GEOSTAT
Mastering Machine Learning for Spatial Prediction 2

Exercises

Madlene Nussbaum

13 August 2017

Contents

1 Selection of covariates 1

2 Partial dependence plots 4

3 Prediction uncertainty with model-based bootstrapping 6

Preparation

Load needed packages:

```
library(randomForest)
library(geoGAM)
```

Load again the data, select the calibration set and remove missing values in covariates.

```
dim(berne)

## [1] 1052  238

# Continuous response
d.ph10 <- berne[ berne$dataset == "calibration" & !is.na(berne$ph.0.10), ]
d.ph10 <- d.ph10[ complete.cases(d.ph10[13:ncol(d.ph10)]), ]
# covariates start at col 13
l.covar <- names(d.ph10[, 13:ncol(d.ph10)])
```

1 Selection of covariates

For tree based ensemble methods covariate importance can be computed. Based on this measure non-relevant covariates can be excluded and possibly model performance can be increased.

Fit random forest model:

```
rf.ph <- randomForest(x = d.ph10[, l.covar],
                      y = d.ph10$ph.0.10)
```

Create the importance plot:

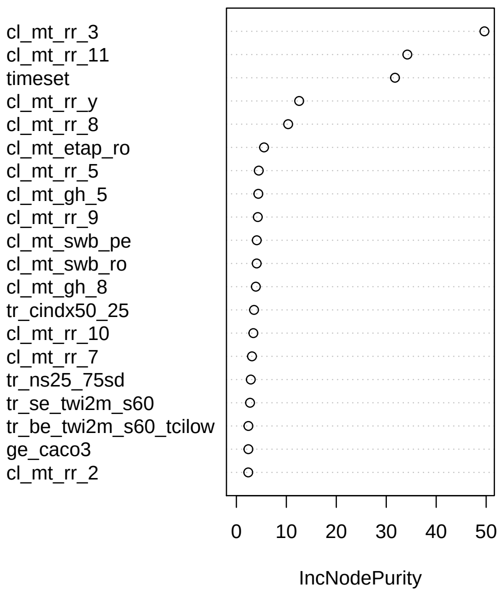

Figure 1: Covariate importance of 20 most important covariates for topsoil pH (before selection).

```
varImpPlot(rf.ph, n.var = 20, main = "")
```

Then, reduce covariates by recursive backward elimination using permuted covariate importance (`type = 2` in `importance()`):

```
# speed up the process by removing 5-10 covariates at a time
s.seq <- sort( c( seq(5, 95, by = 5),
                  seq(100, length(l.covar), by = 10) ),
               decreasing = T)

# collect results in list
qrf.elim <- oob.mse <- list()

# save model and OOB error of current fit
qrf.elim[[1]] <- rf.ph
oob.mse[[1]] <- tail(qrf.elim[[1]]$mse, n=1)
l.covar.sel <- l.covar

# Iterate through number of retained covariates
for( ii in 1:length(s.seq) ){
  t.imp <- importance(qrf.elim[[ii]], type = 2)
  t.imp <- t.imp[ order(t.imp[,1], decreasing = T),]

  qrf.elim[[ii+1]] <- randomForest(x = d.ph10[, names(t.imp[1:s.seq[ii]])],
                                   y = d.ph10$ph.0.10 )
  oob.mse[[ii+1]] <- tail(qrf.elim[[ii+1]]$mse,n=1)

}
```

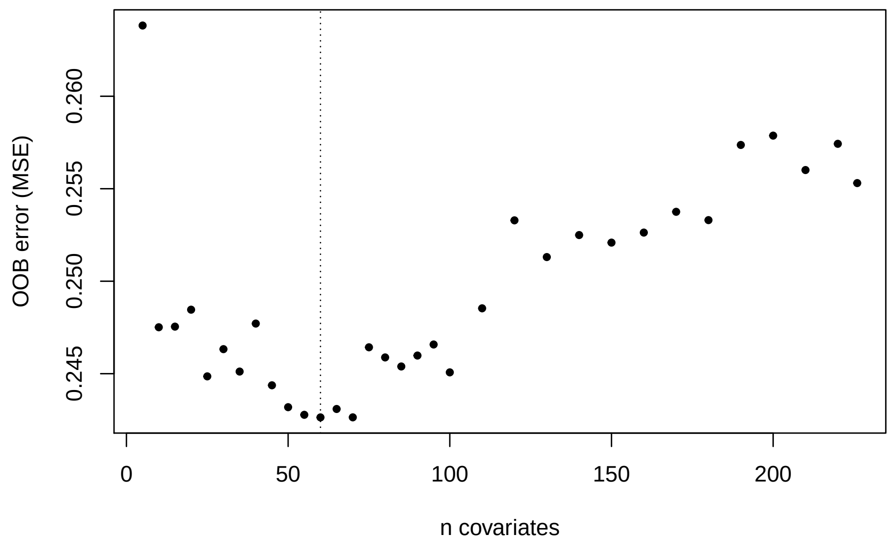

Figure 2: Path of out-of-bag mean squared error as covariates are removed. Minimum is found at 55 covariates.

```
# Prepare a data frame for plot
elim.oob <- data.frame(elim.n = c(length(l.covar), s.seq[1:length(s.seq)]),
                       elim.OOBe = unlist(oob.mse) )
```

