GEOSTAT
Mastering Machine Learning for Spatial Prediction 1
Exercises

Madlene Nussbaum

13 August 2017

Contents

Preparation

Load needed packages:

```
library(grpreg)
library(glmnet)
library(randomForest)
library(mboost)
library(geoGAM)
library(raster)
```

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!

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)]), ]
# 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 response
d.drain <- berne[berne$dataset == "calibration" & !is.na(berne$dclass), ]
```

```r
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

The `berne` dataset contains categorical covariates (factors). The group lasso (R package `grpreg` ensures that all dummy covariates of one factor are excluded (coefficients set to 0) togheter or remain in the model together.)

The main tuning parameter λ is selected by cross validation.

```r
# 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 ])

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.factors, l.numeric), 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:

```r
# 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

# 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)]), ]

# 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])
```

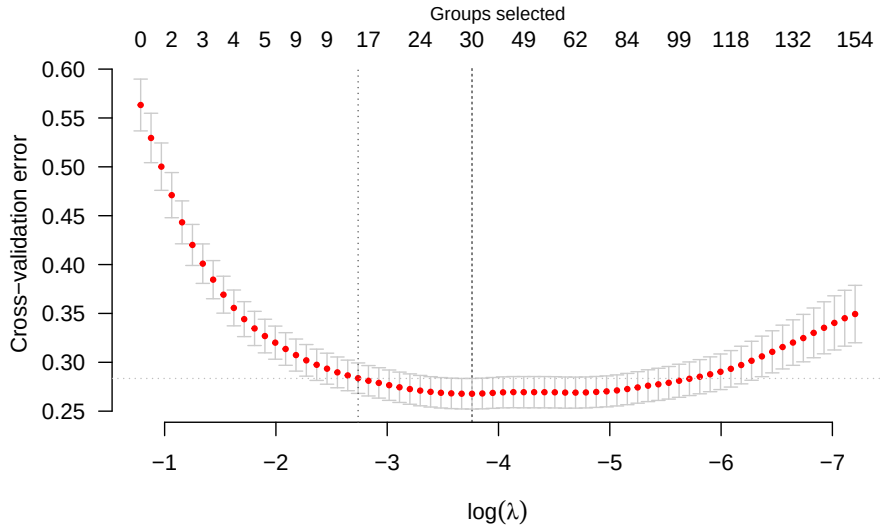

Figure 1: Cross validation error plotted against the tuning parameter lambda. The dashed line indicates lambda at minimal error, the dotted darkgrey line is the optimal lambda with minimal error + 1 SE.

```
# get CV predictions, e.g. to compute R2
ph.lasso.cv.pred <- ph.cvfit$Y[,idx.se]
```

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 coefficents 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)
XX <- model.matrix(~.,d.drain[, l.covar[sample(1:length(l.covar), 20)]])[,-1]

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

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 Gradient boosting

2.1 Boosting with linear baselearners

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)]
```

2.2 Boosting with splines baselearners

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
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
```

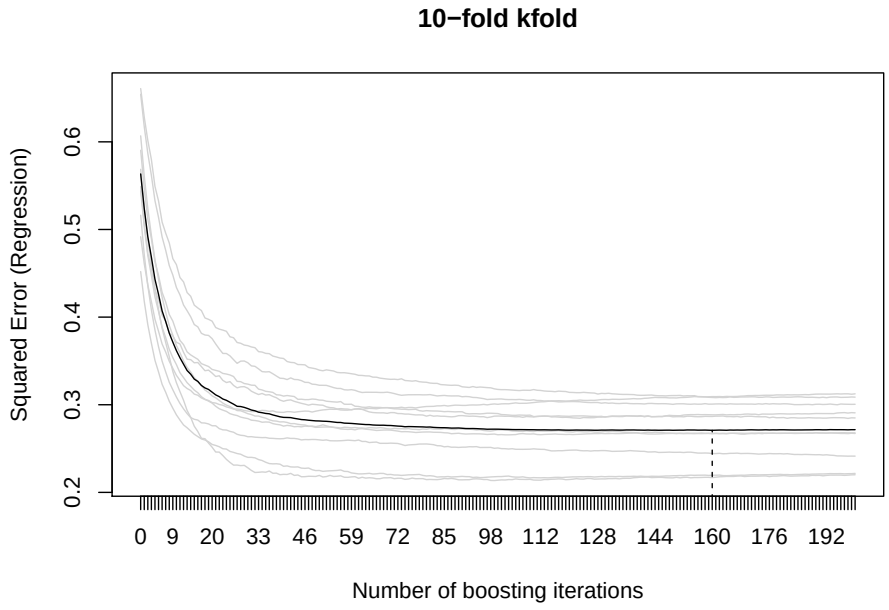

Figure 2: Path of cross validation error along the boosting iterations.

