Series 9

1. Consider again the ozone dataset. In this exercise we focus on **projection pursuit regression (PPR)** and on **multivariate adaptive regression splines (MARS)**. We are interested in comparing PPR models with different number of ridge functions and different smoothers, and MARS models with different degrees of interaction. We use leave-one-out cross-validation to compare the models (see **R-Hints** below).

a) Take the log of the renamed response variable **O3**, remove the outlier and then standardize the predictors. Note that for PPR, the α-matrix, i.e., the matrix containing as columns the "projection" vectors $\alpha_1, ..., \alpha_q \in \mathbb{R}^p$, is easier to interpret when the explanatory variables are on the same scale. You can use the following R -code:

```
> set.seed(123)
> data(ozone, package = "gss")
> chgNames <- function(dfr, oldNms, newNms) {
    colnames(dfr)[colnames(dfr) %in% oldNms] <- newNms
    dfr
 }
> d.ozone <- chgNames(ozone,
                      oldNms = c("upo3", "wdsp",   "hmdt",   "sbtp"),
                      newNms = c("O3",   "wind", "humidity", "temp"))
> d.ozone <- subset(transform(d.ozone, "logO3" = log(O3)), select = -O3)
> d.ozone.e <- d.ozone[-which.max(d.ozone[,"wind"]),]
> d.ozone.es <- d.ozone.e
> d.ozone.es[,-10] <- scale(d.ozone.e[,-10])
```

b) Fit the data with the MARS model with maximal interaction degree 2. In R this can be done with the function `earth()` from the package of the same name. To visualize the effects of the predictors, we first plot the main effects and then the interactions. This can be done with the function `plotmo()`, first with `degree2=FALSE` and then with `degree1=FALSE`.

c) We now apply PPR to our dataset. First, fit a model with the default smoother (Friedman's "super smoother") and 4 terms. For this, use the function `ppr()`. Then, plot the ridge functions. Looking at the graphs, can you say that the model is good? Why?

d) The goal is now to find the best set of tuning parameters for our model (smoother, number of terms, and degrees of freedom) using PPR. We are mainly interested in prediction accuracy and will then use CV-error as a criterion. Use the function `cv()` (see **R-Hints** below) to compute the CV-error for a given set of parameters. Note that the values of the parameters can be specified directly as arguments of the function `cv()`.

There are three different smoothers to choose from (argument `sm.method`). The default is `supsmu`, Friedmans "super smoother". Other possibilities are `spline`, which uses splines with a specified (equivalent) degrees of freedom for each ridge function, and `gcvspline`, which also uses splines but chooses the smoothness by GCV. For each of these smoothers, vary the numbers of ridge functions (`nterms`) for `nterms` $\in \{3, 4, 5\}$. Aditionally, for the case `sm.method="spline"`, vary the degrees of freedom (`df`) for `df` $\in \{5, 6, 7\}$.

In all your models, you may also want to specify a value for `max.terms` to get better results. Look for details in the help-file of `ppr` (i.e. `?ppr`).

e) Fit a MARS model to the data varying the maximum interaction degree for 1, 2, and 3. You can use the `cv`-function: `cv(earth, degree = ?)`. Then, print the CV-errors of all the PPR models and all the MARS models. Which model performs best? Optionally, you can also compare these results with the performance of the GAM model that you used in the last series.

f) Choose the best PPR model (in terms of CV-error) and plot the ridge functions like in Section 7.5 of the lecture notes. Compare this model with the one from **c)**.

g) Interpret the α-matrix (see YourFit$alpha) of the model from **f)** (for example, what does a large value of an element of α mean?). 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.

R-Hints: The following function is a general function for cross validation. You can use it in this exercise, but also later on for other exercises or your own projects. Therefore, it might be worth trying to understand the functions `model.frame` and `model.response`.

```
> cv <- function(fitfn, formula = logO3 ~ . , data = d.ozone.es, ..., trace = TRUE)
 {
   modFrame <- model.frame(formula, data = data)
   nc <- nrow(data)
   ssr <- 0
   if(trace) cat("  j = ")
   for(j in 1:nc) {
     if(trace) cat(if(j %% (nc %/% 10) == 1) paste(j, "") else ".")
     ## Fit without 'j' :
     fit <- fitfn(formula=formula, data = data[-j ,], ...)
     ## Evaluate at 'j' :
     ssr <- ssr + (model.response(modFrame)[j] - predict(fit, modFrame[j,]))^2
   }
   if(trace) cat("\n")
   ssr
 }
```

You can get the code of the function `cv` with the following command:

```
> source("http://stat.ethz.ch/Teaching/R/Scripts/cv-fun.R")
```

Remarks:

1. Scaling of the result of the `cv`-function with the number of observations is not necessary as long as we are only interested in comparing the models.
2. The first argument of `cv()` is a **function**. "`...`" are multiple arguments which are passed to `fitfn`.
3. The default values for formula and data in `cv()` are specified to work with the transformed and standardized ozone dataset `d.ozone.es`.
4. Example of usage:

```
cv.gcv.2 <- cv(ppr, sm.method="gcvspline", nterms = 2, max.terms = 5)
```

Preliminary discussion: Friday, May 13.

Deadline: Friday, May 20.