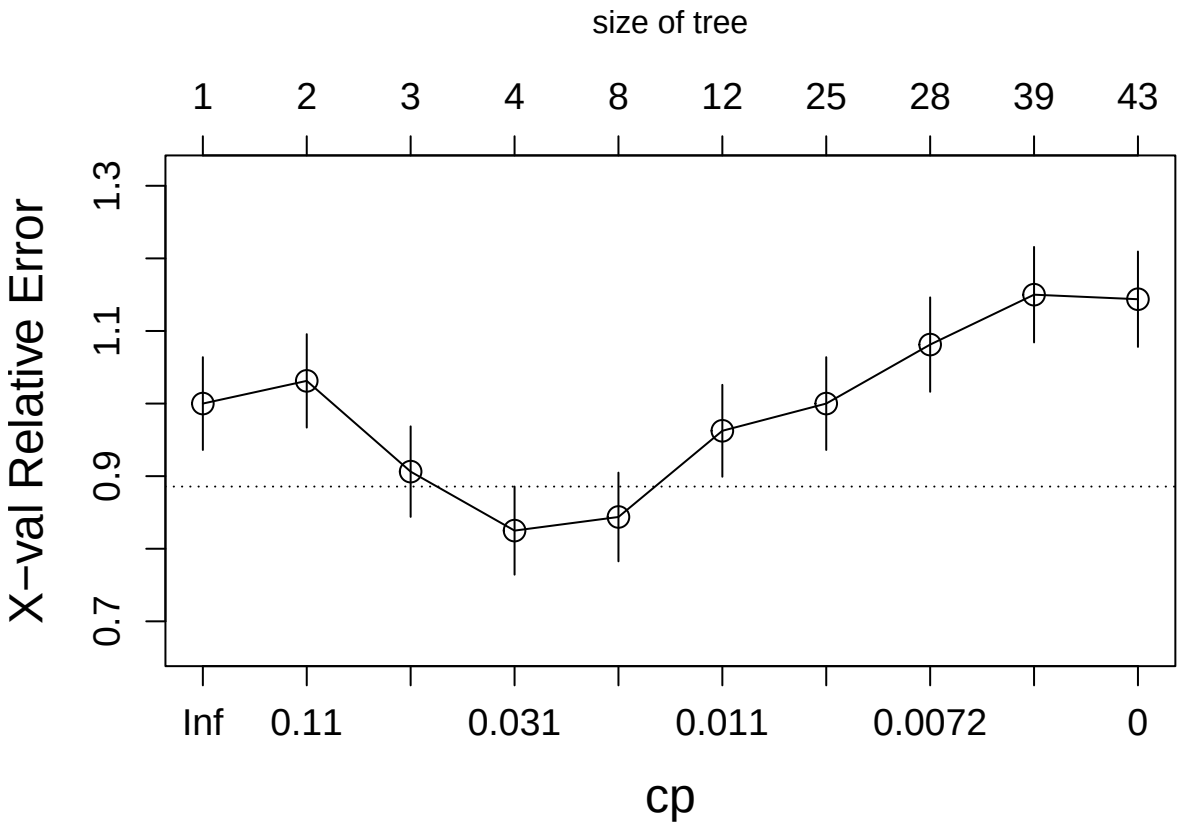
We can get the tree size from the cptable (tree size=number of leaves=number splits+1).

```
fit.tree2$cptable[fit.tree2$cptable[,"CP"]==0,"nsplit"]+1 # number of leaves
```

```
## [1] 43
```

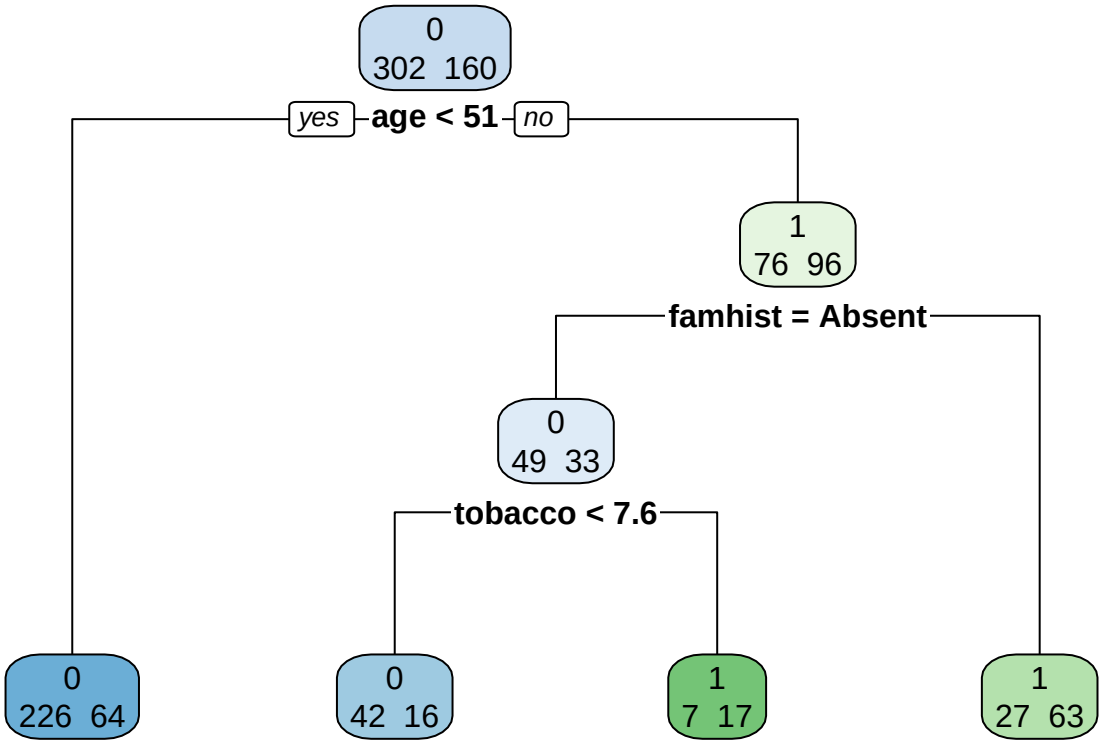
Next, we plot the cross-validation error against the complexity parameter α.

```
plotcp(fit.tree2,cex.lab=1.5,cex.axis=1.2,cex=1.5)
```



We prune the tree and visualize the result.

```
# prune the tree
fit.prune<- prune(fit.tree2,
                  cp=fit.tree2$cptable[which.min(fit.tree$cptable[,"xerror"]),"CP"])
rpart.plot(fit.prune,extra=1)
```



Finally, we compute the confusion matrix and the misclassification error.

```
# confusion matrix of actual and fitted class labels
table(Actual=sahd$chd,Fitted=predict(fit.prune,type="class"))
```

```
##       Fitted
## Actual   0   1
##      0 268  34
##      1  80  80
```

```
# missclassification error
mean(sahd$chd!=predict(fit.prune,type="class"))
```

```
## [1] 0.2467532
```

We sample with replacement (bootstrap sample).

```
set.seed(1)
inthebag <- sample(1:nrow(sahd),size=nrow(sahd),replace=TRUE)
outofbag <- setdiff(1:nrow(sahd),inthebag)
```

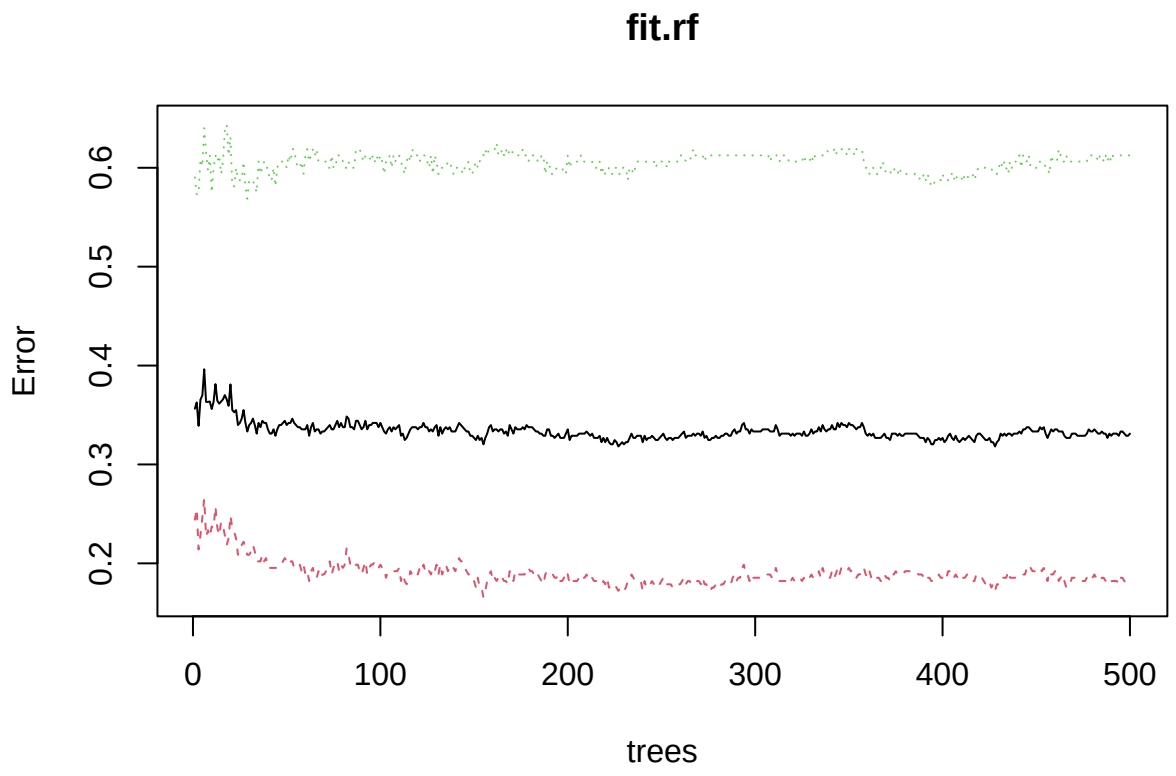
We fit a tree on the in-the-bag samples and calculate the misclassification error on the out-of-bag samples.

```
fit.in <- rpart(chd~.,data=sahd[inthebag,],method="class")
pred.oob <- predict(fit.in,newdata=sahd[outofbag,],type="class")
mean(sahd$chd[outofbag]!=pred.oob)
```

