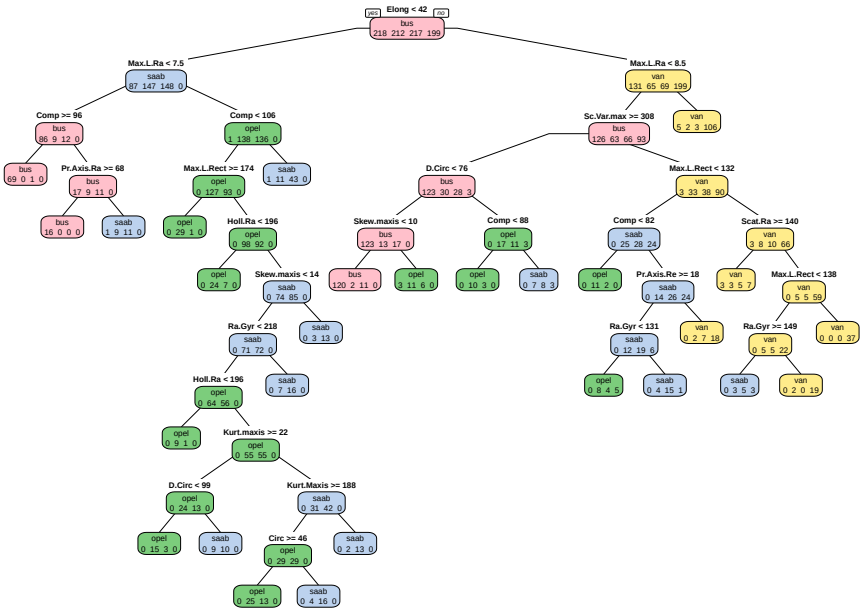
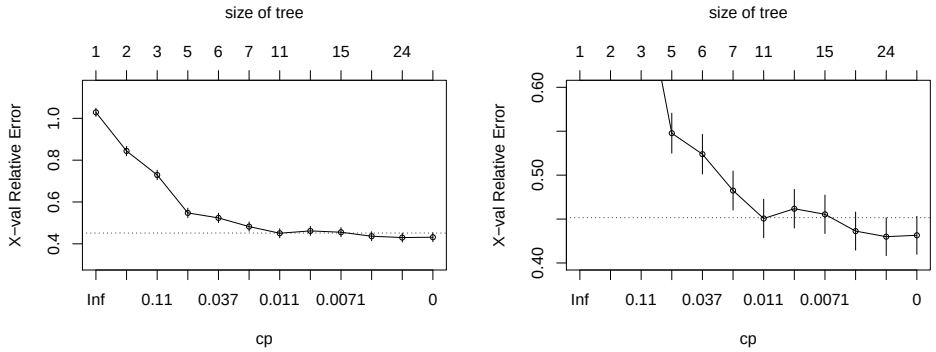
Solution to Series 10

1. a)
```
> library(rpart)
> set.seed(10)
> data <- read.table("http://stat.ethz.ch/Teaching/Datasets/NDK/vehicle.dat", header = TRUE)
> ## train an overfitted tree:
> tree <- rpart(Class ~ ., data = data,
                control = rpart.control(cp = 0.0, minsplit = 30))
> ## plot the tree
> require(rpart.plot)
> prp(tree, extra=1, type=1,
     box.col=c('pink', 'palegreen3',
                'lightsteelblue 2','lightgoldenrod 1')[tree$frame$yval])
>
```



The unpruned CART-fit has 27 terminal nodes and depth 12. Interpretation seems to be difficult because many of the predictors are selected to be substantial for the model at high interaction degrees. Many terminal nodes contain only a very small number of datapoints which is signal of significant overfit.

b) To prune the tree to make it cost-complexity optimal we first look at the cost-complexity plot; for better visibility we zoom in around the limiting line (right figure):

