Predictive Modeling

Series 9

Exercise 9.1

We consider a data set of n observations $x_1, \ldots, x_n$. We draw a bootstrap sample $x_1^*, \ldots, x_n^*$. For a particular data point, x_j say, we now compute the probability that it is *not* part of the bootstrap sample.

a) Compute the probability that x_j is not the first observation in the bootstrap sample, that is
$$P(X_1^* \neq x_j)$$

b) Compute the probability that x_j is not the second observation in the bootstrap sample, that is
$$P(X_2^* \neq x_j)$$

c) Conclude by induction from your results in a) and b) that the probability for x_j not belonging to the bootstrap sample is
$$P(X_i^* \neq x_j, \text{ for all } 1 \leq i \leq n) = \left(1 - \frac{1}{n}\right)^n$$

d) Plot the probability in c) against $n = 0, \ldots, 100$. What asymptotic value is approached by the sequence? What is the exact value?

e) Imagine, that you draw B boostrap samples. What is the expected number of samples that do not contain the observation x_j?

Exercise 9.2

In this exercise we examine the bootstrap in order to estimate the variance of a (non-standard) estimator.

The following code produces two observations `X` and `Y` that are normally distributed and correlated. We think of the two values as returns of two financial assets.

```r
set.seed(123)
require(mvtnorm)
Sigma = matrix(c(5, -12, -12, 53), nrow = 2)
MV = rmvnorm(250, mean = c(12, 53), sigma = Sigma)
X = MV[, 1]
Y = MV[, 2]
```

We want to invest a fixed amount of money in the two assets; a fraction α is invested in X and a fraction $1 - \alpha$ in Y. We wish to choose α to minimize the risk, i.e. the variance, of our investment.

a) Show that the optimal α is given by

$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}$$

where σ_X^2 and σ_Y^2 are the variances of X and Y respectively and where σ_{XY} is the covariance of X and Y.
Hint: For the variance, the following relation applies: $\text{Var}(aX + bY) = a^2\sigma_X^2 + b^2\sigma_Y^2 + 2ab\text{Cov}(X, Y)$.

b) Estimate α by the plug-in estimator

$$\hat{\alpha} = \frac{\hat{\sigma}_Y^2 - \hat{\sigma}_{XY}}{\hat{\sigma}_X^2 + \hat{\sigma}_Y^2 - 2\hat{\sigma}_{XY}}$$

where $\hat{\sigma}_X^2$, $\hat{\sigma}_Y^2$ and $\hat{\sigma}_{XY}$ are the sample variances/covariance.

Hint: The **cov** function computes the sample covariance matrix. The following code does the trick for the variables **X** and **Y**.

```r
SigmaHat = cov(cbind(X, Y))
```

c) Compute $B = 2000$ bootstraped estimators for α and estimate the standard deviation of $\hat{\alpha}$ by the empirical standard deviation of the bootstrapped estimators. Try different numbers of boostrap replicates. What is the minimum number of replicates needed for this problem to become stable?

Hint: The function **sklearn.utils.resample()** can be used for taking the bootstrap samples: The **boot**-package implements the bootstrap loop. The following code-snippet implements the bootstrap for our problem

```r
library(boot)   #load the package

# Now we need the function we would like to estimate
# In our case the alpha:
alpha.fun <- function(data, ind) {
    SigmaHat = cov(data[ind, ])
    sx = SigmaHat[1, 1]
    sy = SigmaHat[2, 2]
    sxy = SigmaHat[1, 2]
    return(alpha = (sy - sxy)/(sx + sy - 2 * sxy))
}

# now you can bootstrap: R is the number of bootstrap
# samples
bootbet = boot(data = cbind(X, Y), statistic = alpha.fun,
    R = 2000)
plot(bootbet)   # do some plotting
sd(bootbet$t)   # compute the standard deviation
```

Exercise 9.3

We apply bagging and random forests to the **Boston** data that comes with the **MASS** library.Run the following code to get accustomed with the data

```r
library(MASS)
## `?`(Boston)
## summary(Boston)
```

The data set contains 506 records of averaged housing values in different suburbs of Boston. The variable **medv** contains the median value of owner occupied homes in 1000s \$. The remaining variables contain information on the social status of the population, nitrogen oxides concentrations, crime rates, etc. We aim at modelling the value of the houses by means of the remaining variabels. We will use (regression) random forests for the task.

a) Generate a training and test set of approximately equal size. Use a code snipped of the lecture notes or one of the previous exercises to do so. Call the resulting variables **train.set** and **test.set**, respectively.

b) Build a random forest (with default settings) on the training data. Get basic informations on the model by simply typing its name in the console. How many trees are used? How many variables are allowed in each split?

Hint: The `randomForest()` function takes a formula as input same as the `lm()` function. The following code line computes a random forest model with default parameters.

```
library(randomForest)
rfm = randomForest(medv ~ ., data = train.set)
rfm
```

c) Apply the `plot()` function to your model. Explain, what you see!

d) Predict the `medv` variable on the test set and compute the mean squared error between true and predicted values. Provide also a residual plot.

Hint: The `predict()` function is a generic function to predict values with a given model.

```
pred.val = predict(rfm, newdata = test.set)
sum((pred.val - test.set$medv)^2)/nrow(test.set)
```

e) The `randomForest()` function has an integer argument `mtry` which gives the number of allowed variables at each split. Try different values for `mtry` in particular `mtry=13` which corresponds to bagging. Find the value of `mtry` for which the test error is minimized.

f) Use the `importance()` command in order to determine the three most influential variables.

Exercise 9.4

In this exercise we will build some models to predict breast cancer from features computed from a digitized image of a fine needle aspirate (FNA) of a breast mass [1] The data set `BreastCancerData.csv` contains the data of 569 patients. For each of them the following parameters are stored

1) `ID number`

2) `Diagnosis` which is the binary response: `M` (malignant) `B` (benign)

[1] The data is taken from Lichman, M. (2013). UCI Machine Learning Repository http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science.