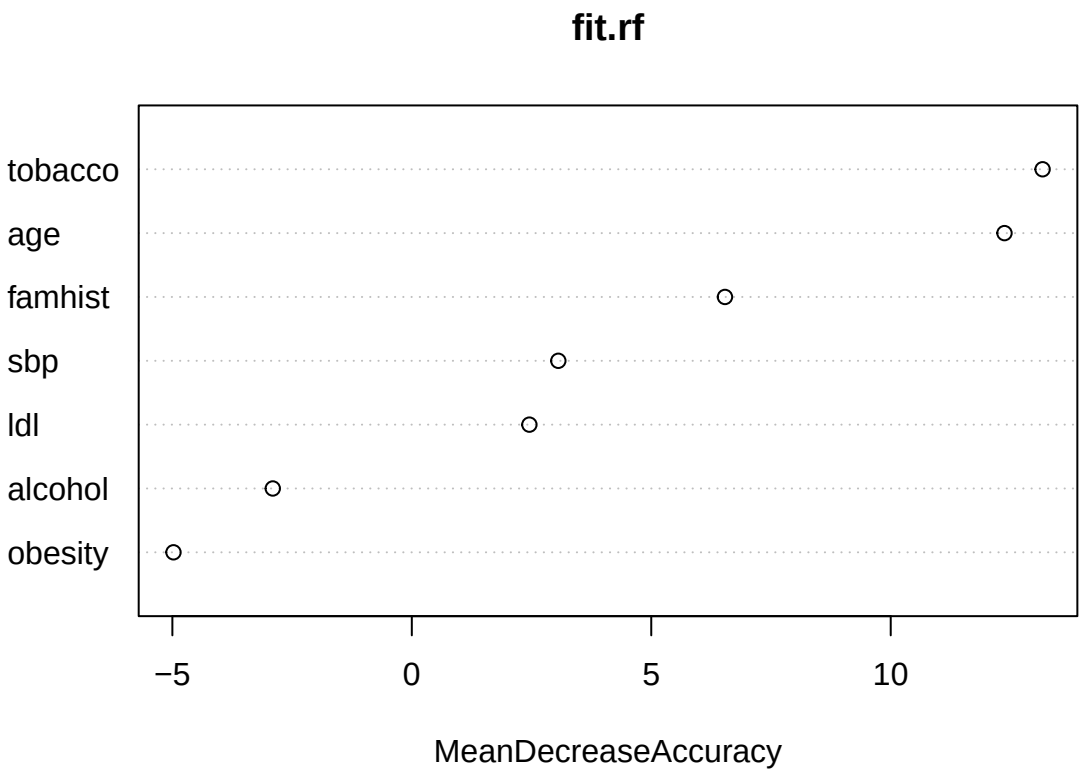
```
## [1] 0.3559322
```

We fit the random forest, plot the error as a function of the number of trees and plot the variable importance.

```
library(randomForest)
sahd$chd <- factor(sahd$chd)
fit.rf <- randomForest(chd~.,data=sahd,importance=TRUE)
plot(fit.rf)
```

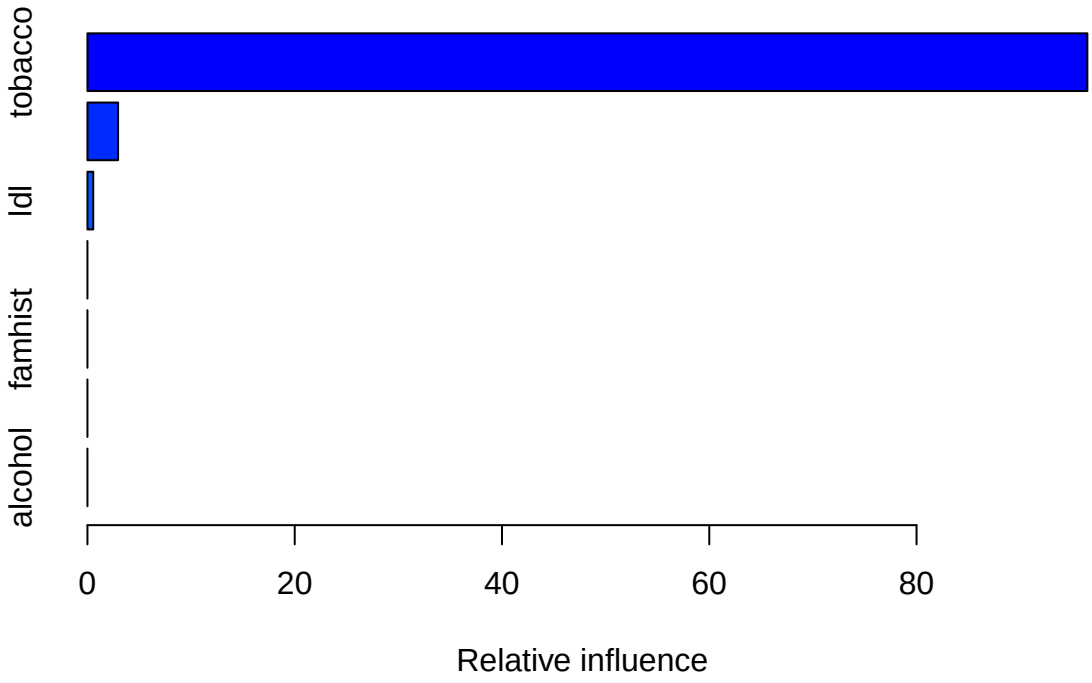



```
varImpPlot(fit.rf,type=1)
```



We run AdaBoost using `gbm` and by specifying `distribution = "adaboost"`. The `summary` provides a measure of variable importance. Prediction can be made using `predict`.

```
fit.boost <-gbm(chd~.,data=sahd,distribution = "adaboost") # note: for adaboost the outcome must be num
summary(fit.boost)
```




```
##             var    rel.inf
## tobacco tobacco 96.4821033
## age         age  2.9567803
## ldl         ldl  0.5611164
## sbp         sbp  0.0000000
## famhist famhist  0.0000000
## obesity obesity  0.0000000
## alcohol alcohol  0.0000000
```

```
newd <- data.frame(sbp=100,tobacco=0,ldl=5,famhist=factor("Present"),obesity=25,alcohol=10,age=50)
predict(fit.boost,
        newdata=newd,
        type="response" )
```

```
## Using 100 trees...
```

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

Phoneme Recognition

In this exercise we investigate prediction of phonemes based on digitized speech data.

1. Read the data set, subset the phonemes “aa” and “ao” and create training and test data.

```
dat <- readRDS(file="data/phoneme.rds")
dat2 <- dat[dat$g%in%c("aa","ao"),]

dtrain <- dat2[grepl("^train",dat2$speaker),-c(1,259)]
```

```
xtrain <- as.matrix(dtrain[,-257])
ytrain <- ifelse(dtrain$g=="ao",1,0)
dtest <- dat2[grepl("^test",dat2$speaker),-c(1,259)]
xtest <- as.matrix(dtest[,-257])
ytest <- ifelse(dtest$g=="ao",1,0)

dtrain$y <- ytrain
dtest$y <- ytest
dtrain <- dtrain[,-257]
dtest <- dtest[,-257]
```

2. Plot the log-periodogram as a function of frequency for 5 examples each of the phonemes "aa" and "ao".
3. Fit a logistic regression model and evaluate the training and test misclassification errors.
4. Run Lasso regression and evaluate the training and test misclassification errors.
5. In the previous approaches we assumed logit-link

$$\text{logit}(x;\beta) = \sum_{j=1}^{256} X_j \beta_j.$$

   Next we assume that the coefficients are a smooth function of the frequency $\beta(f)$, i.e.

$$\beta(f) = \sum_{m=1}^{\nu} h_m(f)\theta_m,$$

   where h_m are B-spline basis functions for a natural cubic spline with $\nu = 12$ degrees of freedom (defined on the set of frequencies). Consider filtered inputs $x^* = \mathbf{H}^T x$ and fit θ by logistic regression on the x^*. Evaluate the training and test misclassification errors.
6. Plot the coefficients of the different models.

The solution to this exercise.

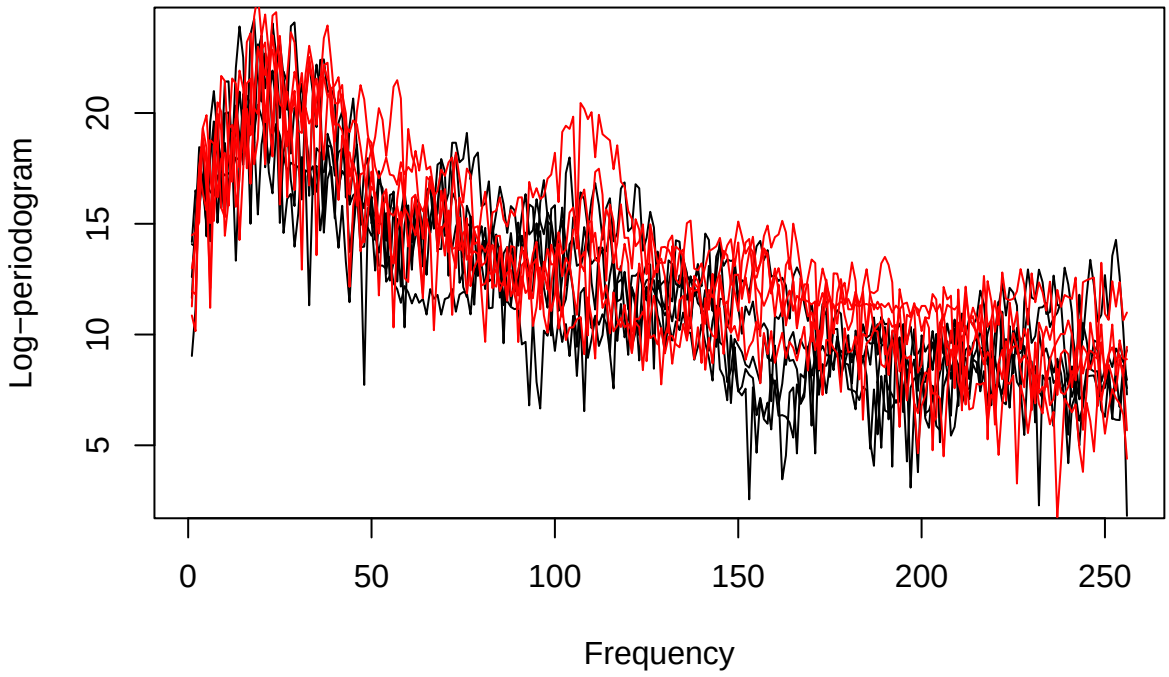
We prepare the data set.

```
library(splines)
dat <- readRDS(file="data/phoneme.rds")
dat2 <- dat[dat$g%in%c("aa","ao"),]

dtrain <- dat2[grepl("^train",dat2$speaker),-c(1,259)]
xtrain <- as.matrix(dtrain[,-257])
ytrain <- ifelse(dtrain$g=="ao",1,0)
dtest <- dat2[grepl("^test",dat2$speaker),-c(1,259)]
xtest <- as.matrix(dtest[,-257])
ytest <- ifelse(dtest$g=="ao",1,0)

dtrain$y <- ytrain
dtest$y <- ytest
dtrain <- dtrain[,-257]
dtest <- dtest[,-257]
```