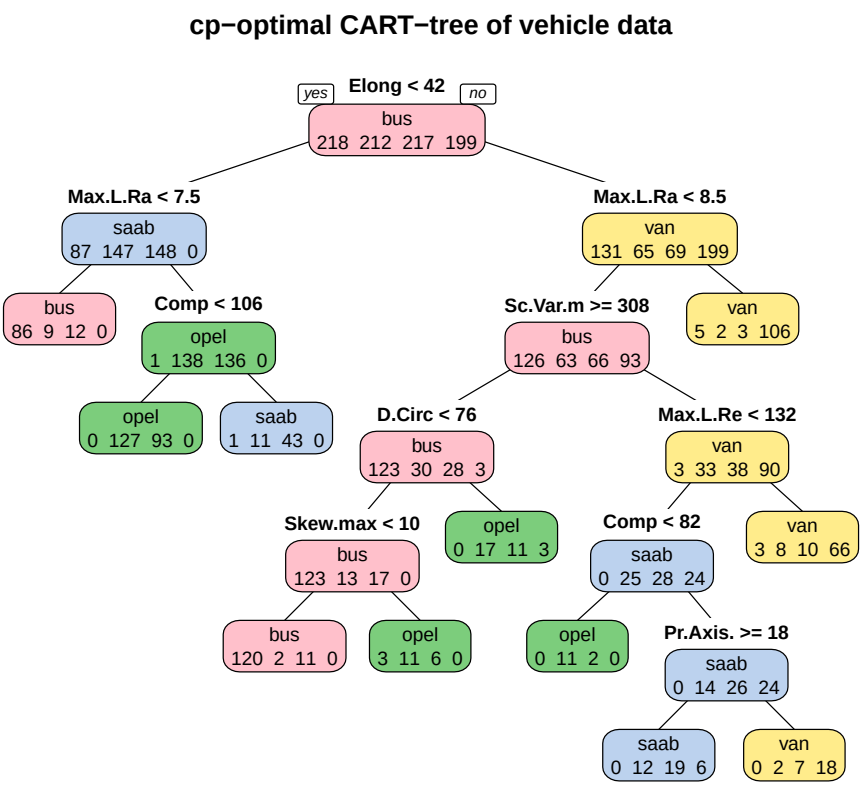


The full tree has the lowest cross-validation error and therefore defines the limit error (dashed line), which will help us to select the "optimal" tree using the one-standard-error rule. The smallest model whose error is below the line has size 11 (corresponding to 10 splits). To get the corresponding cp value, we look at the cost-complexity table:

```
            CP nsplit rel error    xerror       xstd
1  0.205414013      0 1.0000000 1.0286624 0.01967825
2  0.121019108      1 0.7945860 0.8439490 0.02240457
3  0.095541401      2 0.6735669 0.7292994 0.02307830
4  0.050955414      4 0.4824841 0.5477707 0.02275025
5  0.027070064      5 0.4315287 0.5238854 0.02257867
6  0.012738854      6 0.4044586 0.4824841 0.02220631
7  0.008757962     10 0.3535032 0.4506369 0.02185257
8  0.007961783     12 0.3359873 0.4617834 0.02198323
9  0.006369427     14 0.3200637 0.4554140 0.02190949
10 0.001592357     21 0.2643312 0.4363057 0.02167347
11 0.001061571     23 0.2611465 0.4299363 0.02158978
12 0.000000000     26 0.2579618 0.4315287 0.02161094
```

Thus for 10 splits we have a cp value of 0.00875796. The fitted cp-optimally pruned tree turns out to be much more handy and looks as follows:

```
> cp.opt <- tree$cptable[7, "CP"]
> tree.pruned <- prune.rpart(tree, cp = cp.opt)
> prp(tree.pruned, extra=1, type=1,
     main="cp-optimal CART-tree of vehicle data",
     box.col=
       c('pink', 'palegreen3',
              'lightsteelblue 2','lightgoldenrod 1')[tree.pruned$frame$yval])
```



c) The classification performance of CART turns out to be quite poor with a misclassification rate of 26.24%:

```
> mean(residuals(tree.pruned))
[1] 0.2624113
```

d) In this exercise we compare the bootstrap generalization error and the leave-1-out crossvalidated performance of the CART. Recall that the cp value depends on the data that is being used to generate the tree. Therefore, every time you want to prune a tree, you should determine the optimal tree size for that particular training sample using cross-validation.

To evaluate the performance of the algorithm, we write a function that yields the misclassification rate on a test set, based on a tree estimated from a training set:

```
> misclass.sample <- function(data, ind.training, ind.test)
 {
   tree <- rpart(Class ~ ., data = data[ind.training, ],
                         control = rpart.control(cp=0.0,minsplit = 30))

   ## choose optimal cp according to 1-std-error rule:
   min.ind <- which.min(tree$cptable[,"xerror"])
   min.lim <- tree$cptable[min.ind, "xerror"] + tree$cptable[min.ind, "xstd"]
   cp.opt <- tree$cptable[(tree$cptable[,"xerror"] < min.lim),"CP"][1]

   tree.sample <- prune.rpart(tree, cp=cp.opt)

   mean(data$Class[ind.test] != predict(tree.sample,
                                        newdata = data[ind.test, ], type = "class"))
 }
```

We start by computing the bootstrap generalization error:

```
> B <- 1000
> n <- nrow(data)
> boot.err <- function(data, ind)
   misclass.sample(data, ind, 1:nrow(data))
> boot.samples <- replicate(B, boot.err(data, sample(1:n, replace = TRUE)))
> errboot <- mean(boot.samples)
>
```

Finally we calculate the leave-one-out CV-value:

```
> cv.sample <- function(data, ind)
   misclass.sample(data, -ind, ind)
> cv.samples <- sapply(1:n, cv.sample, data = data)
> errcv <- mean(cv.samples)
```

We find a bootstrap generalization error of 0.2525 and a CV-value of 0.3392. The standard bootstrap approach tends to underestimate the generalization error since it uses the same data as training and as test set. We can overcome this problem by using the out-of-bootstrap sample to estimate the generalization error (cf. next task).

e)

```
> oobs.sample <- function(data, ind)
    misclass.sample(data, ind, -ind)
> oobs.samples <- replicate(B, oobs.sample(data, sample(1:n, replace = TRUE)))
> erroobs <- mean(oobs.samples)
```

The generalization error of 0.3226 found now is larger than the (standard) bootstrap generalization error calculated in **d)** (and the variance may be larger; see also Series 7) and much closer to the CV-value calculated in **d)**.