Bern University
of Applied Sciences

OpenGeoHub Summer School

Mastering Machine Learning for Spatial Prediction I

Practical training

Madlene Nussbaum, 4/5 September 2019

Contents

Preparation

Load needed packages:

```r
library(grpreg) # for grouped lasso
library(glmnet) # for general lasso
library(kernlab) # for support vector machines
library(caret) # for model tuning
library(randomForest) # to fit random forest
library(mboost) # for the boosting models with linear and spline terms
library(gbm) # for the boosting model with trees
library(caret) # for the tuning with train
library(geoGAM) # for the berne dataset
library(raster) # for plotting as a raster
library(parallel) # for parallel computing
```

As an example you can work with the Berne soil mapping study area: dataset `berne` in R package `geoGAM`, contains continuous, binary and a multinomial/ordered response and a spatial data `berne.grid` for prediction.

Feel free to work with your own data!

Hint: The processing of this code is quite time consuming on a laptop. Normaly, one uses high performance computing facilities for machine learning.

Load 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)]), ]
# select validation data for subsequent validation
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)]), ]
# Binary response
d.wlog100 <- berne[berne$dataset=="calibration"&!is.na(berne$waterlog.100), ]
d.wlog100 <- d.wlog100[complete.cases(d.wlog100[13:ncol(d.wlog100)]), ]
# Ordered/multinomial tesponse
d.drain <- berne[berne$dataset == "calibration" & !is.na(berne$dclass), ]
d.drain <- d.drain[complete.cases(d.drain[13:ncol(d.drain)]), ]
# covariates start at col 13
l.covar <- names(d.ph10[, 13:ncol(d.ph10)])
```

1 Lasso – linear shrinkage method

Lasso for continuous response

The `berne` dataset contains categorical covariates (factors, e.g. geological map with different substrate classes). The group lasso (R package `grpreg`) ensures that all dummy covariates of one factor are excluded (coefficients set to 0) together or remain in the model as a group.

The main tuning parameter λ is selected by cross validation. λ determines the degree of shrinkage that is applied to the coefficients.

HINT for R newbies: the `apply`-functions in R are replacements for loops (`sapply`: loop over a sequence of numbers, `lapply`: loop over a list). Compared to `for`, an `apply` is much faster and general coding style, though a bit more tricky to program.

Example, how to replace a `for` by a `sapply`:

```
# loop
# first create a vector to save the results
t.result <- c()
for( ii in 1:10){ t.result <- ii^2}
# the same as apply
t.result <- sapply(1:10, function(ii){ ii^2 })
# of course, this example is even shorter using:
t.result <- 1:10^2
```

Now we create the setup using `apply` and fit the grouped lasso:

```
# define groups: dummy coding of a factor is treated as group
# find factors
l.factors <- names(d.ph10[l.covar])[
  t.f <- unlist( lapply(d.ph10[l.covar], is.factor) ) ]
l.numeric <-  names(t.f[ !t.f ])

# create a vector that labels the groups with the same number
g.groups <- c( 1:length(l.numeric),
               unlist(
                 sapply(1:length(l.factors), function(n){
                   rep(n+length(l.numeric), nlevels(d.ph10[, l.factors[n]])-1)
                 })
               )
)
# grpreg needs model matrix as input
XX <- model.matrix( ~., d.ph10[, c(l.numeric, l.factors), F])[,-1]

# cross validation (CV) to find lambda
ph.cvfit <- cv.grpreg(X = XX, y = d.ph10$ph.0.10,
                      group = g.groups,
                      penalty = "grLasso",
                      returnY = T) # access CV results
```

Compute predictions for the validation set with optimal number of groups chosen by lasso:

```
# choose optimal lambda: CV minimum error + 1 SE (see glmnet)
l.se <- ph.cvfit$cvse[ ph.cvfit$min ] + ph.cvfit$cve[ ph.cvfit$min ]
idx.se <- min( which( ph.cvfit$cve < l.se ) ) - 1

# create model matrix for validation set
newXX <- model.matrix( ~., d.ph10.val[, c(l.factors, l.numeric), F])[,-1]
t.pred.val <-  predict(ph.cvfit, X = newXX,
                       type = "response",
                       lambda =  ph.cvfit$lambda[idx.se])
# get CV predictions, e.g. to compute R2
ph.lasso.cv.pred <- ph.cvfit$Y[,idx.se]
```

Get the lasso (non-zero) coefficients of the optimal model:

```
# get the non-zero coefficients:
t.coef <- ph.cvfit$fit$beta[, idx.se ]
t.coef[ t.coef > 0 ]

##                (Intercept)                 cl_mt_gh_4
##               1.5655968457               0.0064595963
##               tr_se_curv2m   tr_se_curvplan2m_std_10c
##               0.0006211146               0.0208871971
##   tr_se_curvplan2m_std_25c   tr_se_curvplan2m_std_50c
##               0.0163669489               0.0022316051
##                  tr_vdcn25           timesetd1979_2010
```

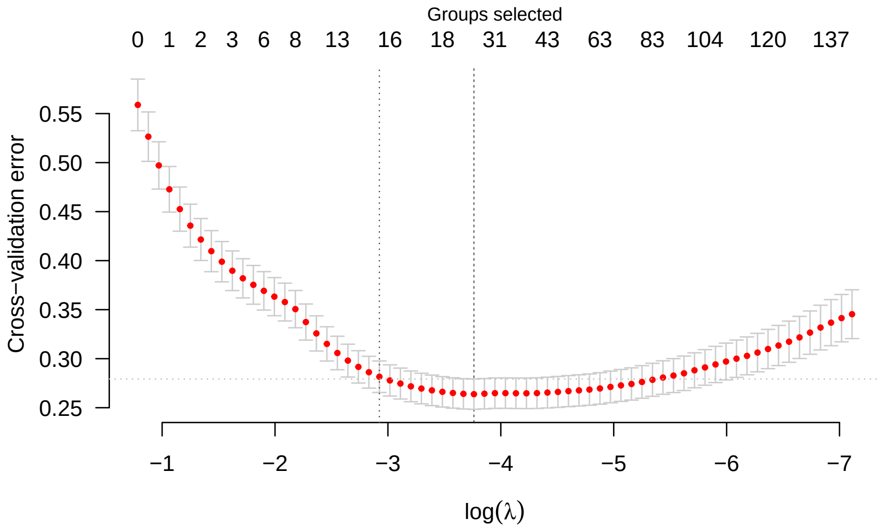

Figure 2: Cross validation error plotted against the tuning parameter lambda. The dashed line indicates lambda at minimal error, the dotted darkgrey line is the optimal lambda with minimal error + 1 SE.

```
##                0.0012151094                 0.1386393581
##    timesetd1968_1974_field ge_geo500h3idTorfbedeckung
##                0.2826352126                 0.1163145242
```

Lasso for multinomial response

I am not aware of a lasso implementation for multinomial responses that can handle groups of factors. Therefore, we use “standard” lasso from R package `glmnet` (the option `type.multinomial = "grouped"` does only ensure all coefficients of the multinomial model for the same covariate are treated as groups).

```
# create model matrix for drainage classes
# use a subset of covariates only, because model optimization for
# multinomial takes long otherwise

set.seed(42) # makes sample() reproducible
XX <- model.matrix(~.,d.drain[, l.covar[sample(1:length(l.covar), 30)]])[,-1]

drain.cvfit <- cv.glmnet( XX, d.drain$dclass, nfold = 10,
                          keep = T, # access CV results
                          family = "multinomial",
                          type.multinomial = "grouped")
```

For getting the coefficients of the final model you run the `glmnet` function again with the selected λ. Please note: The multinomial fit results in a coefficient for each covariate and response level.

```
drain.fit <- glmnet( XX, d.drain$dclass,
                     family = "multinomial",
                     type.multinomial = "grouped",
                     lambda = drain.cvfit$lambda.min)
# The coeffs are here:
# drain.fit£beta£well
# drain.fit£beta£moderate
# drain.fit£beta£poor
```

Please continue:

- Select the lasso for a binary response (e.g. presence/absence of waterlogging `waterlog.100`). Use `family = "binomial"` in `cv.grpreg` and make sure your response is coded as 0/1.
- For the multinomial lasso fit of drainage class: compute predictions for the validation set (`predict` with `s="lambda.1se"` or `s="lambda.min"`). Then, evaluate prediction accuracy by e.g. using Pierce Skill Score, see function `verify` or `multi.cont` in R package `verification`.

2 Support vector machines

We use support vector machines (SVM) for regression from the package `kernlab` with radial kernel basis functions that fit local relations in feature space. The tuning parameter `C` defines the flexibility of the SVM to allow for wrongly predicted data points and the parameter σ the degree of non-linearity of the radial kernel. Here we apply a two step approach to find optimal tuning parameters. `C` and σ of a first pass are used as starting points to find optimal parameters around the first estimates.

We tune the SVM with `caret` with a cross-validation. `caret` is a meta package that provides a homogenous interface to about 80 machine learning methods.