We plot the log-periodograms.

```
id.ao <- sample(which(ytrain==1),5)
id.aa <- sample(which(ytrain==0),5)
plot(xtrain[id.ao[1],],type="l",
```

```
     xlab="Frequency",ylab="Log-periodogram")
for(i in 2:5){
  lines(xtrain[id.ao[i],])
}
for(i in 1:5){
  lines(xtrain[id.aa[i],],col="red")
}
```



We run logistic regression and calculate the train and test errors.

```
# logistic regression
fit <- glm(y~.,data=dtrain,family=binomial)
coef.glm <-  coefficients(fit)
pred_train <- as.numeric((predict(fit,type="response")>0.5))
pred_test <- as.numeric((predict(fit,type="response",newdata=dtest)>0.5))
mean(pred_train!=ytrain)
```

```
## [1] 0.09311424
```

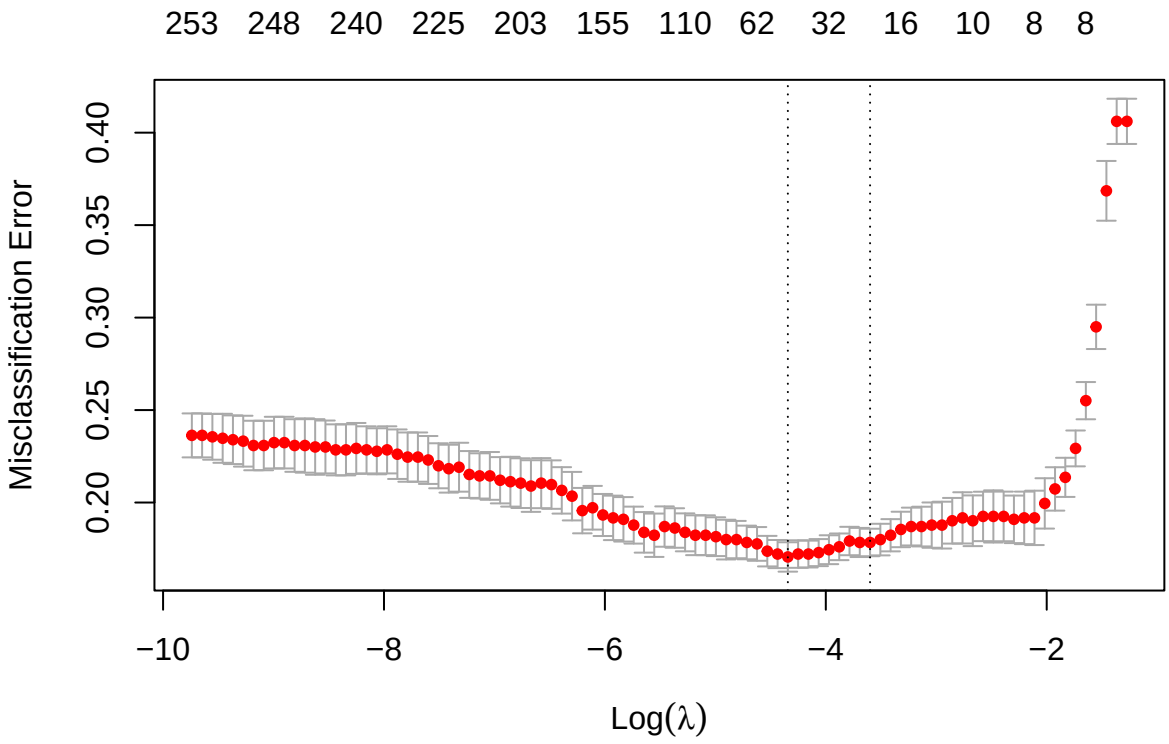
```
mean(pred_test!=ytest)
```

```
## [1] 0.2437358
```

We run Lasso regression and calculate the train and test errors.

```
# lasso regression
fit.glmnet <-glmnet(xtrain,ytrain,family = "binomial",alpha=1)
cv.glmnet <- cv.glmnet(xtrain,ytrain,family = "binomial",type.measure = "class",
                       alpha = 1,nfolds = 10)
```

```
coef.lasso <- as.numeric(coefficients(fit.glmnet,s = cv.glmnet$lambda.1se))[-1]
plot(cv.glmnet)
```




```
pred_train <- c(predict(fit.glmnet,xtrain,s = cv.glmnet$lambda.1se,type = "class"))
pred_test <- c(predict(fit.glmnet,xtest,s = cv.glmnet$lambda.1se,type = "class"))
mean(pred_train!=ytrain)
```

```
## [1] 0.170579
```

```
mean(pred_test!=ytest)
```

```
## [1] 0.2072893
```

We use the natural cubic spline basis with $\nu = 12$ to express the coefficients as a smooth function of the frequencies. We calculate the train and test errors.

```
# coefficient smoothing
hmat <- ns(x=1:256,df=12)
xstar <- xtrain%*%hmat
fit.smooth <- glm(dtrain$y~.,data=data.frame(xstar),family="binomial")
coef.smooth <- as.numeric(hmat%*%coef(fit.smooth)[-1])
pred_train <- as.numeric((predict(fit.smooth,type="response")>0.5))
pred_test <- as.numeric((predict(fit.smooth,type="response",newdata=data.frame(xtest%*%hmat))>0.5))

mean(pred_train!=ytrain)
```

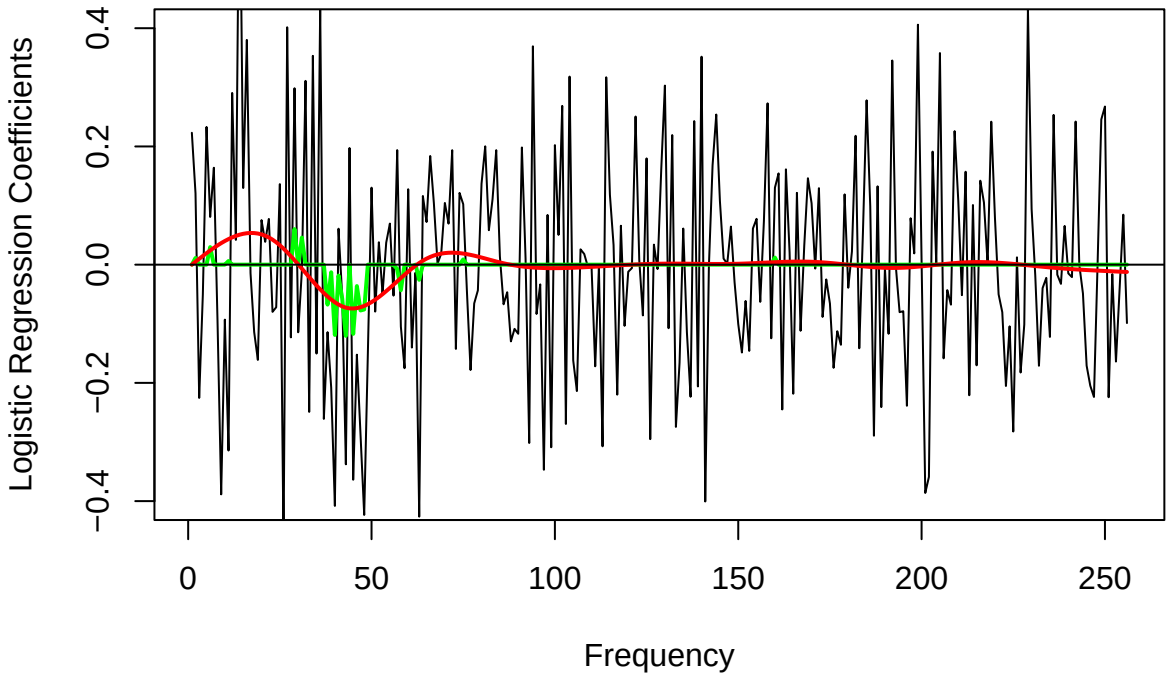
```
## [1] 0.1690141
```

```
mean(pred_test!=ytest)
```

```
## [1] 0.1867882
```

We plot the regression coefficients.

```
plot( coef.glm[-1],
      ylim=c(-0.4,+0.4),
      type="l",
      xlab="Frequency",
      ylab="Logistic Regression Coefficients" )
lines(coef.lasso,col="green",lwd=2)
lines(coef.smooth,col="red",lwd=2)
abline(h=0)
```



Classification and the sonar data set

In this exercise we explore the sonar data set from the *mlbench* package. We use the *caret* package for classification.

1. Load the sonar data set and read the help page.
2. Split the data into a training and test set.
3. Use logistic regression and apply forward selection. Show the estimated coefficients of the final model and calculate the prediction error.
4. Apply elastic-net regularized logistic regression. Show the trace plot and the cross-validation curve. Calculate the prediction error.

5. Run the AdaBoost method. Obtain the variable importance scores and calculate the prediction error. (Hint: use the function `gbm` with argument `distribution="adaboost"`).

The solution to this exercise.

We explore the sonar data set from the *mlbench* package. The goal is to train a model to discriminate between rock "R" and metal "M" based on sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. The $p = 60$ covariates represent energies within particular frequency bands. We first load the data and split into training and test data.

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

set.seed(107)
inTrain <- createDataPartition(
  y = sonar$Class,
  ## the outcome data are needed
  p = .5,
  ## The percentage of data in the
  ## training set
  list = FALSE
)
dtrain <- sonar[ inTrain,]
xtrain <- as.matrix(dtrain[,-61])
ytrain <- dtrain$Class
dtest  <- sonar[-inTrain,]
xtest <- as.matrix(dtest[,-61])
ytest <- dtest$Class
```

In order to compare the performance of the different methods we setup `trainControl`. We use 10-fold cross-validation.

```
tc <- trainControl(method = "cv", number = 10) # 10-fold CV, performance=accuracy
```

We first run logistic regression, once with only an intercept and once including all covariates. We note that `glm` reports convergence problems when using all covariates. This indicates that the model is too complex to fit.

```
fit.lo <-  glm(Class~1,family=binomial,data=dtrain)
fit.full <-  glm(Class~.,family=binomial,data=dtrain)
fit.full.caret <- train(Class ~.,
                        data = dtrain,
                        method = "glm",
                        family = "binomial",
                        trControl = tc)
```

Next we use forward selection using `stepAIC` from the MASS package.

