Exercise Series 8

1. Consider again the ozone dataset. As in exercise 8 take the log of the response and remove the outlier.

Now we focus on **projection pursuit regression (PPR)**. We want to compare models with different number of ridge functions and different smoothers. Use leave-one-out cross-validation to compare models.

a) Choose three different smoothers (argument `sm.method`). The default is `supsmu`, Friedmans "super smoother". The other two possibilities are `spline` which uses splines with a specified (equivalent) degree of freedom for each ridge function and `gcvspline` which chooses the smoothness by GCV.
For each of these smoothers vary the numbers of ridge functions (`nterms`), from 1 to 8. If your computer power allows you can also try different degrees of freedom (`df`) for `spline`, from 1 to 6.
Use `max.terms` to get better results, `max.terms`=`nterms`+3. The following description is taken from the help-file of `ppr`:
The algorithm first adds up to 'max.terms' ridge terms one at a time; it will use less if it is unable to find a term to add that makes sufficient difference. It then removes the least "important" term at each step until 'nterms' terms are left.
R-Hints:

```
## Recall d.ozone.e denotes the edited ozone data and
## iy is the index of the response logupo3.

## The first argument is a *function*. "..." are arguments which are passed
## to fitfn. Do *not* replace these dots!
cv <- function(fitfn, ...){
  ssr <- 0
  for(j in 1:nrow(d.ozone.e)){
    print(j)
    train <- d.ozone.e[-j,]
    test <- d.ozone.e[j,]
    fit <- fitfn(x = train[,-iy], y = train[,iy], ...)
    ssr <- ssr + (test[iy] - predict(fit, test[-iy]))^2
  }
  ssr
}

## Example to call simultaneously for different values of nterms:
cv.sm <- numeric(length=8)
for(i in 1:8){ ## print(i)
  cv.sm[i] <- cv(ppr, nterms = i, maxt.terms = i+3)}
```

b) Choose the best model from a) and visualize the ridge functions as in the manuscript on page 67.

c) Interpret the α-matrix (`$alpha`) for your model. Because the predictors are on different scales (check `apply(d.ozone.e,2,sd)`), the interpretation is not straight forward. To correct the scale you can multiply each row of the α-matrix with the standard deviation of the corresponding predictor and rescale the columns to unit length. You can use `round()` to get a better overview. If you have another model which performs nearly as well as the best but has a much nicer interpretation you may want to prefer it to the best model.

d) Compare your PPR-model to MARS with an interaction degree of 2. You can use the `cv`-function from above: `cv(mars, degree = 2)`.

2. In this exercise 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.

We are going to use two competing classification methods to fulfill this task, namely CART-trees with cost-complexity-optimized size and neural networks with variable number of hidden units.

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`.

Packages: rpart and nnet

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 units. To prevent overfit 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 layers `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` (below).

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.

```
##plot factors
plt <- function(dat,xcol,ycol,lcol,...)
{
  plot(dat[,xcol],dat[,ycol],type="n",xlab=names(dat)[xcol],ylab=names(dat)[ycol],
       main = "Cross-Section Classification Plot",...)
  chars <- c("b","o","s","v")
  for(l in 1:length(levels(dat[,lcol])))
      {
        set <- dat[,lcol] == levels(dat[,lcol])[l]
        text(dat[set,xcol],dat[set,ycol],chars[l],col = 2+l)
      }
}

##plot decision lines
b1 <- function(Z,...)
{
  for(i in 1: length(levels(dat[,lcol])))
      {
      z <- Z[,i] - apply(Z[,-i],1,max)
      c <- contourLines(x,y,matrix(z,gr.len),levels = 0.0)
      if(length(c) !=0) points(c[[1]]$x,c[[1]]$y,type="l",...)
      }
}

#compute 2D-nnet-classification-decision lines for CART and nnet
plt.bdy <- function(formula,xcol,ycol,lcol,size=size.opt,decay=dec.opt,
                    nrep=10,gr.len=100,...)
{
  mult.fig(2,main="Cross Section-Classification-Plots for CART and nnet")
  plt(t.ds,xcol,ycol,lcol,xlim=c(0,18))
  x.rg <- range(t.ds[,xcol])
  y.rg <- range(t.ds[,ycol])
  x <- seq(x.rg[1],x.rg[2],length=gr.len)
  y <- seq(y.rg[1],y.rg[2],length=gr.len)
  mn <- apply(t.ds[-lcol],2,mean)
  std <- apply(t.ds[-lcol],2,sd)
  gr <- as.data.frame(matrix(sapply(mn,rep,gr.len^2),gr.len^2,ncol(t.ds)-1))
  gr.sc <- as.data.frame(matrix(0,gr.len^2,ncol(t.dsnet)-1))
  gr[,c(xcol,ycol)] <- expand.grid(x,y)
  gr.sc[,c(xcol,ycol)] <- expand.grid((x-mn[xcol])/std[xcol],(y-mn[ycol])/
                                                                std[ycol])
  names(gr) <- names(gr.sc) <- names(t.ds)[-lcol]
  Z <- matrix(0,nrow(gr),length(levels(t.ds[,lcol])))
  tree.pr <- predict(rp.pr,newdata=gr,type="prob")
  b1(tree.pr,lwd = 3)
  plt(t.ds,xcol,ycol,lcol,xlim=c(0,18))
  #compute nnet average
  for(iter in 1:nrep)
    {
      fit.nn <- nnet(formula,t.dsnet,maxit=500,size=size.opt,decay=dec.opt,
                     trace=FALSE,...)
      fit.pr <- predict(fit.nn,gr.sc)
      b1(fit.pr)
      Z <- Z + fit.pr
    }
  b1(Z, lwd=3, lty = 2)
}
```

Preliminary discussion: Friday, June 9, 2006. **Deadline:** Friday, June 16, 2006.

Advice: Thursdays from 12.00-13.00, LEO C15, Leonhardstr. 27. Or contact either Bernadetta Tarigan, `tarigan@stat.math.ethz.ch`, or Nicoleta Gosoniu, `gosoniu@ifspm.unizh.ch`.