```
# We have to set up the design matrix ourselfs
# (without intercept, hence remove first column)
XX <- model.matrix( ~., d.ph10[, c(l.covar), F])[,-1]

# set seed for random numbers to split cross-valiation sets
set.seed(31)
# Setup for 10fold cross-validation
ctrl <- trainControl(method="cv",
                     number=10,
                     savePredictions = "final")

# 1. pass of training - find region of C and lambda
svm.tune1 <- train(x = XX,
                   y = d.ph10[, "ph.0.10"],
                   method = "svmRadial", # radial kernel function
                   tuneLength = 9, # check 9 values of the cost function
                   preProc = c("center","scale"), # center and scale data
                   trControl=ctrl)

# 2. pass of training - find best value for C and lambda
# setup a tuning grid with values around the result of the first pass
sig <- svm.tune1$bestTune$sigma
t.sigma <- sort( unique( round(abs( c(sig, sig + seq(0, sig*2, by = sig/1),
                                      sig - seq(0, sig*2, by = sig/1)) ), 6)))
tune.grid <- expand.grid(
  sigma = t.sigma[t.sigma>0], # sigma must be positive
  C = sort( unique( abs( c(svm.tune1$bestTune$C,
                           svm.tune1$bestTune$C - seq(0, 0.3, by = 0.1),
                           svm.tune1$bestTune$C + seq(0, 0.3, by = 0.1) )) ))
)
#Train and Tune the SVM
svm.model <- train(x = XX,
                   y = d.ph10[, "ph.0.10"],
                   method = "svmRadial",
                   preProc = c("center","scale"),
                   tuneGrid = tune.grid,
                   trControl = ctrl)

# -> if this takes too long: take a short cut with
# svm.model <- svm.tune1
```

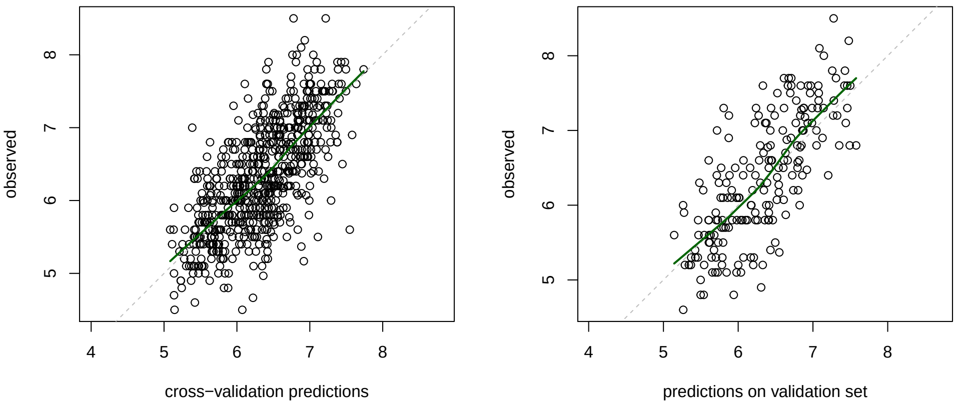

Figure 3: Predictions from cross-validation (left) and the validation dataset (right) plotted against the observed values (dashed: 1:1-line, green: lowess scatterplott smoother).

```
# create validation plots with lowess scatterplot smoothers
# for cross-validation
par(mfrow = c(1,2))
plot(svm.model$pred$pred, svm.model$pred$obs,
     xlab = "cross-validation predictions",
     ylab = "observed",
     asp = 1)
abline(0,1, lty = "dashed", col = "grey")
lines(lowess(svm.model$pred$pred, svm.model$pred$obs), col = "darkgreen", lwd = 2)