3)-32) 10 real valued features of each cell nucleus: radius, texture, perimeter, area, smoothness, compactness, concavity, number of concave points, symmetry, fractal dimension.

For each of these values the mean, standard deviation and the maximum are stored. So, predictor 3 is the mean radius, predictor 13 is the standard deviation of the radius and 23 is the maximal nucleus radius detected in the breast mass.

Load the data by:

```
BC = read.table("Daten/BreastCancerData.csv", sep = ",",
    header = F)
BC$V2 = as.factor(BC$V2)   #variable 2 is the response
```

a) Create a training set containing a random sample of 500 observations, and a test set containing the remaining observations. Refer Exercise 9.4 for the **R**-code .

b) Compute a random forest with $m = \sqrt{p}$ on the training set. Apply the **plot()** function to your model and interpret the output. **Hint:** The **randomForest** function takes a formula as input same as the **lm** function. The following code line computes a random forest model with default parameters.

```
library(randomForest)
rfm = randomForest(v2 ~ ., data = train.set[, -1])
```

Note that the first column of the data is removed (it is an ID number and hence not informative). Find out how to modify the parameter m.

c) From the plot in b) do you think that the default number of bootstrap replicates is sufficient? Find out how to modify the number of replicates in the **randomForest** function and use $B = 5000$ replicates.

d) Apply the functions **importance()** and **VarImpPlot()** to the model.

What are the 2 most influential variables? Produce a scatter-plot of these variables and use coloring according to the classes.

e) Predict the values in the test set and produce a confusion matrix. Compute the classification error and compare it to the training error.

Hint: The **predict** function is a generic way to apply a model to new data. This code snippet shows how to apply it

```
pred.class = predict(rfm, newdata = test.set[, -1], type = "class")
# confusion matrix
cm = table(pred.class, test.set$V2)
addmargins(cm)
```

f) Compute a tree model with the default settings in `R` (cf. Series 9) using only the two most important predictors. Apply the `summary()` function to the tree and infer the training error. Compare the result to the random forest and interpret the result.

Result Checker

E 9.1: a) $1 - \frac{1}{n}$ b) $1 - \frac{1}{n}$ d) $\frac{1}{e}$ e) $\left(1 - \frac{1}{n}\right)^n \cdot B \approx \frac{B}{e}$

Predictive Modeling

Solutions to Series 9

Solution 9.1

a) The probability is $1 - P(X_1^* = x_j)$ with $P(X_1^* = x_j) = \frac{1}{n}$.

b) Since we consider drawing with replacement, the probability to choose x_j does not change.

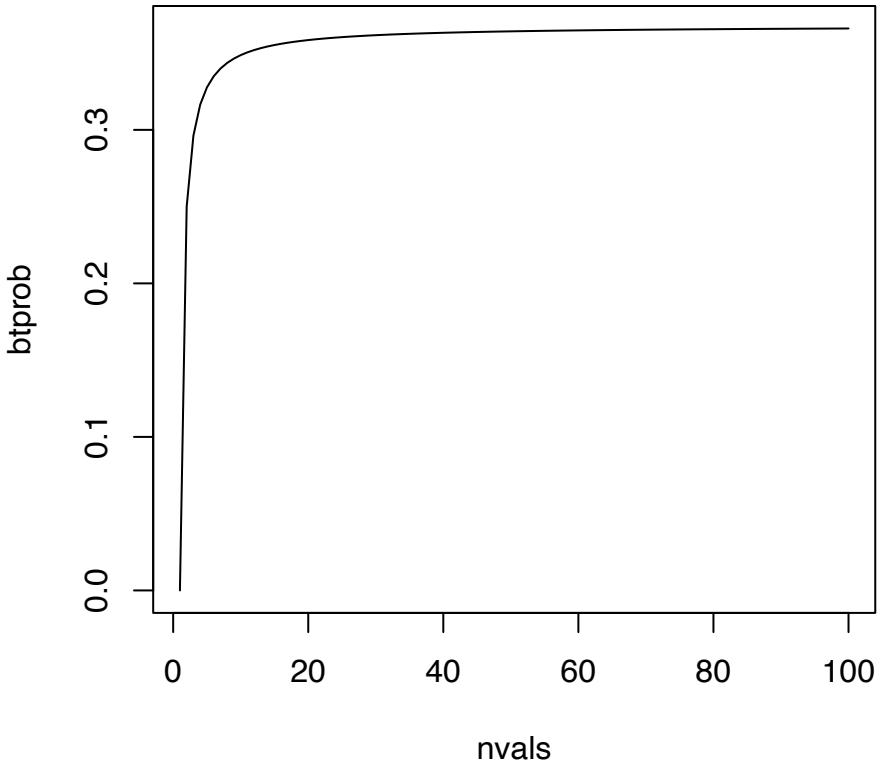
c) Since the events are independent, the probability to fulfil both conditions is the product of the two.

$$P(X_1^* \neq x_j \text{ and } X_2^* \neq x_j) = \left(1 - \frac{1}{n}\right)^2$$

Obviously, for n events this yields:

$$P(X_1^* \neq x_j \text{ and } X_2^* \neq x_j \text{ and } \ldots X_n^* \neq x_j) = \left(1 - \frac{1}{n}\right)^n$$

d)
```r
nvals <- c(1:100)
btprob <- (1 - 1/nvals)^nvals
plot(nvals, btprob, type = "l")
```



To calculate the asymptotic value of the sequence we have a look at $\lim_{n\to\infty}\left(1-\frac{1}{n}\right)^n$. To calculate this limit it is useful to know that the exponential function can be written as the limit of a sequence.