Please continue:

- Optimize m$_{try}$ before you start the covariate selection (function `train`, package `caret`). How much does the OOB error decrease? Are both steps (tuning, selection) worth the effort from a point of view of prediction performance?
- Implement the same covariate selection for gradient boosting with trees as baselearners (package `gbm` or `caret`). Do you find the same covariates in the final set? Why do you expect differences?

2 Partial dependence plots

Interpretation of the most important covariates can be done by partial dependence plots. But keep in mind that the remaining covariate set might be still multi-collinear, hence covariates might be exchangeable.

```
# select the model with minimum OOB error
rf.selected <- qrf.elim[[ which.min(elim.oob$elim.OOBe)]]

t.imp <- importance(rf.selected, type = 2)
t.imp <- t.imp[ order(t.imp[,1], decreasing = T),]

# 4 most important covariates
( t.3 <- names( t.imp[ 1:4 ] ) )

## [1] "cl_mt_rr_3"  "cl_mt_rr_11" "timeset"     "cl_mt_rr_y"

par( mfrow = c(2,2))

# Bug in partialPlot(): function does not allow a variable for the
#  covariate name (e. g. x.var = name) in a loop
partialPlot(x = rf.selected,
            pred.data = d.ph10[, names(rf.selected$forest$xlevels)],
            x.var = "cl_mt_rr_3", ylab = "ph [-]", main = "")
partialPlot(x = rf.selected,
            pred.data = d.ph10[, names(rf.selected$forest$xlevels)],
            x.var = "cl_mt_rr_11", ylab = "ph [-]", main = "" )
partialPlot(x = rf.selected,
            pred.data = d.ph10[, names(rf.selected$forest$xlevels)],
            x.var = "timeset", ylab = "ph [-]", main = "" )
partialPlot(x = rf.selected,
            pred.data = d.ph10[, names(rf.selected$forest$xlevels)],
            x.var = "cl_mt_rr_y", ylab = "ph [-]", main = "" )
```

Please continue:

- Create partial dependence plots for the boosted trees model (`?plot.gbm`, `plot(.., i.var = ..)`). Do you find the same relationships?
- What do you conclude from the plots? Are the plotted covariates good predictors?