# for independent validation set
# calculate predictions for the validation set
newXX <- model.matrix( ~., d.ph10.val[, l.covar, F])[,-1]
t.pred.val <- predict.train(svm.model, newdata = newXX)
plot(t.pred.val, d.ph10.val[, "ph.0.10"],
     xlab = "predictions on validation set",
     ylab = "observed",
     asp = 1)
abline(0,1, lty = "dashed", col = "grey")
lines(lowess(t.pred.val, d.ph10.val[, "ph.0.10"]), col = "darkgreen", lwd = 2)
```

3 Random forest

Here we fit a random forest with the package `randomForest`. If you work with very large datasets consider using the package `ranger`. More important: use paralell computing as demonstrated here with the function `mclapply` (does not work on Windows, then use `mc.cores = 1`).

This is advanced programming with functions and `apply`. But, I wanted to show how you can do the tuning of a ML method yourself without using a meta-function like `train` from package `caret` (see below). Like this you can control it does what it should (and the code is not much longer)

```
# Fit a random forest with default parameters
# (often results are already quite good)
set.seed(1)
rf.model.basic <- randomForest(x = d.ph10[, l.covar ],
                               y = d.ph10[, "ph.0.10"])

# tune main tuning parameter "mtry"
# (the number of covariates that are randomly selected to try at each split)

# define function to use below
f.tune.randomforest <- function(test.mtry, # mtry to test
                                d.cal,     # calibration data
                                l.covariates ){ # list of covariates
  # set seed
  set.seed(1)
  # fit random forest with mtry = test.mtry
  rf.tune <- randomForest(x = d.ph10[, l.covar ],
                          y = d.ph10[, "ph.0.10"],
                          mtry = test.mtry)
  # return the mean squared error (mse) of this model fit
  return( tail(rf.tune$mse, n=1) )
}

# vector of mtry to test
seq.mtry <- c(1:(length(l.covar) - 1))
# Only take every fifth for speed reasons
seq.mtry <- seq.mtry[ seq.mtry %% 5 == 0 ]

# Apply function to sequence.
t.OOBe <-   mclapply(seq.mtry, # give sequence
                     FUN = f.tune.randomforest, # give function name
                     mc.cores = 1, ## number of CPUs
                     mc.set.seed = FALSE, # do not use new seed each time
                     # now here giv the arguments to the function:
                     d.cal = d.ph10,
                     l.covar = l.covar )

# Hint: Who is not comfortable with "mclapply"
# the same could be achieved with
# for(test.mtry in 1:m.end){
#    .. content of function + vector to collect result... }
# create a dataframe of the results
mtry.oob <- data.frame(mtry.n = seq.mtry, mtry.OOBe = unlist(t.OOBe))

# get the mtry with the minimum MSE
s.mtry <- mtry.oob$mtry.n[ which.min(mtry.oob$mtry.OOBe) ]

# compute random forest with optimal mtry
set.seed(1)
rf.model.tuned <- randomForest(x = d.ph10[, l.covar ],
                               y = d.ph10[, "ph.0.10"],
                               mtry = s.mtry)
```

```
plot( mtry.oob$mtry.n, mtry.oob$mtry.OOBe, pch = 4,
      ylab = "out-of-bag MSE error", xlab = "mtry")
abline(v = s.mtry, lty = "dashed", col = "darkgrey")
lines( lowess( mtry.oob$mtry.n, mtry.oob$mtry.OOBe ), lwd = 1.5, col = "darkgrey")
```

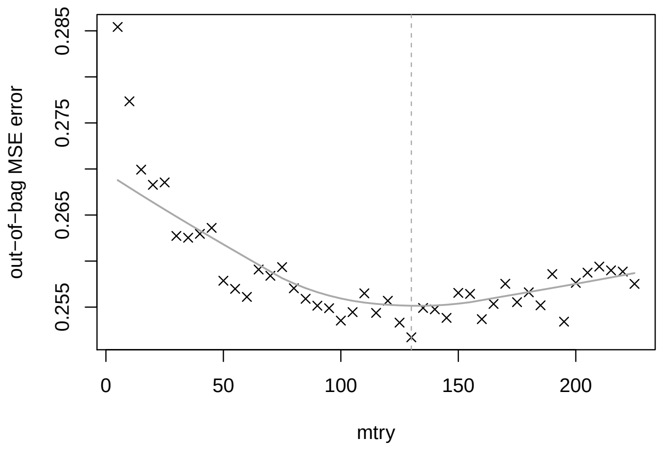

Figure 4: Tuning parameter mtry plotted against the out-of-bag mean squared error (grey line: lowess smoothing line, dashed line: mtry at minimum MSE).

Please continue:

- Compute the predictions for the validation set with the tuned and the model with default values (`predict(rf.model.tuned, newdata = ...)`) and compute a root mean squared error. Was the tuning effort worthwhile?
- Implement the same tunging with the package `caret`. Check the option `methofd="oob"` of `trainControl`. This function is handed to `train`.

4 Gradient boosting

4.1 Boosting with trees as baselearners

There are various R packages to fit boosting models (e.g. `mboost`, `xgboost`). We use `gbm` here. We can again tune it with `caret`. `caret` is a meta package that provides a homogenous interface to about 80 machine learning methods.

Fit gradient boosting with trees:

```
# create a grid of the tuning parameters to be tested,
# main tuning parameters are:
gbm.grid <- expand.grid(
  # how many splits does each tree have
  interaction.depth = c(2,5,10,15,20),
  # how many trees do we add (number of iterations of boosting algorithm)
  n.trees = seq(2,250, by = 5),
  # put the shrinkage factor to 0.1 (=10% updates as used
  # in package mboost), the default (0.1%) is a bit too small,
  # makes model selection too slow.
  # minimum number of observations per node can be left as is
  shrinkage = 0.1, n.minobsinnode = 10)

# make tuning reproducible (there are random samples for the cross validation)
set.seed(291201945)

# train the gbm model
gbm.model <- train(x=d.ph10[, l.covar ],
                   y=d.ph10[, "ph.0.10"],
                   method = "gbm", # choose "generalized boosted regression model"
                   tuneGrid = gbm.grid,
                   verbose = FALSE,
                   trControl = trainControl(
                     # use 10fold cross validation (CV)
                     method = "cv", number = 10,
                     # save fitted values (e.g. to calculate RMSE of the CV)
                     savePredictions = "final"))

# print optimal tuning parameter
gbm.model$bestTune

##    n.trees interaction.depth shrinkage n.minobsinnode
## 11      52                 2       0.1             10
```

```
# compute predictions for the small part of the study area
# (agricultural land, the empty pixels are streets, forests etc.)
data("berne.grid")

berne.grid$pred <- predict.train(gbm.model, newdata = berne.grid )

# create a spatial object for a proper spatial plot
coordinates(berne.grid) <- ~x+y
```

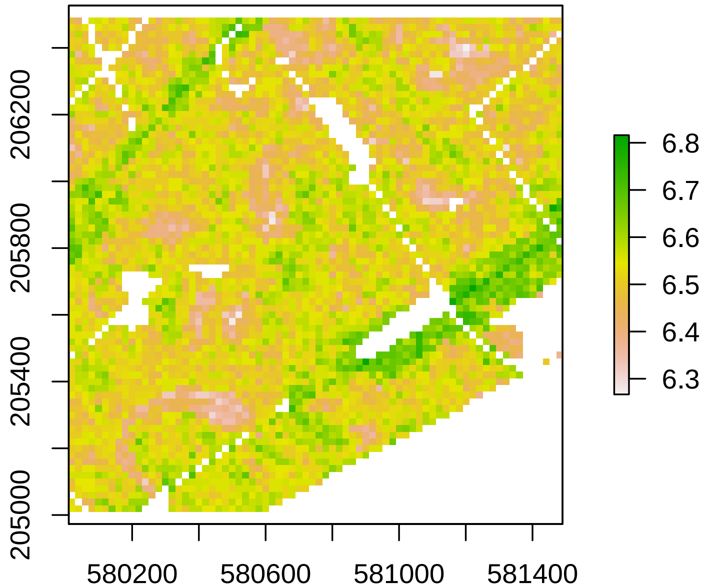

Figure 5: Predictions computed with an optimized boosted trees model of topsoil pH (0–10 cm) for a very small part of the Berne study region (white areas are streets, developped areas or forests, CRAN does not accept larger datasets).

```
# add the Swiss projection (see ?berne.grid)
# see https://epsg.io for details on projections
proj4string(berne.grid) <- CRS("+init=epsg:21781")
# create a raster object from the spatial point dataframe
gridded(berne.grid) <- TRUE
plot(raster(berne.grid, layer = "pred"))
```

Lets check the partial dependencies of the 4 most important covariates:

```
# get variable importance
t.imp <- varImp(gbm.model$finalModel)

# check how many covariates were never selected
sum( t.imp$Overall == 0 )

## [1] 168

# order and select 4 most important covariates
t.names <- dimnames(t.imp)[[1]][ order(t.imp$Overall, decreasing = T)[1:4] ]

par(mfrow = c(2,2))
for( name in t.names ){
  # select index of covariate
  ix <- which( gbm.model$finalModel$var.names == name )
  plot(gbm.model$finalModel, i.var = ix)
}