$$e^x := \lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n$$

Inserting $x = -1$ yields the limit

$$e^{-1} := \lim_{n\to\infty}\left(1-\frac{1}{n}\right)^n = \frac{1}{e}.$$

We are now able to compare the exact value with the last value of the sequence.

```
# last value of sequence
print(btprob[100])

## [1] 0.3660323

# exact value
1/exp(1)

## [1] 0.3678794
```

e) For every bootstrap sample, the probability that x_j is not contained in the sample is given by $\left(1 - \frac{1}{n}\right)^n$. Hence, the expected number of samples that do not contain x_j is

$$\left(1 - \frac{1}{n}\right)^n \cdot B \approx \frac{B}{e}.$$

Solution 9.2

a) For the variance, the following relation applies: $\text{Var}(aX + bY) = a^2\sigma_X^2 + b^2\sigma_Y^2 + 2ab\text{Cov}(X, Y)$. From this it follows that

$$\text{Var}(\alpha X + (1-\alpha)Y) = \alpha^2\sigma_X^2 + (1-\alpha)^2\sigma_Y^2 + 2\alpha(1-\alpha)\,\text{Cov}(X, Y)$$

If we take the derivative with respect to α and set the expression equal to zero, we find

$$\begin{aligned} 0 &= \frac{d}{d\alpha}\left(\alpha^2\sigma_X^2 + (1-\alpha)^2\sigma_Y^2 + 2\alpha(1-\alpha)\,\text{Cov}(X, Y)\right) \\ &= 2\alpha\sigma_X^2 + 2\alpha\sigma_Y^2 - 4\alpha\,\text{Cov}(X, Y) - 2\sigma_Y^2 + 2\,\text{Cov}(X, Y) \end{aligned}$$

Solving for α yields

$$\alpha = \frac{\sigma_Y^2 - \text{Cov}(X, Y)}{\sigma_X^2 + \sigma_Y^2 - 2\,\text{Cov}(X, Y)}$$

b)
```
SigmaHat = cov(cbind(X, Y))
sx = SigmaHat[1, 1]
sy = SigmaHat[2, 2]
```

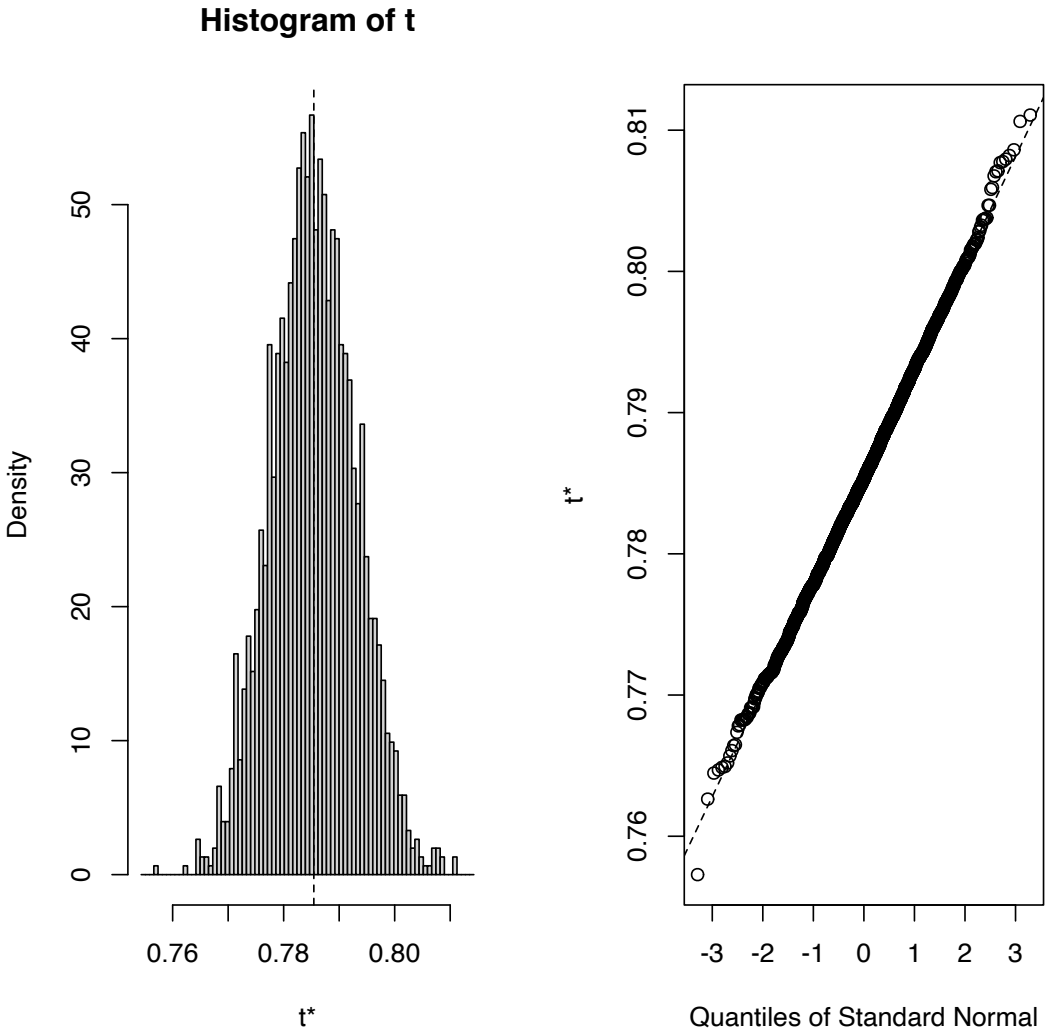
```
sxy = SigmaHat[1, 2]
alpha = (sy - sxy)/(sx + sy - 2 * sxy)
alpha

## [1] 0.785437
```

c)
```
library(boot)  #load the package

# Now we need the function we would like to estimate
# In our case the alpha:
alpha.fun <- function(data, ind) {
    SigmaHat = cov(data[ind, ])
    sx = SigmaHat[1, 1]
    sy = SigmaHat[2, 2]
    sxy = SigmaHat[1, 2]
    return(alpha = (sy - sxy)/(sx + sy - 2 * sxy))
}

# now you can bootstrap: R is the number of bootstrap
# samples
bootbet = boot(data = cbind(X, Y), statistic = alpha.fun,
    R = 2000)
plot(bootbet)  # do some plotting
```