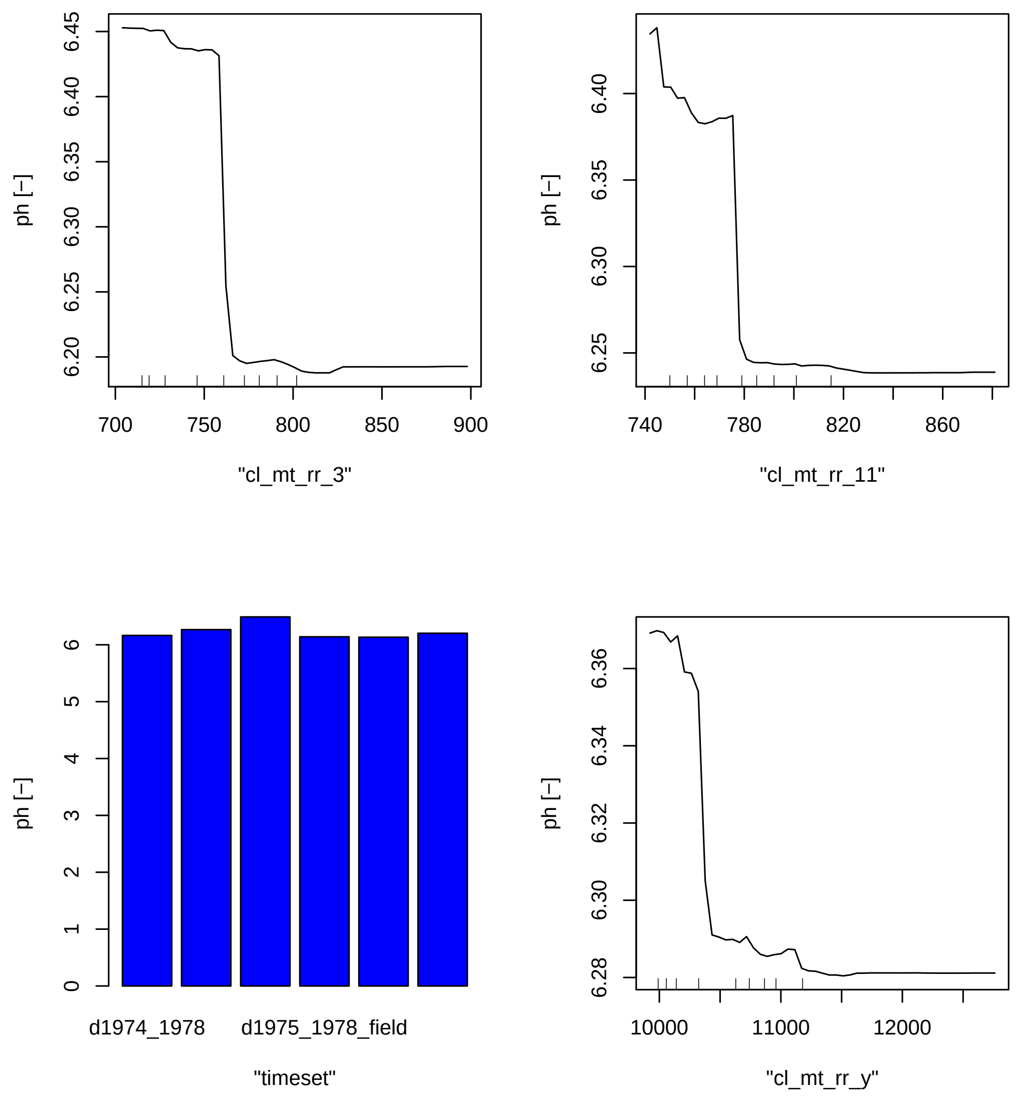

Figure 3: Partial dependence plots for the 4 most important covariates.

3 Prediction uncertainty with model-based bootstrapping

When reporting predictions it is important to give prediction uncertainty along with them. For any method not yielding uncertainty estimates form the method itself (e.g. kriging variances) a model-based bootstrap approach can be used.

```
# Fit a geoGAM model
# only use 4 covariates to speed up
ph.geogam <- geoGAM(response = "ph.0.10",
                    covariates = c("timeset", "cl_mt_rr_3",
                                   "tr_se_twi2m_s60", "sl_physio_neu"),
                    data = d.ph10,
                    max.stop = 20)

# select validation data
d.ph10.val <- berne[berne$dataset == "validation" & !is.na(berne$ph.0.10), ]
d.ph10.val <- d.ph10.val[complete.cases(d.ph10.val[13:ncol(d.ph10)]), ]

# compute predictions (mean) for each validation site
ph.pred <- predict(ph.geogam,
                   newdata = d.ph10.val)
```

Compute model based bootstrap with 300 repetitions:

```
## compute model based bootstrap with 300 repetitions
# if this takes too long, reduce R to e.g. 50
ph.boot <- bootstrap(ph.geogam,
                     newdata = d.ph10.val,
                     R = 300)
```

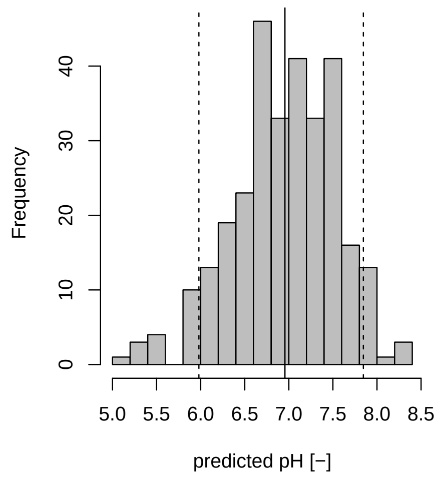

Figure 4: Histogram of predictive distribution for one single prediction point, computed by 300 bootstrap repetitions.

Plot for evaluation of prediction intervals (as shown in presentation):