# -> improve the plots by using the same y-axis (e.g. ylim=c(..,..))
#    for all of them, and try to add labels (xlab = , ylab = )
#    or a title (main = )
```

4.2 Boosting with linear baselearners (advanced task)

Boosting algorithm can be used with any kind of base procedures / baselearners. Many packages (e.g. gbm, xgboost) use trees. Here we try linear and splines baselearners.

For details on mboost see the hands-on tutorial in the vignette to the package: https://cran.r-project.org/web/packages/mboost/vignettes/mboost_tutorial.pdf

Select a boosting model with linear baselearners (this results in shrunken coefficients, similar to the lasso, see Hastie et al. 2009):

```
# Fit model
ph.glmboost <- glmboost(ph.0.10 ~., data = d.ph10[ c("ph.0.10", l.covar)],
                        control = boost_control(mstop = 200),
                        center = TRUE)

# Find tuning parameter: mstop = number of boosting itertations
set.seed(42)
ph.glmboost.cv <- cvrisk(ph.glmboost,
                         folds = mboost::cv(model.weights(ph.glmboost),
                                            type = "kfold"))

# print optimal mstop
mstop(ph.glmboost.cv)

## [1] 160

## print model with fitted coefficents
# ph.glmboost[ mstop(ph.glmboost.cv)]
```

4.3 Boosting with splines baselearners (advanced task)

To model non-linear relationships we use splines baselearners. Spatial autocorrelation can be captured by adding a smooth spatial surface. This type of model needs a bit more setup. Each covariate type has its own specification. All baselearners should have the same degrees of freedom, otherwise biased model selection might be the result.

```
# quick set up formula

# Response
f.resp <- "ph.0.10 ~ "

# Intercept, add to dataframe
```

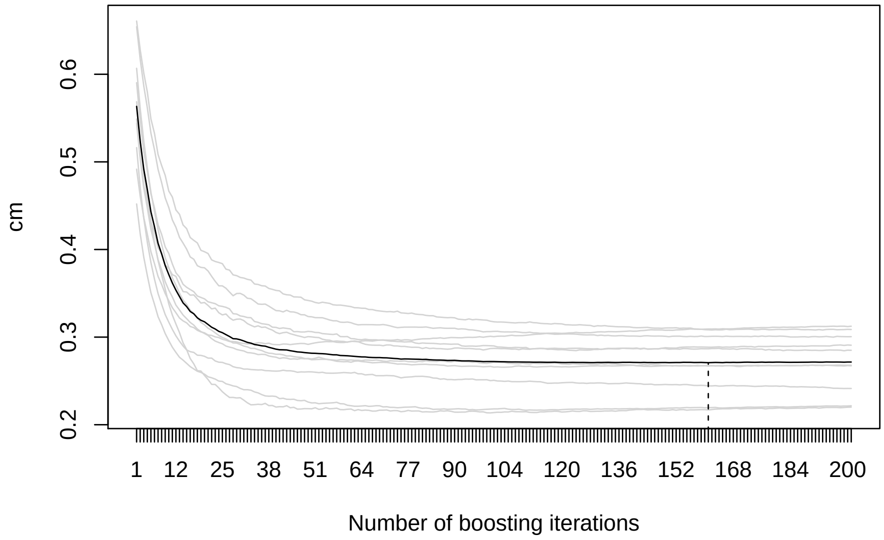

Figure 6: Path of cross validation error along the boosting iterations.

```r
f.int <- "bols(int, intercept = F, df = 1)"
d.ph10$int <- rep(1, nrow(d.ph10))

# Smooth spatial surface (needs > 4 degrees of freedom)
f.spat <- "bspatial(x, y, df = 5, knots = 12)"

# Linear baselearners for factors, maybe use df = 5
f.fact <- paste(
  paste( "bols(", l.factors, ", intercept = F)" ),
  collapse = "+"
)

# Splines baselearners for continuous covariates
f.num <- paste(
  paste( "bbs(", l.numeric, ", center = T, df = 5)" ),
  collapse = "+"
)

# create complete formula
ph.form <- as.formula( paste( f.resp,
                              paste( c(f.int, f.num, f.spat, f.fact),
                                     collapse = "+")) )
# fit the boosting model
ph.gamboost  <- gamboost(ph.form, data = d.ph10,
                         control = boost_control(mstop = 200))