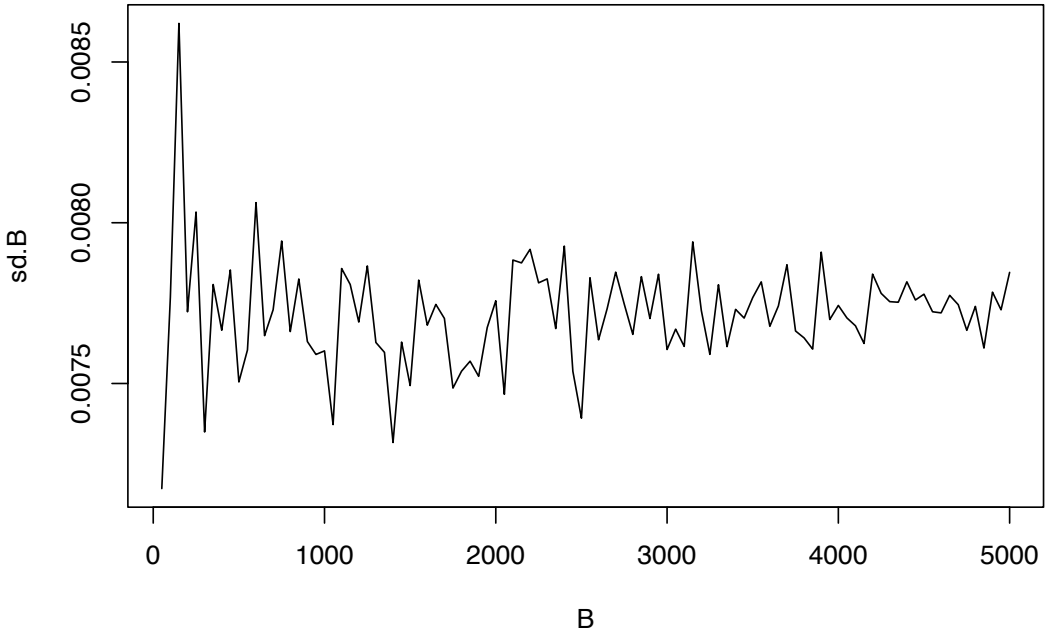

```
sd(bootbet$t)  # compute the standard deviation

## [1] 0.007553484
```

We investigate how the empirical standard deviation of $\hat{\alpha}$ changes as a function of the numbers of bootstrap replicates.

```
B <- seq(50, 5000, by = 50)
sd.B <- numeric(length(B))
for (i in B) {
    bootbet = boot(data = cbind(X, Y), statistic = alpha.fun,
        R = i)
    sd.B[i/50] <- sd(bootbet$t)  # compute the standard deviation
}
```

```r
plot(B, sd.B, type = "l")
```



We conclude that the estimation becomes roughly stable after 1000 bootstrap replicates.

Solution 9.3

a)
```r
set.seed(1)  #random seed for repeatability
idx.train = sample.int(nrow(Boston), size = nrow(Boston)/2)
train.set = Boston[idx.train, ]
test.set = Boston[-idx.train, ]
```

b)
```r
library(randomForest)
rfm = randomForest(medv ~ ., data = train.set)
rfm

##
## Call:
##  randomForest(formula = medv ~ ., data = train.set)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
```

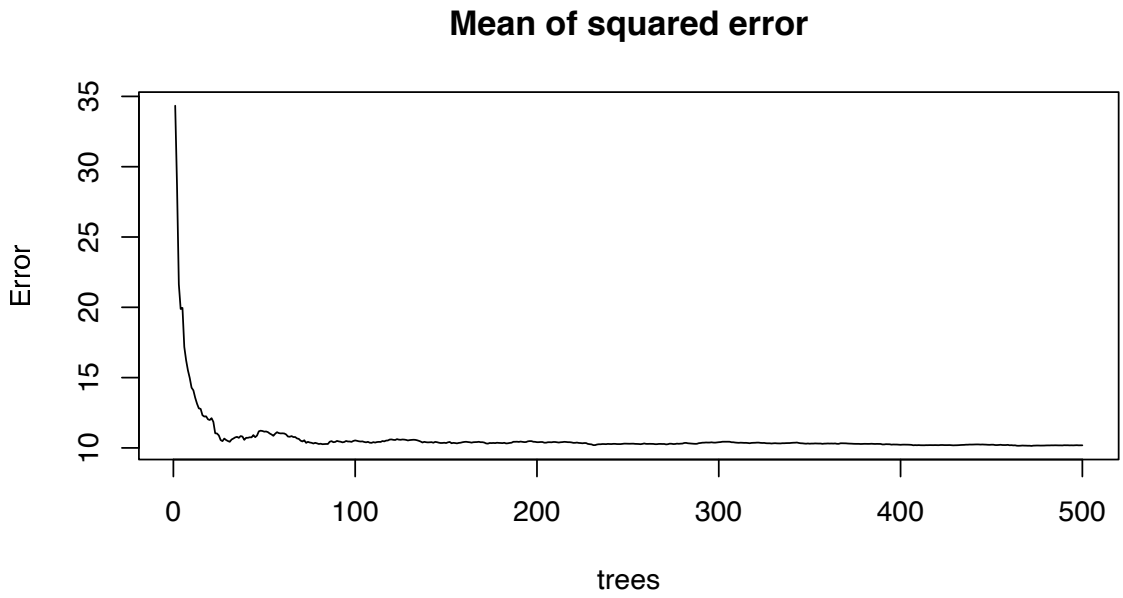


Abbildung 1: Mean of squared residuals for the Boston data

```
##           Mean of squared residuals: 10.18505
##                     % Var explained: 86.75
```

The output shows that the default setting for the number of trees **ntree** is 500 and that at each split only **mtry** variables are considered (that are chosen at random at each split).