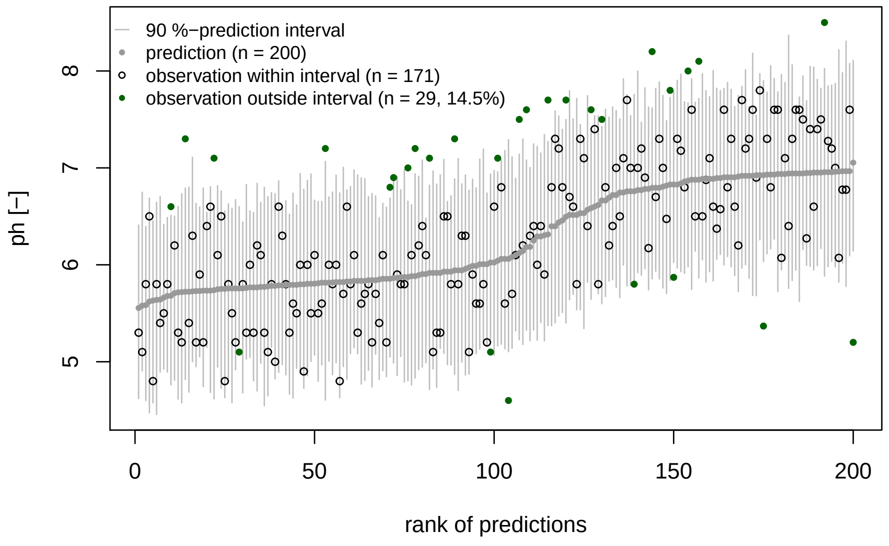

Figure 5: Coverage of 90-prediction intervals computed by model-based boostrap.

```r
# Coverage probabilities plot
# create sequence of nominal probabilities
ss <- seq(0,1,0.01)
# compute coverage for sequence
t.prop.inside <- sapply(ss, function(ii){
  boot.quantile <-  t( apply(ph.boot, 1, quantile, probs = c(0,ii) ) )[,2]
  return( sum(boot.quantile <= d.ph10.val$ph.0.10)/nrow(d.ph10.val) )
})

plot(x = ss, y = t.prop.inside[length(ss):1],
     type = "l", asp = 1,
     ylab = "coverage probabilities",
     xlab="nominal probabilities" )
# add 1:1-line
abline(0,1, lty = 2, col = "grey60")
# add lines of the two-sided 90 %-prediction interval
abline(v = c(0.05, 0.95), lty = "dotted", col = "grey20")
```

Please continue:

- Are you satisfied with the prediction intervals?
- Compute prediction intervals for random forest (Package `quantregForest`) and create the same plots. How do the intervals of quantile regression forest perform compared to the validation data?
- Create maps of the prediction intervals for the `berne.grid` data. Is there much spatial structure in the uncertainty? Do quantile regression forest and bootstrapped intervals differ? If yes, why (which assumption)?

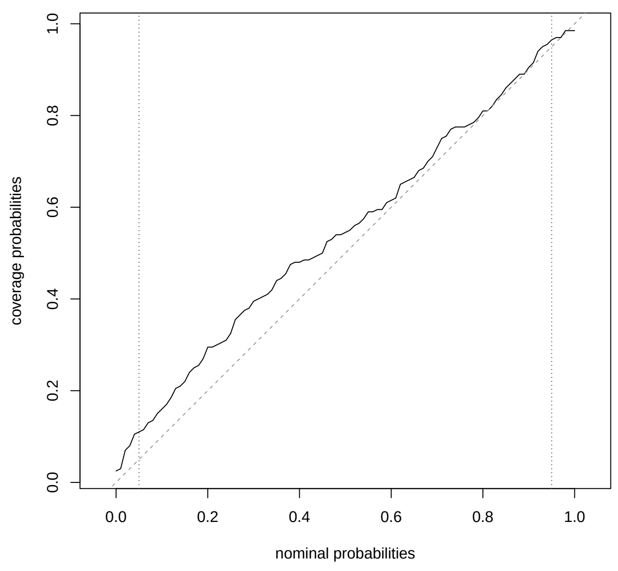

Figure 6: Coverage probabilities of one-sided prediction intervals computed for the validation data set of topsoil pH of the Berne study area.

R session information

This document was generated with:

```
toLatex(sessionInfo(), locale = FALSE)
```

- R version 3.4.1 (2017-06-30), `x86_64-pc-linux-gnu`
- Running under: `Progress Linux 4+ (dschinn-backports)`
- Matrix products: default
- BLAS: `/usr/lib/libblas/libblas.so.3.7.0`
- LAPACK: `/usr/lib/lapack/liblapack.so.3.7.0`
- Base packages: base, datasets, graphics, grDevices, methods, stats, utils
- Other packages: geoGAM 0.1-1, knitr 1.16, randomForest 4.6-12
- Loaded via a namespace (and not attached): codetools 0.2-15, coin 1.2-0, compiler 3.4.1, digest 0.6.12, evaluate 0.10.1, grid 3.4.1, grpreg 3.1-2, highr 0.6, lattice 0.20-34, magrittr 1.5, MASS 7.3-47, Matrix 1.2-7.1, mboost 2.8-0, mgcv 1.8-17, modeltools 0.2-21, multcomp 1.4-6, mvtnorm 1.0-6, nlme 3.1-129, nnls 1.4, parallel 3.4.1, party 1.2-3, quadprog 1.5-5, sandwich 2.3-4, splines 3.4.1, stabs 0.6-2, stats4 3.4.1, stringi 1.1.5, stringr 1.2.0, strucchange 1.5-1, survival 2.40-1, TH.data 1.0-8, tools 3.4.1, zoo 1.8-0