# Find tuning parameter
ph.gamboost.cv <- cvrisk(ph.gamboost,
                         folds = mboost::cv(model.weights(ph.gamboost),
                                            type = "kfold"))
```

Analyse boosting model:

```
# print optimal mstop
mstop(ph.gamboost.cv)

## [1] 198

## print model info
ph.gamboost[ mstop(ph.glmboost.cv)]

##
##   Model-based Boosting
##
## Call:
## gamboost(formula = ph.form, data = d.ph10, control = boost_control(mstop = 200))
##
##
##   Squared Error (Regression)
##
## Loss function: (y - f)^2
##
##
## Number of boosting iterations: mstop = 160
## Step size:  0.1
## Offset:  6.314042
## Number of baselearners:  228

## print number of chosen baselearners
length( t.sel <-  summary( ph.gamboost[ mstop(ph.glmboost.cv)] )$selprob )

## [1] 47

# Most often selected were:
summary( ph.gamboost[ mstop(ph.glmboost.cv)] )$selprob[1:5]

##             bols(timeset, intercept = F)
##                                  0.10000
##      bbs(cl_mt_gh_3, df = 5, center = T)
##                                  0.05625
##      bbs(cl_mt_gh_8, df = 5, center = T)
##                                  0.05000
## bbs(tr_se_po2m_r500, df = 5, center = T)
##                                  0.04375
## bbs(tr_se_twi2m_s60, df = 5, center = T)
##                                  0.04375
```

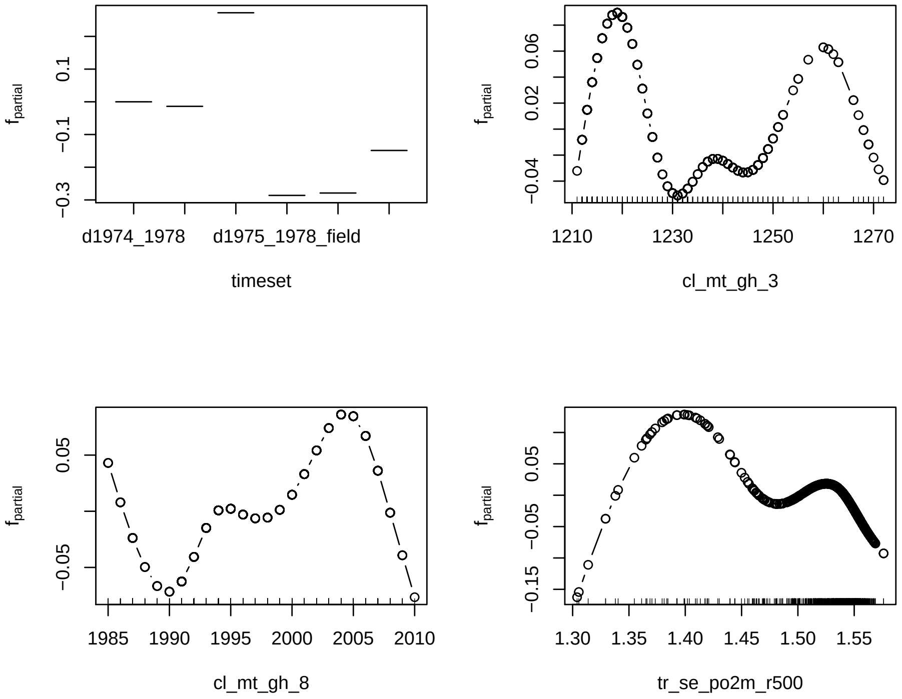

Figure 7: Residual plots of the 4 covariates with highest selection frequency.

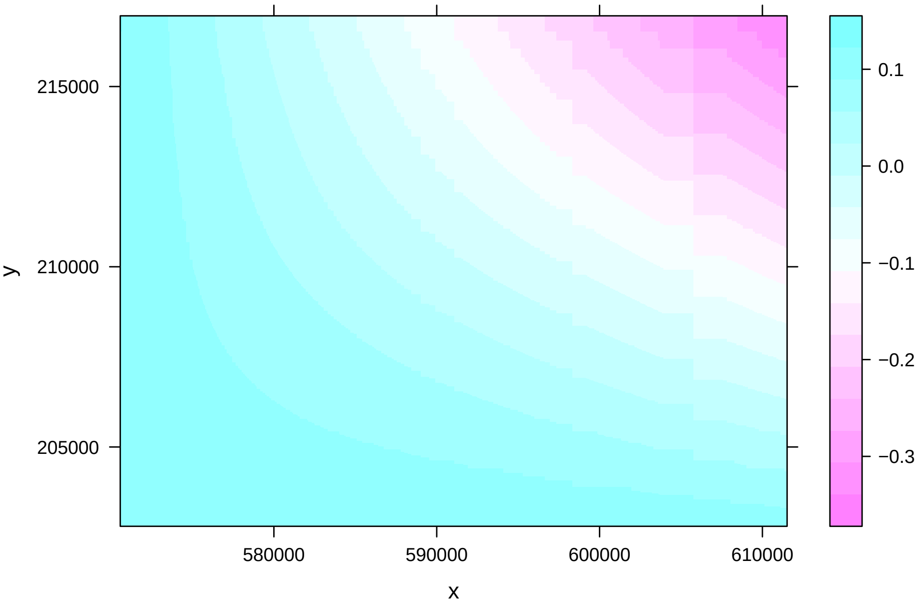

Figure 8: Modelled smooth spatial surface based on the coordinates.

5 Model averaging

So far we calibrated several models to predict topsoil pH. With model averaging we can combine these predictions computing a simple `mean`. Besides simple averaging, we could use weights like $\frac{1}{MSE}$ (make sure they sum up to 1).

Compute validation statistics (e.g. root mean squared error, R^2) on the validation set for the predictions of each model and the (weighted) averaged predictions. Is the prediction accuracy improved?

