Exercise Series 10

1. In this series we are going to explore the dataset `vehicle.dat` which can be found at `"http://stat.ethz.ch/Teaching/Datasets/NDK/vehicle.dat"`. The dataset contains 846 observations of 19 variables. The aim is to classify the response (which is named `Class`) into four different car types (`bus,van,saab,opel`) by means of 18 predictors such as compactness, some information about the car axes and certain length ratios of the cars' silhouettes.

For this, we are going to use two competing classification methods, namely CART-trees with cost-complexity-optimized size and neural networks with variable number of hidden units. Besides analyzing the classification performance on the `vehicle` data, we are also interested in the predictive power of the two methods. To evaluate the generalization error we are going to do a bootstrap analysis in exercise 2.

For CART the optimal tree size can be found automatically using the methods from package `rpart` whereas for neural networks we have to find the optimal number of hidden units ourselves by performing a 10fold inner cross-validation. To access the methods dealing with neural networks you need to load the package `nnet`.

1. a) First of all, generate a classification tree using the methods from `rpart`. Set the options `cp = 0` and `minsplit = 30` such that the resulting tree becomes too large and overfits the data. To visualize the tree properly you have to make a suitable choice for the parameters of `plot` and `text`. For details look at `?plot.rpart` and `?text.rpart`. Try to interpret the tree. Use `set.seed(100)` for reproducibility.

R-Hints:

```
library(rpart)
t.formula <- Class ~.
r.rp <- rpart(t.formula,data=???,control=rpart.control(cp=0.0,minsplit=30))
plot(r.rp,???)
text(r.rp,???)
```

b) Now it comes to pruning the tree from part **a)**. We let `rpart` perform a cost-complexity-analysis to find an optimal `cp`-value by cross-validating a sequence of subtrees of the tree in **a)**. Read off the optimal `cp` from the cost-complexity-table (optimality is to be understood according to the *one standard-error rule*), visualize the pruned tree with the optimal `cp` and finally calculate its misclassification rate.

R-Hints:

```
# to access the cost-complexity table use:
printcp(r.rp)
# to plot classification error (relative to root tree)  vs. subtree size
# (dotted line represents one standard error limit) use:
plotcp(r.rp)
# to prune the tree use method "prune.rpart":
rp.pr <- prune.rpart(r.rp,cp = ???)
# for visualization use "plot" and "text" again:
# for misclassification rate look at:
?residuals.rpart
```

c) Next we want to fit a neural network with variable number of hidden units. To prevent overfitting and speed up the optimization process neural networks can be penalized by the sum of squares of the weights ω_i. The regulating proportionality factor is called `decay`, because a high `decay`-value obviously shrinks the weights. Such a shrinkage only makes sense if the predictors are scaled to the same order of magnitude (usually to have mean 0 and standard deviation 1). Therefore whenever using the `decay`-option you should scale your data first. We have already performed for you a search for the optimal `decay`-parameter by an inner cross-validation and found `dec.opt = 0.0045`. It's left to you to search for the optimal number of hidden units `size.opt`. Write a function that performs a 10fold inner cross-validation to find `size.opt`. The maximal size you should consider is `size = 10`. Bear in mind that `nnet` chooses random starting values. This means that all your `nnet`-fits should be averaged over, say, `nreps = 10` realizations. Finally, calculate the misclassification rate for the optimal `nnet`.

R-Hints:

```
# to scale a data frame look at:
?scale

# always use "trace=FALSE" and optimal decay "decay=dec.opt":
learn <- nnet(Class ~.,data = ???, trace = FALSE, size = ???,decay=dec.opt,...)

# as a prediction result for nnet you get the probabilities for the
# four different factors. Averaging over starting values can be done
# as follows (can be used inside your CV-code):
res <- matrix(0,nrow(data),length(levels(Class)))
for(rep in 1: nreps){
            learn <- nnet(Class ~., data = ???, maxit = 500, trace = FALSE, ...)
            res[???] <- res[???] + predict(learn, data[???])
}

# number of misclassifications:
sum(as.numeric(Class) != max.col(res/nreps))
```

d) Next try to illustrate the optimal CART and nnet-fits on a two dimensional cross-section. To do that we choose the two most selective variables according to the pruned CART tree from **b)** and set all other predictors to their mean values. On a 2D-grid we plot the classification decision boundary lines for the optimal CART tree and for a few realizations of the optimal nnet and their average as well. You can use the functions `plt`, `plt.bdy` and `b1`, all defined in the file.

`http://stat.ethz.ch/teaching/lectures/FS_2008/CompStat/CARTnetplots.R`.
Try to understand what those functions can do for you (`t.ds` and `t.dsnet` stand for the vehicle and scaled vehicle data respecively) and then invoke them correctly with the correct arguments. The resulting plots might look a bit strange, this is because we are only looking at the projections of the high-dimensional `vehicle`-data onto a two-dimensional cross-section. Such a plot has to be interpreted with care.

2. To arrive at a decision whether CART or neural networks performs better in the `vehicle`-setting, compute the bootstrap generalization error for the two methods (see manuscript p.44). You might want to use the same bootstrap samples for both methods. Because `nnet` and `rpart` are rather slow, you have to restrict yourself to a rather small number of bootstrap-samples, eg. `B=20`. Choose `nreps=5` for `nnet` here. Note that it is *not* the same to evaluate the predictive performance of the two methods in order to find optimal parameters in an inner CV-analysis on one hand and for a final full analysis on the other hand. Therefore this exercise is definitely not redundant.

If you have solved exercise 1, then use the number of layers you have found out to perform best in exercise 1. If you haven't solved exercise 1, use 6 layers and a decay value of 0.0045.

Preliminary discussion: Friday, May 16, 2008. **Deadline:** Friday, May 23, 2008.