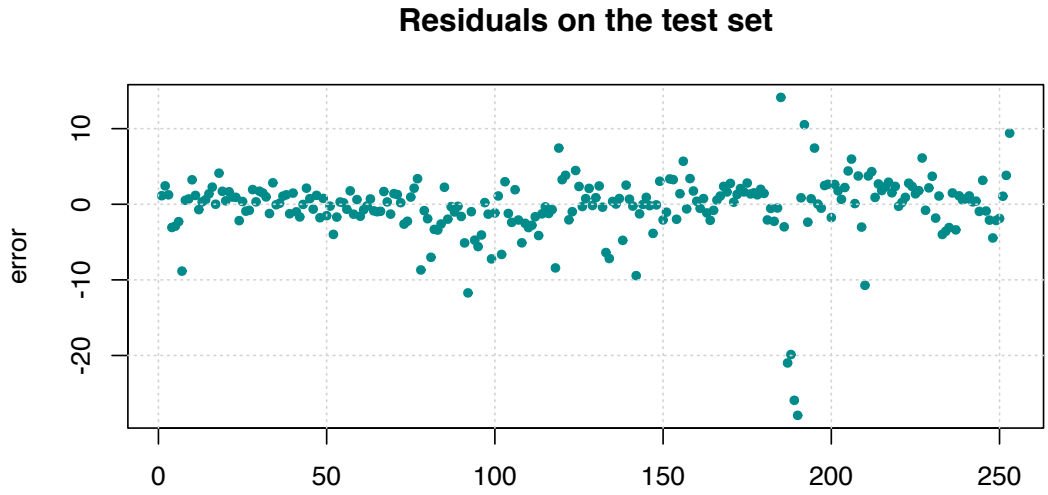
c)
```
plot(rfm, main = "Mean of squared error")
```

The plot shows how the mean of the squared error on the training set evolves with increasing number of trees. The values that are plotted are stored in **rfm$mse**.

d)
```
pred.val = predict(rfm, newdata = test.set)
sum((pred.val - test.set$medv)^2)/nrow(test.set)

## [1] 18.56047

plot(pred.val - test.set$medv, xlab = "", ylab = "error",
    pch = 20, col = "darkcyan", main = "Residuals on the test set")
grid()
```



e) The following code produces a random forest with 5 variables per split. It turns out that this is the optimal number that minimizes the test error.

```
rfm = randomForest(medv ~ ., data = train.set, mtry = 5)
pred.val = predict(rfm, newdata = test.set)
sum((pred.val - test.set$medv)^2)/nrow(test.set)

## [1] 18.64693
```

f)
```
importance(rfm)

##         IncNodePurity
## crim        1123.3330
## zn           130.8637
## indus        608.2750
## chas          67.3419
## nox          886.7656
## rm          6991.8532
## age          732.3495
## dis          742.9981
## rad          121.3822
## tax          421.8535
## ptratio     1102.3743
## black        313.3892
## lstat       5827.3664
```

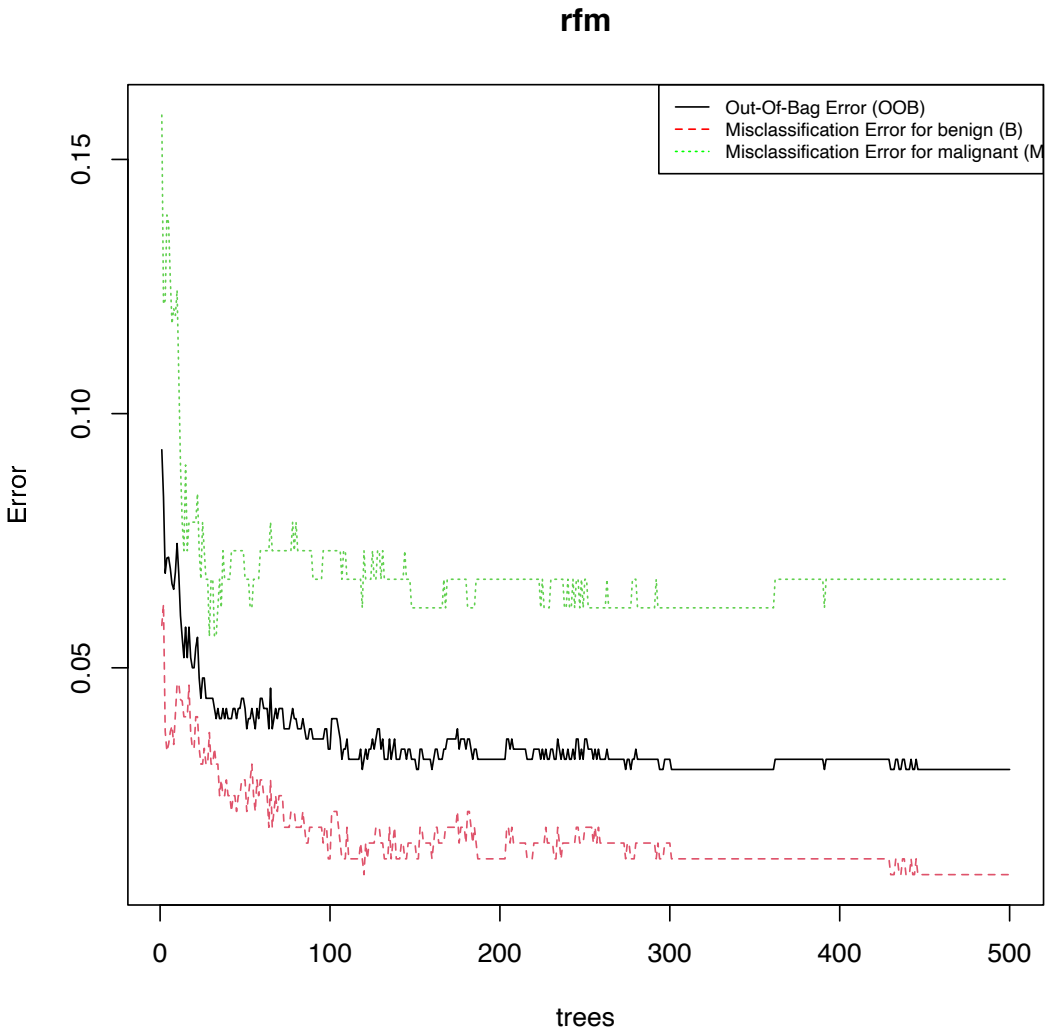
We hence find that **lstat** (ower status of the population in %), **rm** (average