```
up <- paste("~", paste(colnames(dtrain), collapse=" + "))
fit.fw <- stepAIC(fit.lo,
                  scope=up,
                  direction = "forward",
                  steps=10,
                  trace = FALSE)
#summary(fit.fw)
kable(broom::tidy(fit.fw),digits=2,booktabs=TRUE)
```

term	estimate	std.error	statistic	p.value
(Intercept)	4.84	1.74	2.78	0.01
V11	-13.08	5.32	-2.46	0.01
V51	-106.83	46.93	-2.28	0.02
V36	12.56	3.55	3.54	0.00
V43	-15.67	4.67	-3.35	0.00
V4	-54.34	18.41	-2.95	0.00
V15	4.74	2.47	1.92	0.05
V3	30.03	14.72	2.04	0.04
V22	-3.45	1.60	-2.16	0.03
V9	-14.82	6.05	-2.45	0.01
V7	23.17	9.66	2.40	0.02

We can repeat the same analysis using the `train` function from the *caret* package. (Note that there is no tuning parameter involved for `glmStepAIC`.)

```
fit.fw.caret <- train(x=xtrain,
                      y=ytrain,
                      method = "glmStepAIC",
                      family = "binomial",
                      direction ="forward",
                      steps=10,
                      trControl = tc,
                      trace = FALSE
)
# accuracy
kable(fit.fw.caret$results,digits=2,booktabs=TRUE)
```

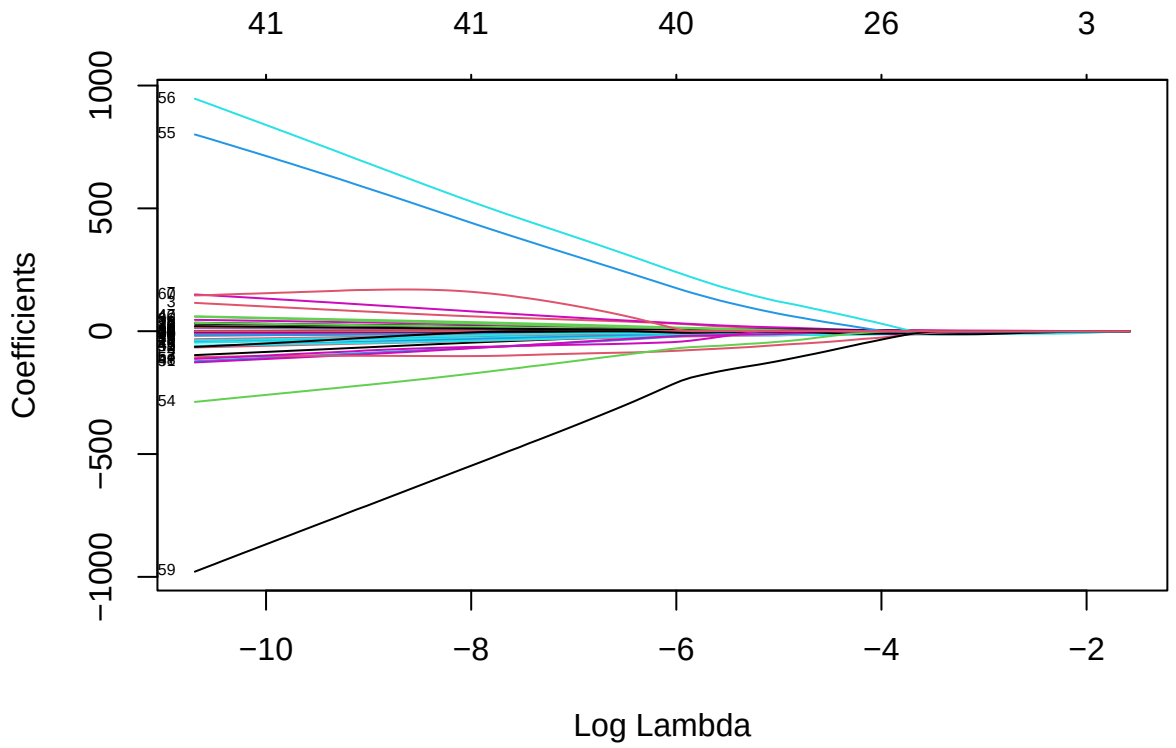
parameter	Accuracy	Kappa	AccuracySD	KappaSD
none	0.68	0.35	0.16	0.32

```
# summary of the model
kable(broom::tidy(fit.fw.caret$finalModel),digits=2,booktabs=TRUE)
```

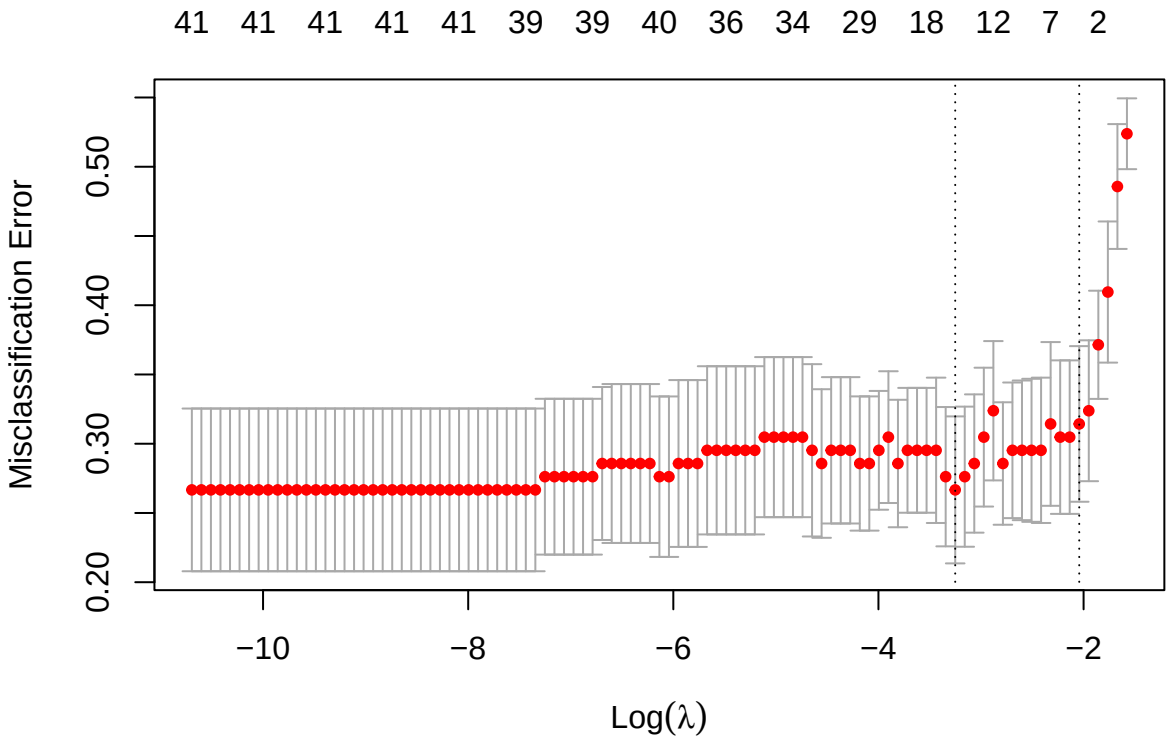
term	estimate	std.error	statistic	p.value
(Intercept)	4.84	1.74	2.78	0.01
V11	-13.08	5.32	-2.46	0.01
V51	-106.83	46.93	-2.28	0.02
V36	12.56	3.55	3.54	0.00
V43	-15.67	4.67	-3.35	0.00
V4	-54.34	18.41	-2.95	0.00
V15	4.74	2.47	1.92	0.05
V3	30.03	14.72	2.04	0.04
V22	-3.45	1.60	-2.16	0.03
V9	-14.82	6.05	-2.45	0.01
V7	23.17	9.66	2.40	0.02

Next we employ L1-penalized logistic regression.

```
fit.l1 <-glmnet(x=xtrain,y=ytrain,alpha=1,family="binomial")
plot(fit.l1,xvar="lambda",label=TRUE)
```

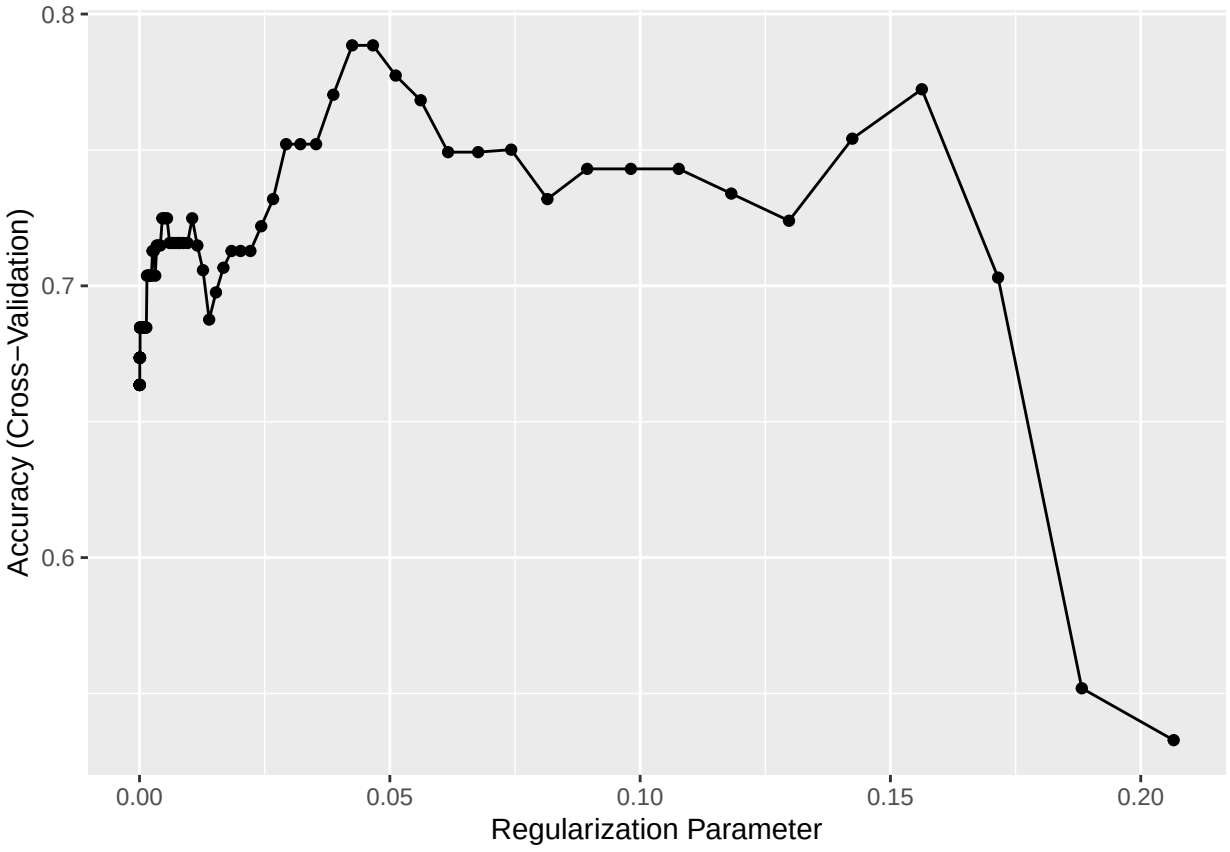



```
cvfit <- cv.glmnet(x=xtrain, y=ytrain,
                   family = "binomial",
                   type.measure = "class",alpha=1)
#cvfit$lambda.min
plot(cvfit)
```



We repeat the same analysis using the caret package.