You could now add models computed from random forest, support vector machines or gradient boosted trees. Does this improve model accuracy?

Note: Be aware not to select the final model based on the validation data. If you start tuning your predictions on your validation data, you loose the independent estimate of prediction accuracy... better choose your method for the final predictions based on cross validation (e.g. on the same sets).

R session information

This document was generated with:

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

- R version 3.5.0 (2018-04-23), `x86_64-pc-linux-gnu`
- Running under: `Progress Linux 4+ (dschinn-backports)`
- Matrix products: default
- BLAS: `/usr/lib/openblas-base/libblas.so.3`
- LAPACK: `/usr/lib/libopenblasp-r0.2.19.so`
- Base packages: base, datasets, graphics, grDevices, methods, parallel, stats, utils
- Other packages: caret 6.0-81, foreach 1.4.4, gbm 2.1.4, geoGAM 0.1-2, ggplot2 3.1.1, glmnet 2.0-16, grpreg 3.2-0, kernlab 0.9-27, knitr 1.20, lattice 0.20-38, Matrix 1.2-15, mboost 2.9-1, quantregForest 1.3-7, randomForest 4.6-14, raster 2.8-4, RColorBrewer 1.1-2, sp 1.3-1, stabs 0.6-3
- Loaded via a namespace (and not attached): assertthat 0.2.1, bindr 0.1.1, bindrcpp 0.2.2, class 7.3-14, codetools 0.2-15, colorspace 1.4-1, compiler 3.5.0, crayon 1.3.4, data.table 1.11.8, digest 0.6.18, dplyr 0.7.8, evaluate 0.12, Formula 1.2-3, generics 0.0.1, glue 1.3.1, gower 0.1.2, grid 3.5.0, gridExtra 2.3, gtable 0.3.0, highr 0.7, inum 1.0-0, ipred 0.9-8, iterators 1.0.10, lava 1.6.4, lazyeval 0.2.2, libcoin 1.0-1, lubridate 1.7.4, magrittr 1.5, MASS 7.3-51.1, mgcv 1.8-25, ModelMetrics 1.2.2, munsell 0.5.0, mvtnorm 1.0-8, nlme 3.1-137, nnet 7.3-12, nnls 1.4, partykit 1.2-2, pillar 1.3.1, pkgconfig 2.0.2, plyr 1.8.4, prodlim 2018.04.18, purrr 0.2.5, quadprog 1.5-5, R6 2.4.0, Rcpp 1.0.1, recipes 0.1.4, reshape2 1.4.3, rgdal 1.3-6, rlang 0.3.4, rpart 4.1-13, rstudioapi 0.8, scales 1.0.0, splines 3.5.0, stats4 3.5.0, stringi 1.4.3, stringr 1.4.0, survival 2.43-1, tibble 2.1.1, tidyselect 0.2.5, timeDate 3043.102, tools 3.5.0, withr 2.1.2