number of rooms per dwelling) and **dis** (weighted mean of distances to five Boston employment centres) are the most important variables for explaining the housing values.

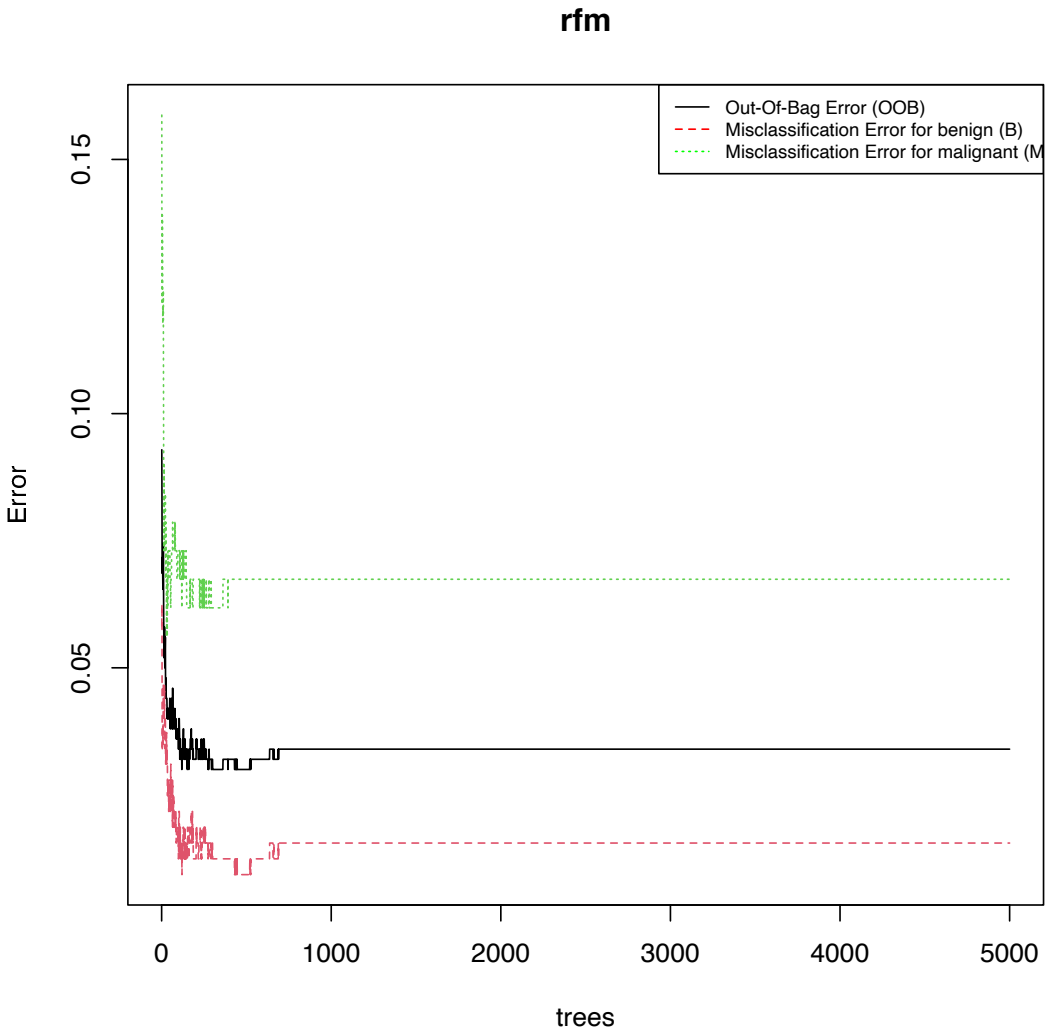
Solution 9.4

a)
```
set.seed(123)  #random seed for repeatability
idx.train = sample.int(nrow(BC), size = 500)
train.set = BC[idx.train, ]
test.set = BC[-idx.train, ]
```

b)
```
library(randomForest)
set.seed(123)
rfm = randomForest(V2 ~ ., data = train.set[, -1], mtry = sqrt(ncol(B
plot(rfm)
legend(x = "topright", cex = 0.7, legend = c("Out-Of-Bag Error (OOB)"
    "Misclassification Error for benign (B)", "Misclassification Erro
    col = c("black", "red", "green"), lty = c(1, 2, 3))
```