```
lambda.grid <- fit.l1$lambda
fit.l1.caret<-train(x=xtrain,
                    y=ytrain,
                    method = "glmnet",
                    family="binomial",
                    preProc = c("center","scale"),
                    tuneGrid = expand.grid(alpha = 1,lambda=lambda.grid),
                    trControl = tc
)

ggplot(fit.l1.caret)
```




```
fit.l1.caret
# best lambda
fit.l1.caret$bestTune$lambda
# model coefficients
coef(fit.l1.caret$finalModel,fit.l1.caret$bestTune$lambda)
```

Next we investigate AdaBoost. We can use the `gbm` function.

```
dtrain$Class <-  factor(ifelse(dtrain$Class=="M",0,1))
fit.boost <-gbm(Class~.,data=dtrain)
```

Alternatively, we can use the *caret* package. We first specify the tuning parameter grid.

```
gbmGrid <-  expand.grid(
  n.trees = seq(1,1000,by=20),
  interaction.depth = c(1,3,6),
  shrinkage = 0.1,
  n.minobsinnode = 10
)
```

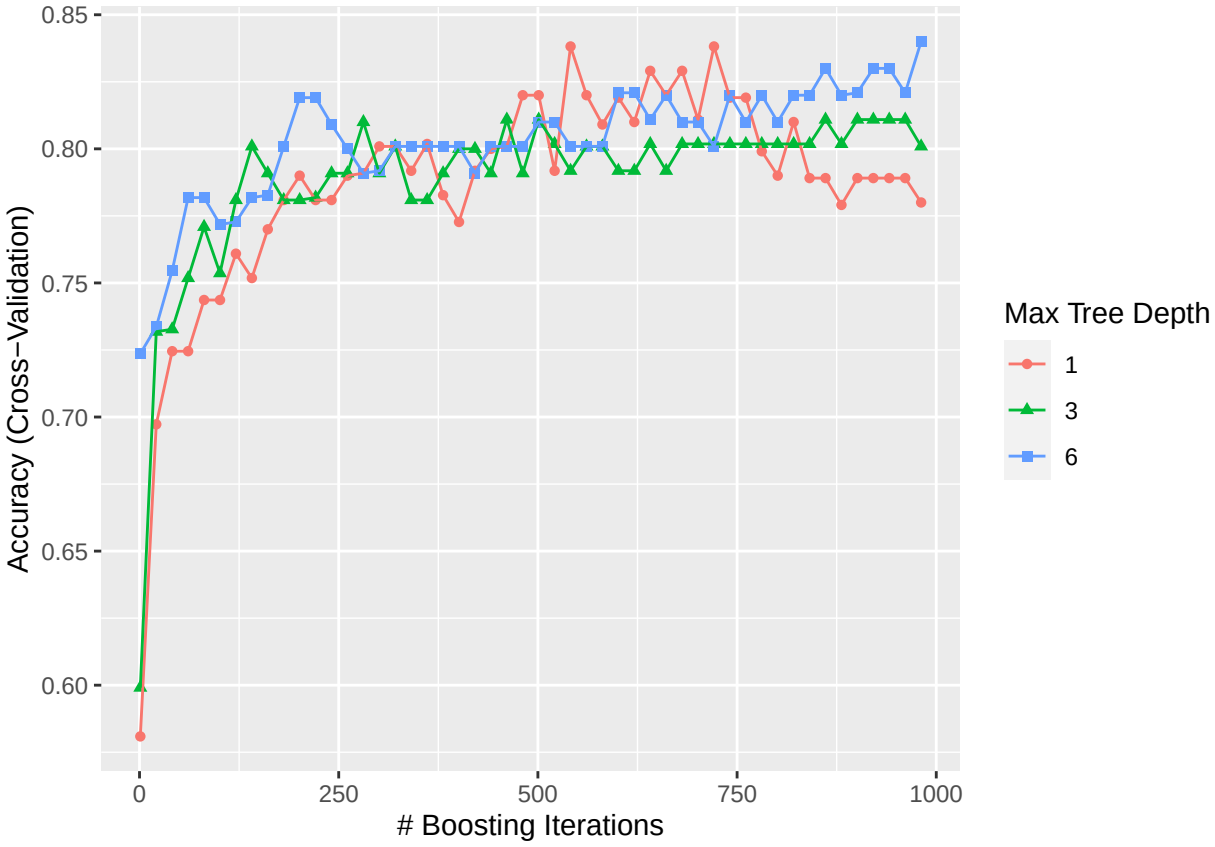
Next we train the model.

```
fit.boost.caret<-train(
  x=xtrain,
  y=ytrain,
  method = "gbm",
  trControl = tc,
  verbose = FALSE,
```

```
  tuneGrid = gbmGrid
)
```

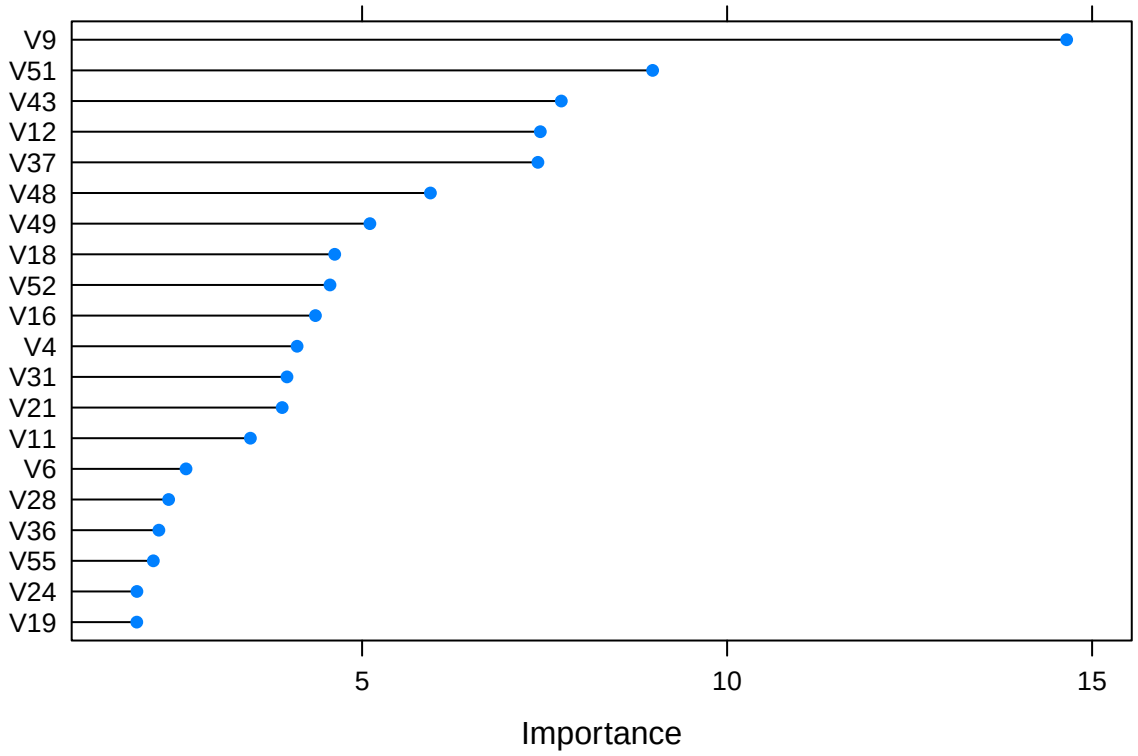
and plot the cross-validation accuracy.

```
ggplot(fit.boost.caret)
```



Finally we obtain the variable importance.

```
gbmImp <- varImp(fit.boost.caret, scale = FALSE)
plot(gbmImp,top=20)
```



Next we make predictions and compare the misclassification errors on the test data.

```
# Make predictions
confusionMatrix(data =fit.boost.caret %>% predict(xtest), ytest)
confusionMatrix(data =fit.l1.caret %>% predict(xtest), ytest)
confusionMatrix(data =fit.fw.caret %>% predict(xtest), ytest)
confusionMatrix(data =fit.full.caret %>% predict(xtest), ytest)
```

With the *caret* package we can also compare the 3 models using the following commands.

```
models <- list(full=fit.full.caret,
               fw=fit.fw.caret,
               l1=fit.l1.caret,
               boost=fit.boost.caret)
summary(resamples(models),metric = "Accuracy")
```

Survival analysis and the Lymphoma data

In this exercise we explore the Lymphoma data set to predict survival based on gene expression data.

1. Load the Lymphoma data and make a histogram of the survival times.
2. Plot the estimated survival curve using `survfit` (Kaplan-Meier method).
3. Fit a Cox regression model with the first three genes as predictors. Use the function `coxph`.
4. Build a predictive model using `glmnet` (data pre-processing: use the top 100 genes and scale the resulting predictor matrix). Which genes are selected? What is the C-index for the optimal tuning parameter?

5. Use the predictive model and classify patients into "good" and "poor" prognostic groups by thresholding the linear predictor at zero. Calculate the Kaplan-Meier curves for the two groups. What is your conclusion? Do you have any concerns?

6. The linear predictor scores computed in 5. are biased as they are evaluated on the same data for which they were computed. We now use a variant of cross-validation, known as *pre-validation*, in order to obtain a fair evaluation of the model. Calculate a pre-validated data set using the code below and calculate the Kaplan-Meier curves for patients with good and poor prognosis.

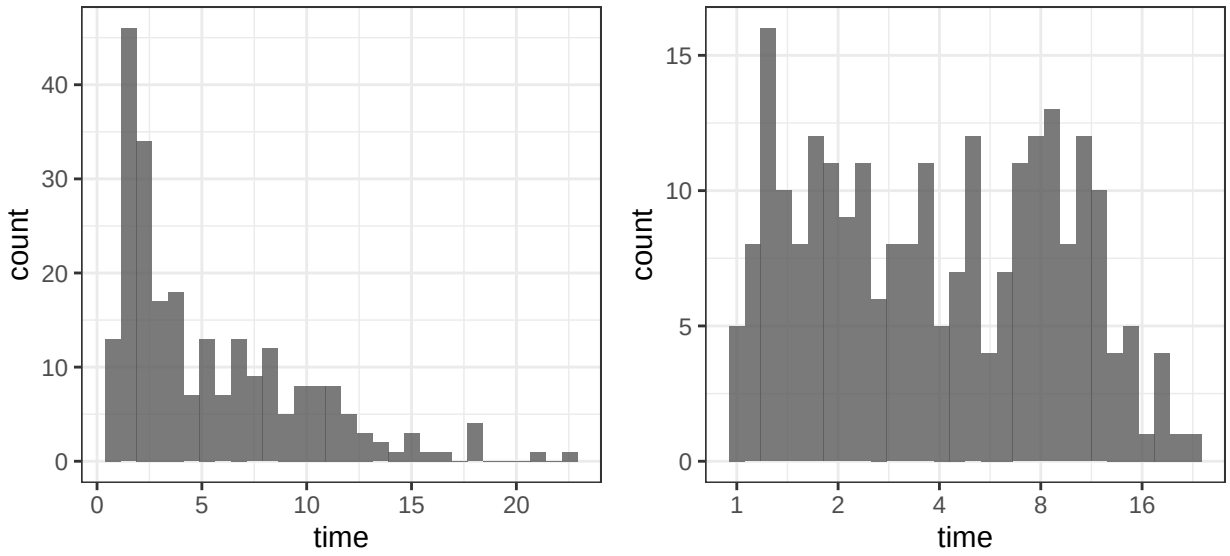
```
# split data into K=5 folds
n.fold <- 5
foldid <- sample(1:n.fold, size = nrow(x), replace = TRUE)