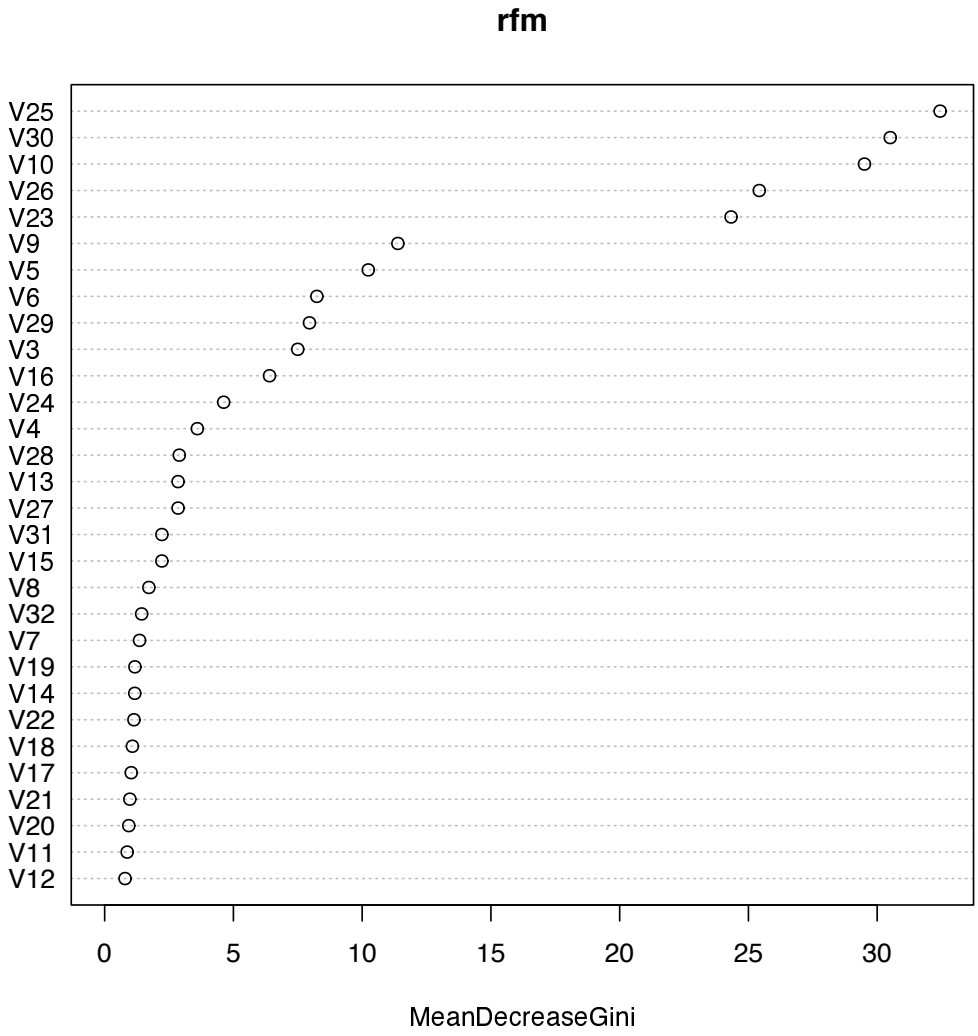


c)
```
library(randomForest)
set.seed(123)
rfm = randomForest(V2 ~ ., data = train.set[, -1], mtry = sqrt(ncol(
    ntree = 5000)
plot(rfm)
legend(x = "topright", cex = 0.7, legend = c("Out-Of-Bag Error (OOB)"
    "Misclassification Error for benign (B)", "Misclassification Erro
    col = c("black", "red", "green"), lty = c(1, 2, 3))
```



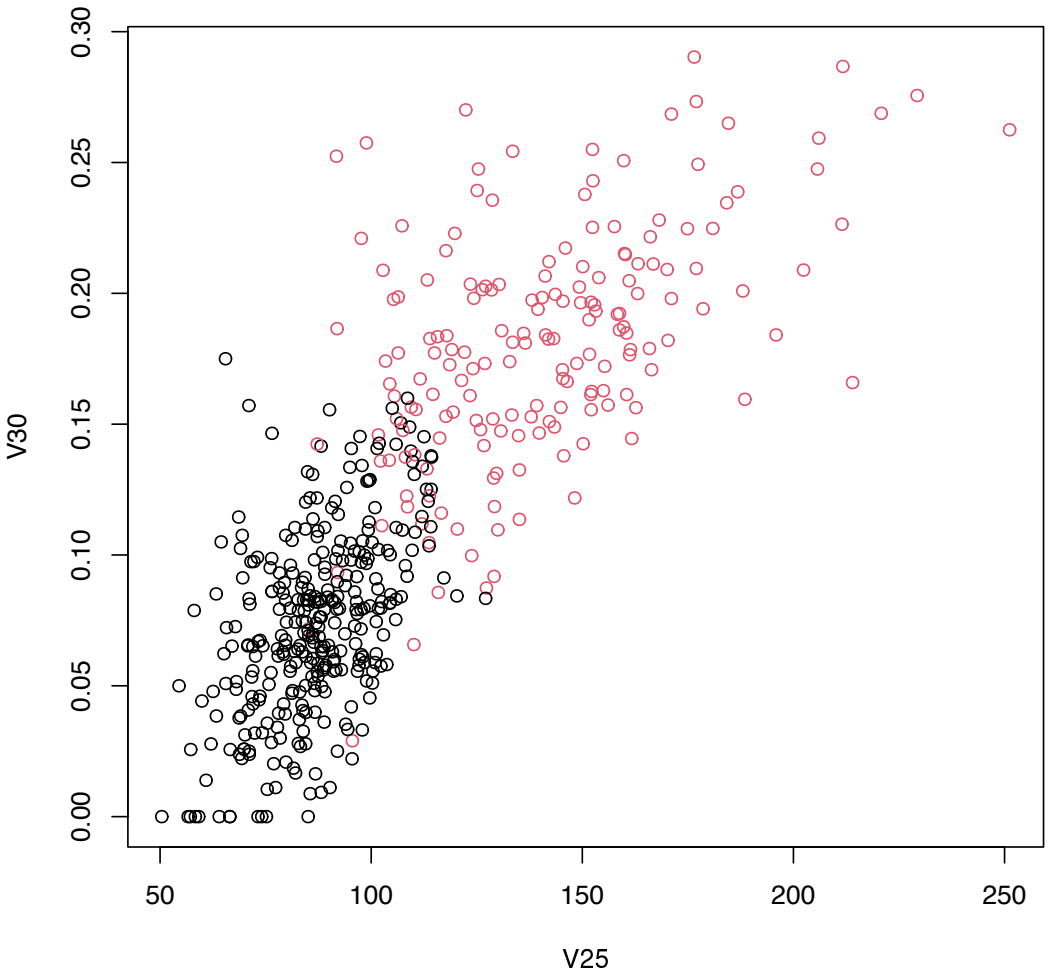
Since all three misclassification error types reach the minimum after approx. 500 trees (each tree corresponds to one bootstrap sample), the default number of 500 trees seems fine.

d)
```
varImpPlot(rfm)
```



The two most important predictor variables are **V25** and **V30**.

```
plot(V30 ~ V25, data = train.set, col = as.integer(train.set$V2))
```



e)
```
pred.class = predict(rfm, newdata = test.set[, -1], type = "class")
# confusion matrix
cm_test = table(pred.class, test.set$V2)
addmargins(cm_test)

##
## pred.class  B  M Sum
##          B 33  0  33
##          M  2 34  36
##          Sum 35 34  69
```

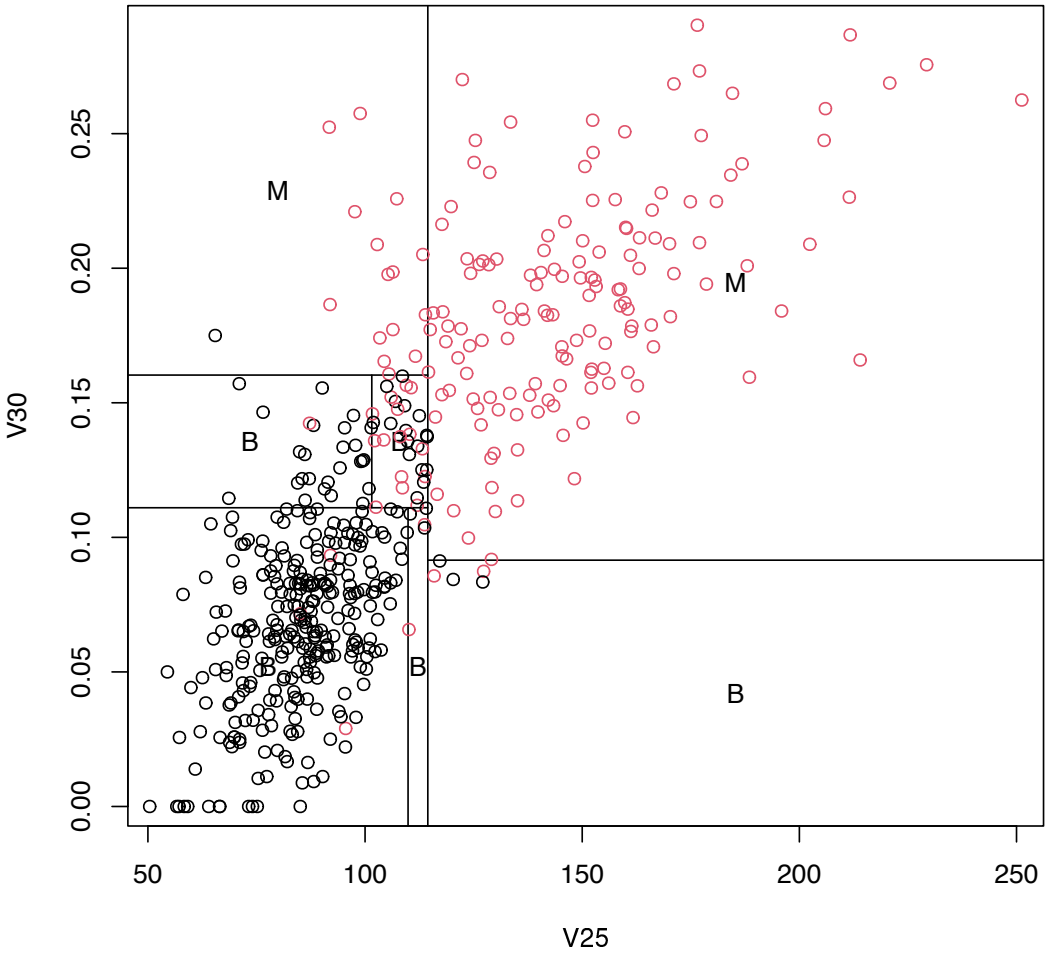
The test error is given by 0.0289855.

```
pred.class = predict(rfm, newdata = train.set[, -1], type = "class")
# confusion matrix
cm_train = table(pred.class, train.set$V2)
addmargins(cm_train)

##
## pred.class   B   M Sum
##          B 322   0 322
##          M   0 178 178
##          Sum 322 178 500
```

The training error is given by 0.

f)
```
library(tree)
tm = tree(V2 ~ V25 + V30, data = train.set[, -1])
partition.tree(tm)
points(V30 ~ V25, data = train.set, col = as.integer(train.set$V2))
```




```
summary(tm)

##
## Classification tree:
## tree(formula = V2 ~ V25 + V30, data = train.set[, -1])
## Number of terminal nodes:  7
## Residual mean deviance:  0.2165 = 106.8 / 493
## Misclassification error rate: 0.048 = 24 / 500
```