# pre-validation
dat.preval <- data.frame(y)
dat.preval$lp <- NA

for (i in 1:n.fold){

  # train model on samples not in the kth fold
  omitk <- which(foldid==i)
  fitk <- cv.glmnet(x[-omitk,],y.surv[-omitk,],
                    family="cox",
                    type.measure="C",
                    nfolds = 5,
                    alpha=1)

  # calculated linear predictor on samples in the kth fold
  lp <- predict(fitk,
                newx=x[omitk,],
                s=cv.coxnet$lambda.min,
                type="link")
  dat.preval$lp[omitk] <- lp
}
```

Solution to the exercise.

We load the data set.

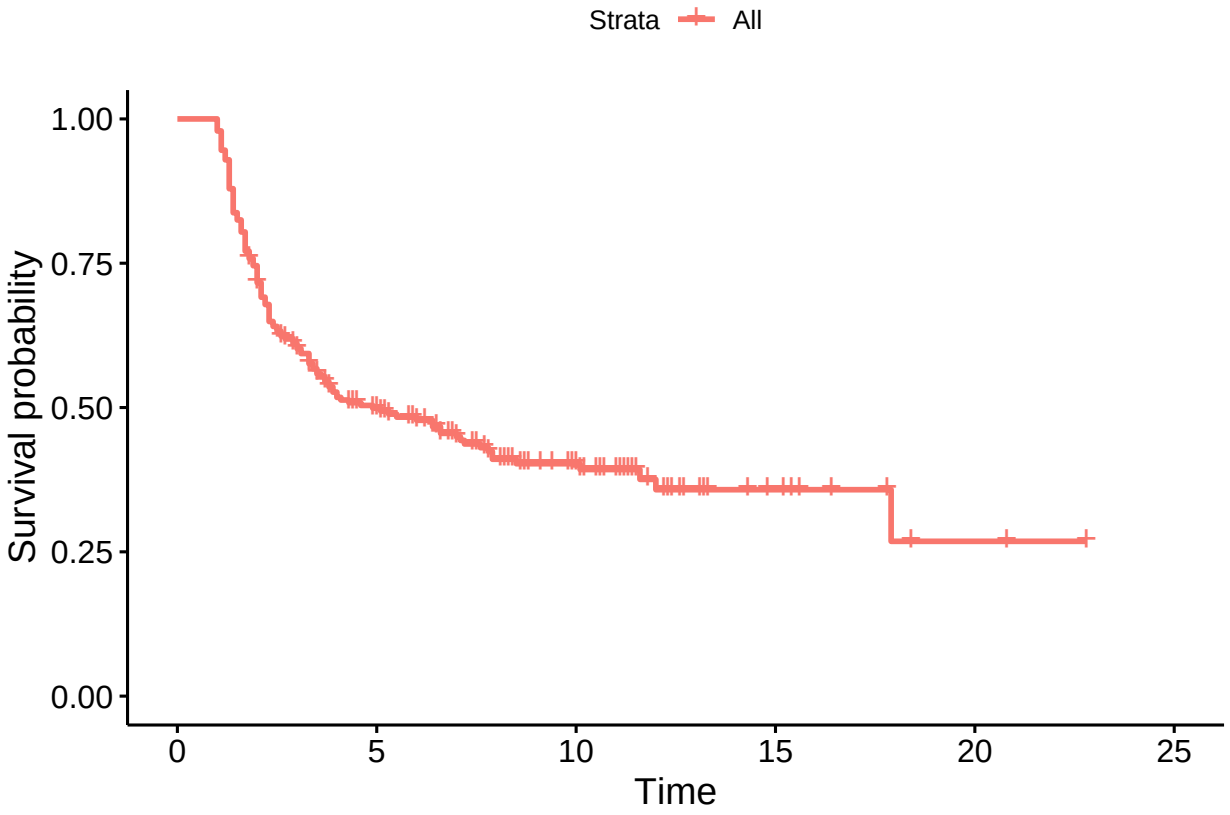
```
# read gene expression matrix
x <- read.table("data/lymphx.txt")%>%
  as.matrix

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

Plot the distribution of the survival times.



Plot of the Kaplan-Meier estimates.

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



Fit a Cox regression model.

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

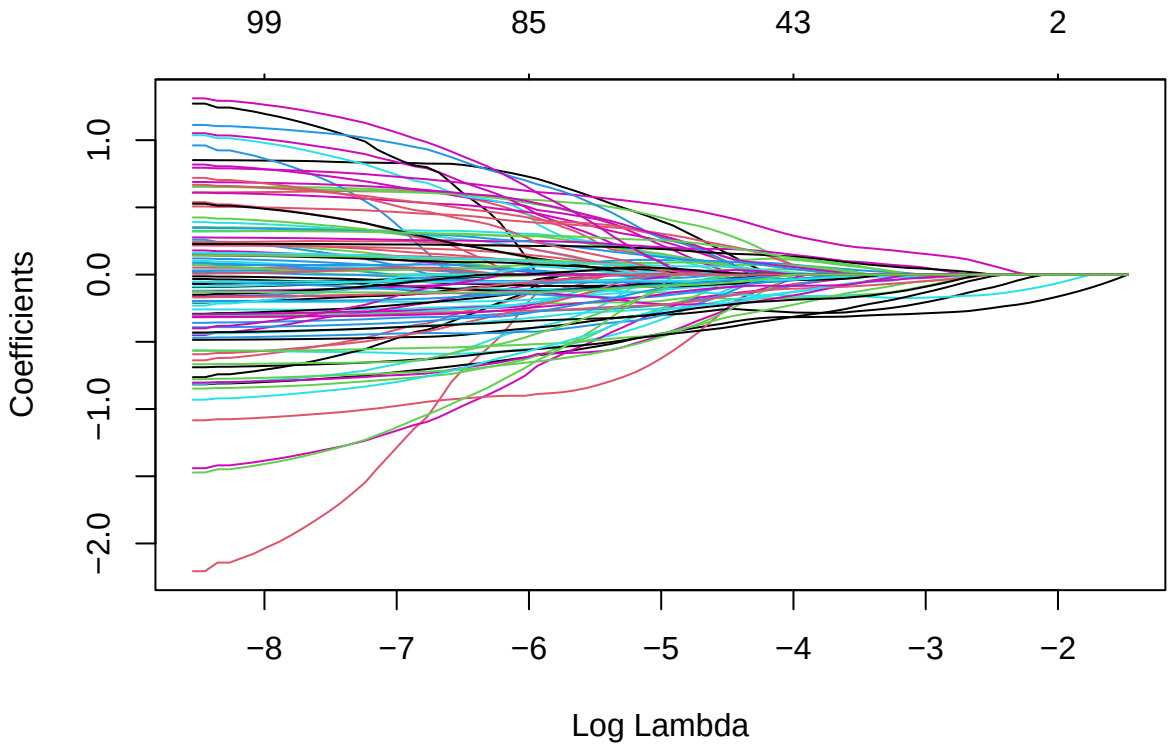
```
## Call:
## coxph(formula = Surv(time, status) ~ ., data = dat)
##
##   n= 240, number of events= 138
##
##       coef exp(coef) se(coef)      z Pr(>|z|)
## V1  0.6382    1.8931   0.4504  1.417    0.156
## V2 -0.5778    0.5611   0.4023 -1.436    0.151
## V3 -0.1508    0.8600   0.3785 -0.398    0.690
##
##    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
```

Build a predictive model using `glmnet`. Data pre-processing.

```
# filter for top genes (highest variance) and scale the input matrix
topvar.genes <- order(apply(x,2,var),decreasing=TRUE)[1:100]
x <- scale(x[,topvar.genes])
```

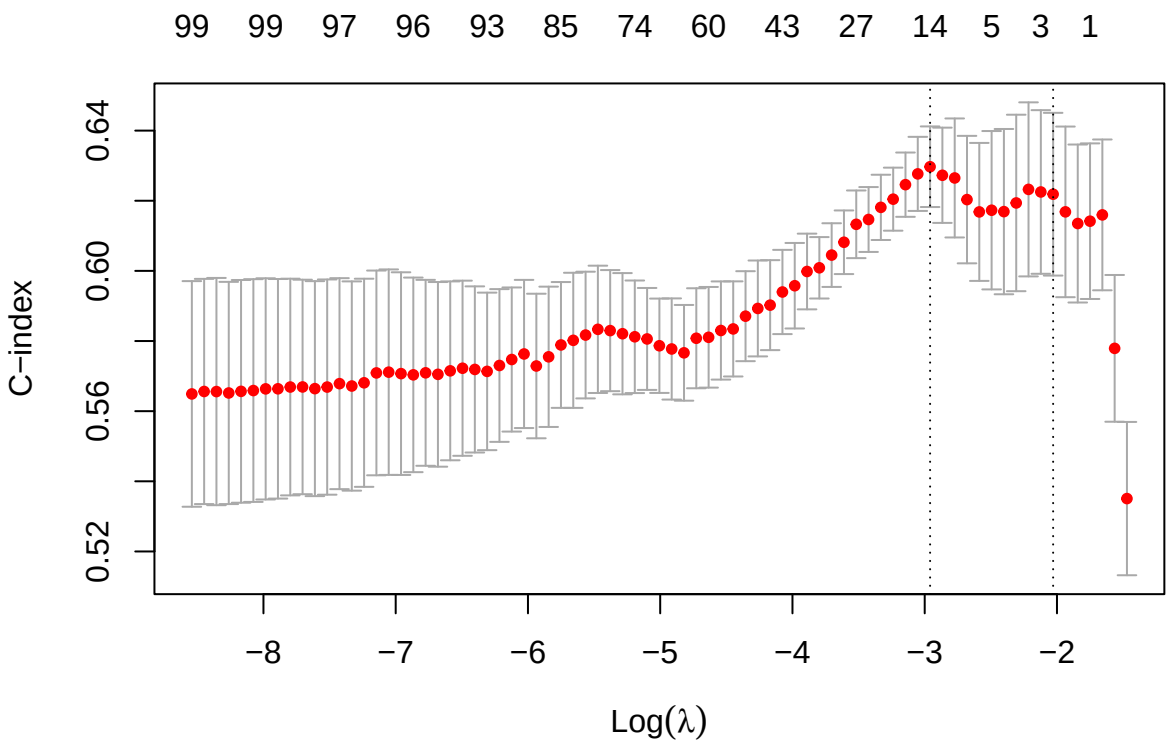
Run `glmnet` using `family="cox"`.

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



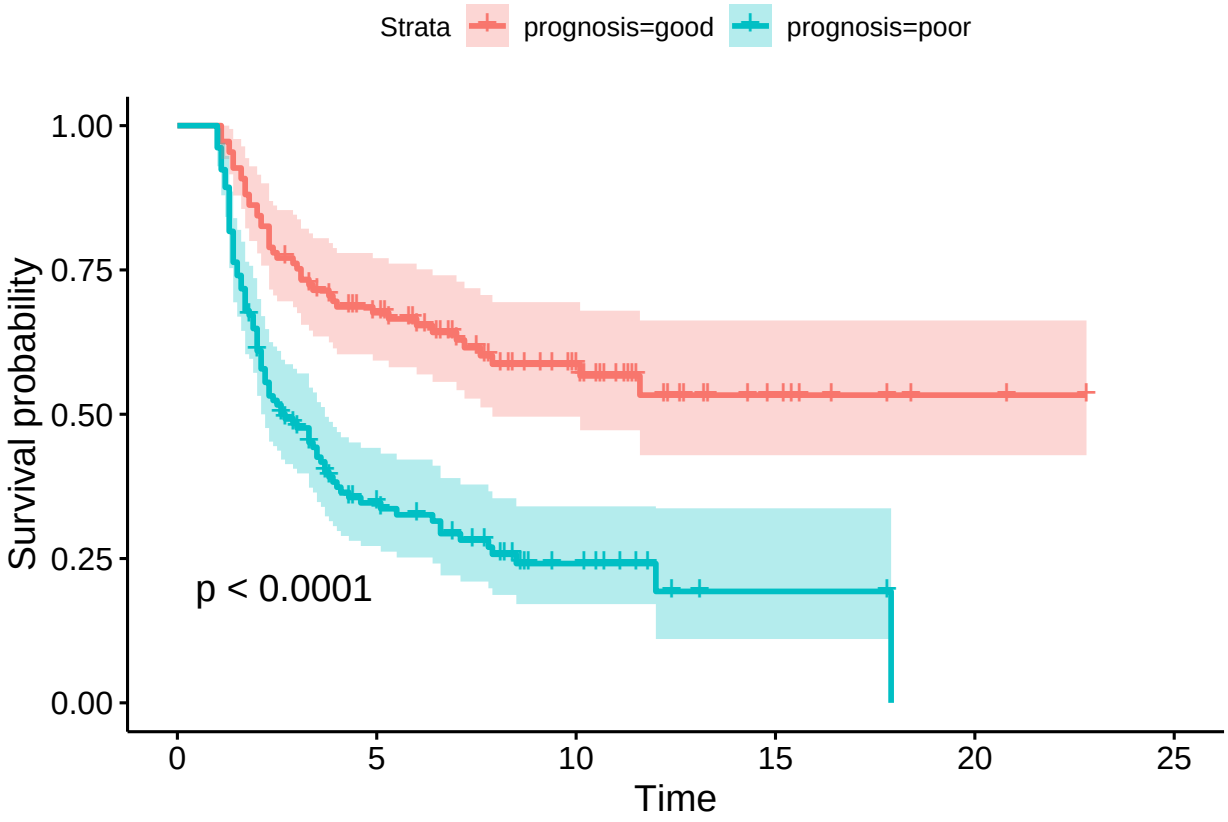
Calculate the cross-validated C-index.

```
cv.coxnet <- cv.glmnet(x,y.surv,
                       family="cox",
                       type.measure="C",
                       nfolds = 5)
plot(cv.coxnet)
```



Classify patients into groups with good and poor prognosis (based on thresholding the linear predictor at zero).

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



The curves are very well separated. However, these linear predictor scores are biased: we are evaluating their performance on the same data for which they were computed.

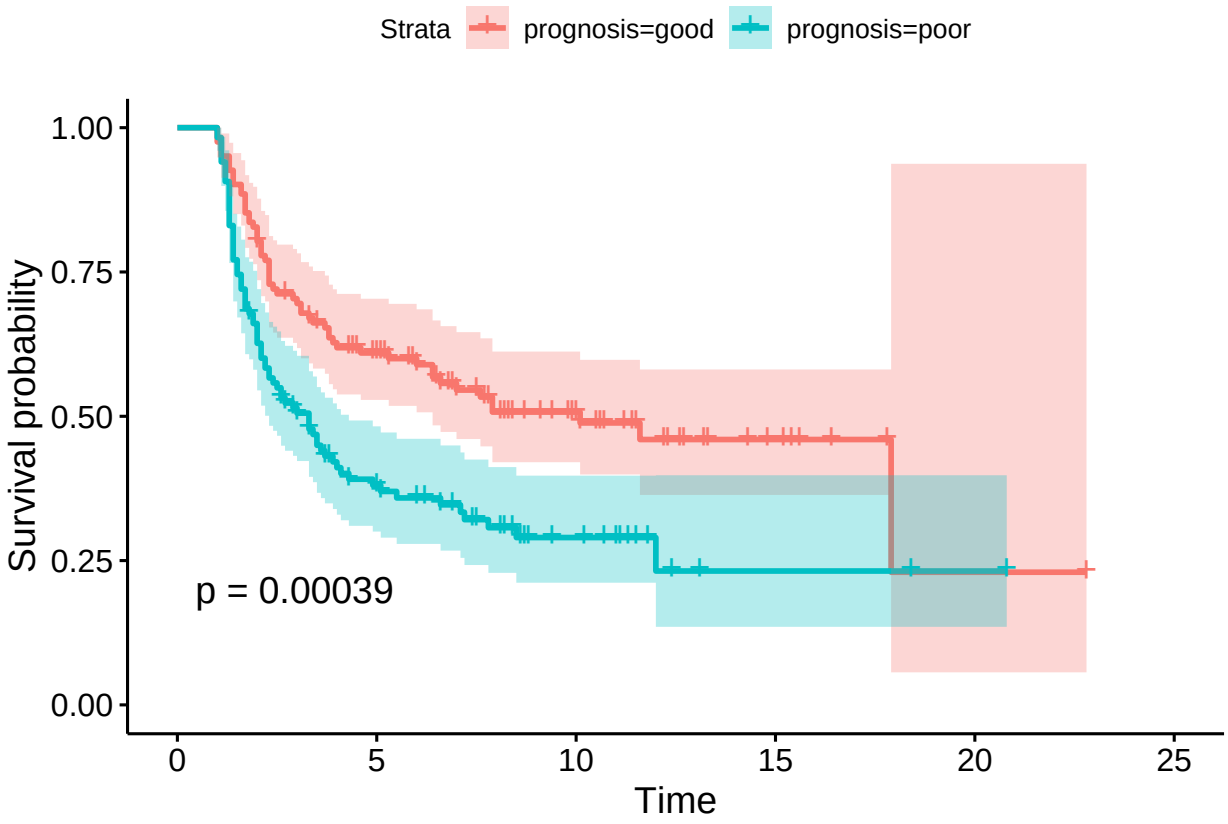
In order to obtain a fair evaluation of the model we calculate a pre-validated data set.

```
set.seed(150381)

# split data into K=5 folds
n.fold <- 5
foldid <- sample(1:n.fold, size = nrow(x), replace = TRUE)

# pre-validation
dat.preval <- data.frame(y)
dat.preval$lp <- NA

for (i in 1:n.fold){

  # train model on samples not in the kth fold
  omitk <- which(foldid==i)
  fitk <- cv.glmnet(x[-omitk,],y.surv[-omitk,],
                    family="cox",
                    type.measure="C",
                    nfolds = 5,
                    alpha=1)

  # calculated linear predictor on samples in the kth fold
  lp <- predict(fitk,
```

```
                newx=x[omitk,],
                s=cv.coxnet$lambda.min,
                type="link")
  dat.preval$lp[omitk] <- lp
}
```

Plot the Kaplan-Meier curves for the good and poor prognosic groups based on the pre-validated data.

```
dat.preval$prognosis <- ifelse(dat.preval$lp>0,"poor","good")
fit.surv <- survfit(Surv(time, status) ~ prognosis,
                    data = dat.preval)
ggsurvplot(fit.surv,conf.int = TRUE,pval=TRUE)
```



Survival analysis based on simulated data (difficult)

In this exercise we simulate high-dimensional survival data and explore regularized cox regression.

1. Simulate high-dimensional survival data from the cox proportional hazards model with $n = 50$ and $p = 100$ covariates. Assume that only the first two covariates are active, i.e. have non-zero regression coefficient. Create training and test data.
2. Based on training data fit a cox regression model including the first 20 covariates. Visualize the hazard-ratios using a forest plot.
3. Based on training data run l1-penalized cox regression using `glmnet`. Show the trace plot and the cross-validated C-index.
4. Based on training data fit a cox regression model including only the active covariates obtained from the

previous glmnet fit.

5. Use the `pec` package to obtain the Brier score on training and test data for the two cox regression models.

Solution to the exercise.

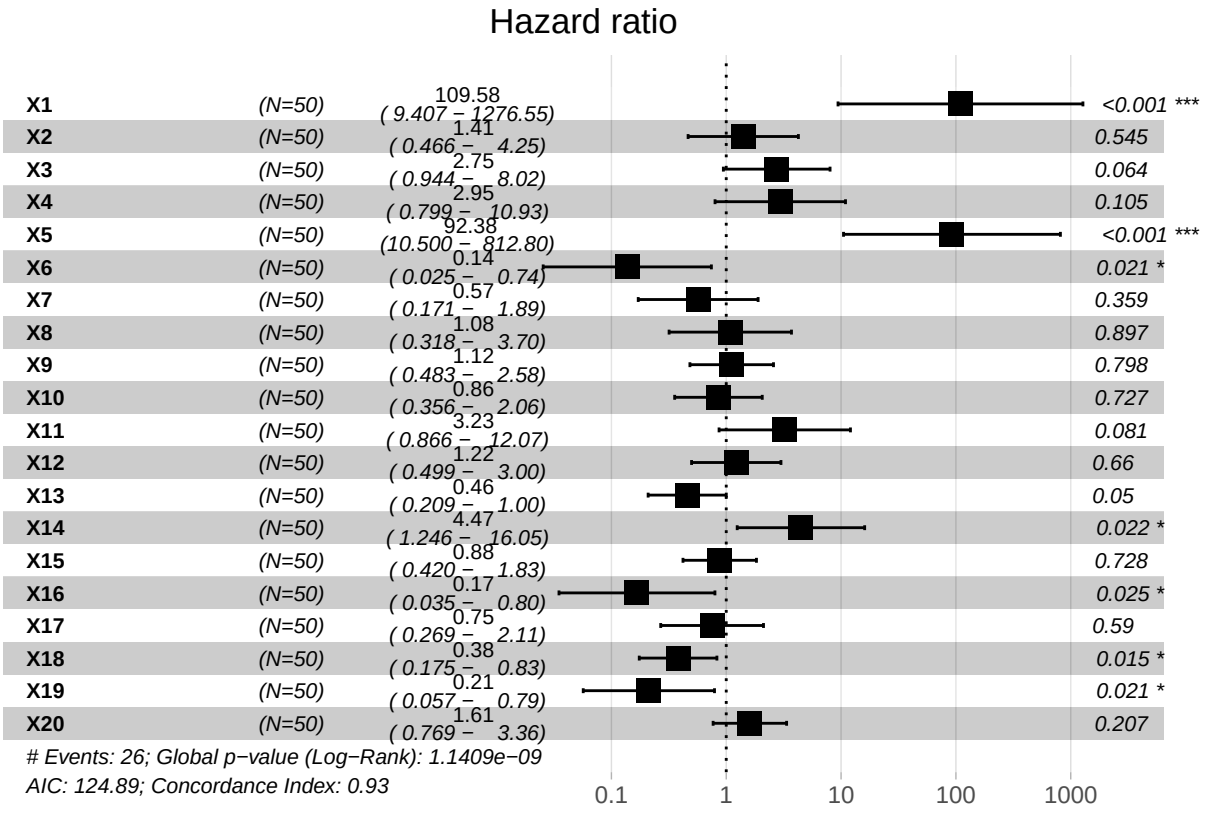
We start by simulating training and test data.

```
## load packages
library(survival)
library(survminer)

## dimensions
p <- 100
n <- 50

## training data

# design matrix
t.dat <- data.frame(matrix(rnorm(n * p), nrow = n))
fm <- as.formula(paste0("~0+",paste(paste0("X",1:p),collapse="+")))
x_design <- model.matrix(fm,data=t.dat)
beta <- c(2,0,0,0,2,rep(0,p-5))
lh <- x_design%*%beta

# survival and censoring time
rate_base <- 1/5 # baseline hazard
rate_cens <- 1/6 # censoring hazard
hazard <- rate_base*exp(lh)
survtime <- rexp(n,rate=hazard)
censtime <- rexp(n,rate=rate_cens)
status <- as.numeric(survtime<=censtime)
time <- survtime*status+censtime*(1-status)
dtrain <- cbind(
  data.frame(survtime,time,status)%>%
    dplyr::mutate(status.cens=1-status),
  t.dat)

## test data

# design matrix
t.dat <- data.frame(matrix(rnorm(n * p), nrow = n))
fm <- as.formula(paste0("~0+",paste(paste0("X",1:p),collapse="+")))
x_design <- model.matrix(fm,data=t.dat)
beta <- c(2,0,0,0,2,rep(0,p-5))
lh <- x_design%*%beta

# survival and censoring time
rate_base <- 1/5 # baseline hazard
rate_cens <- 1/6 # censoring hazard
hazard <- rate_base*exp(lh)
survtime <- rexp(n,rate=hazard)
censtime <- rexp(n,rate=rate_cens)
status <- as.numeric(survtime<=censtime)
time <- survtime*status+censtime*(1-status)
```

```
dtest <- cbind(
  data.frame(survtime,time,status)%>%
    dplyr::mutate(status.cens=1-status),
  t.dat)
```

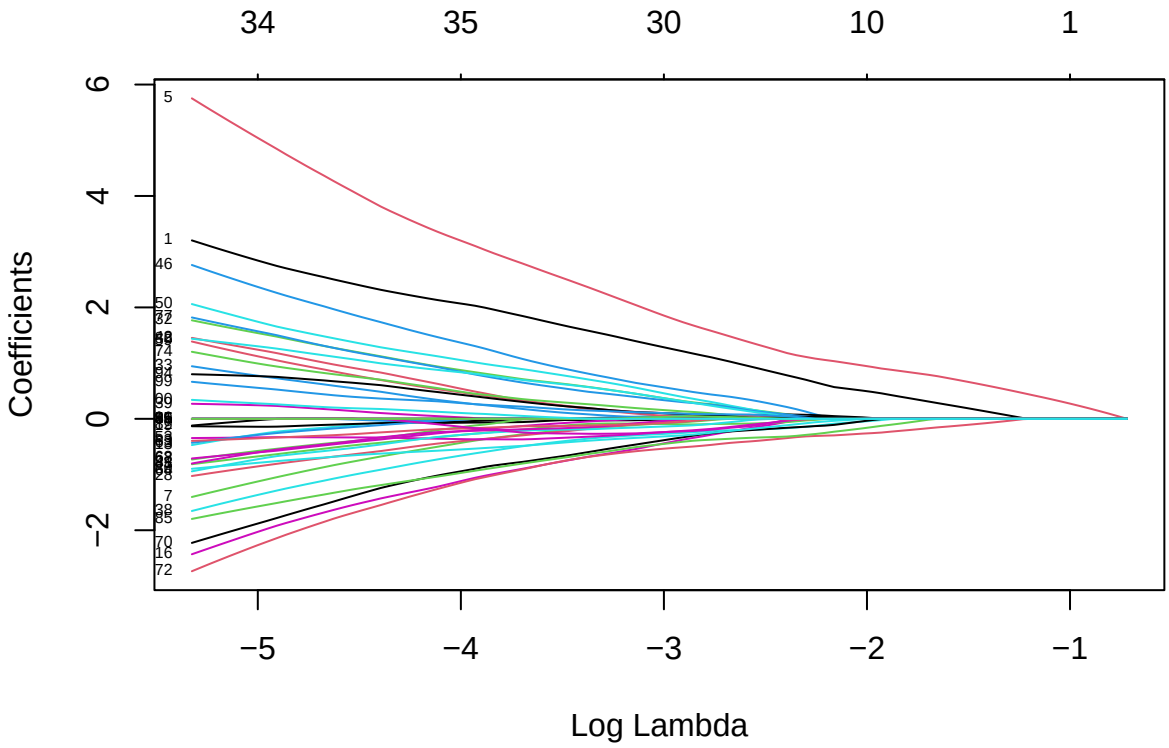
We perform cox regression and create a forest plot.

```
fm1 <- as.formula(paste("Surv(time,status)~",paste(paste0("X",1:20),collapse="+")))
fit1 <- coxph(fm1,data=dtrain,x=TRUE)
survminer::ggforest(fit1)
```



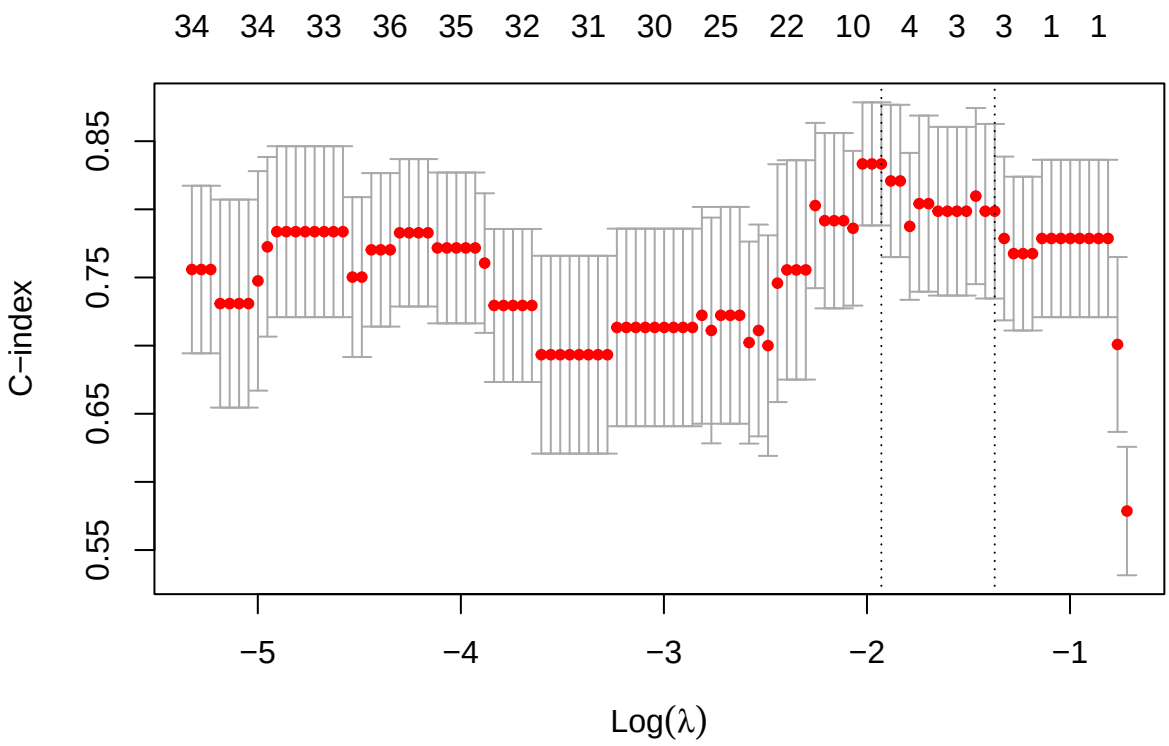
We perform l1-regularized cox regression and generate the trace plot.

```
xtrain <- as.matrix(dtrain[,-c(1:4)])
ytrain <- with(dtrain,Surv(time,status))
fit.coxnet <- glmnet(xtrain, ytrain, family = "cox",alpha=0.95)
plot(fit.coxnet,xvar="lambda",label=TRUE)
```



We perform cross-validation and plot the C-index.

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



We perform cox regression based on only the active covariates.

```
act <- which(as.numeric(coef(fit.coxnet,s=cv.coxnet$lambda.min))!=0)
fm2 <- as.formula(paste("Surv(time,status)~",paste(paste0("X",act),collapse="+")))
fit2 <- coxph(fm2,data=dtrain,x=TRUE)
```

Finally, we calculate the Brier score on training and test data.

```
library(pec)
fit.pec.train <- pec::pec(
  object=list("mod1"=fit1,"mod2"=fit2),
  data = dtrain,
  formula = Surv(time, status) ~ 1,
  splitMethod = "none")


fit.pec.test <- pec::pec(
  object=list("mod1"=fit1,"mod2"=fit2),
  data = dtest,
  formula = Surv(time, status) ~ 1,
  splitMethod = "none")

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