```
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
```

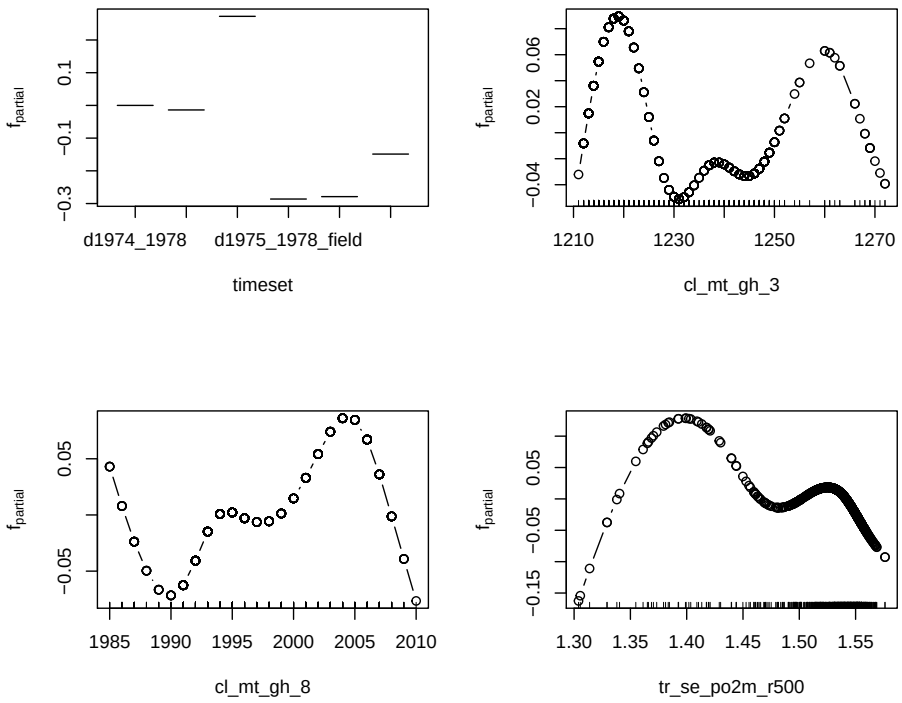

Figure 3: Residual plots of the 4 covariates with highest selection frequency.

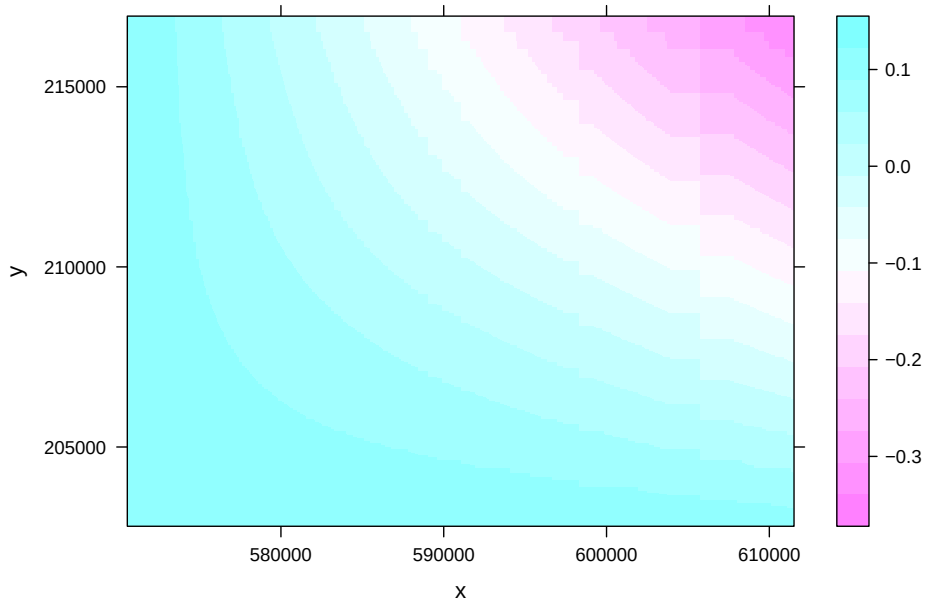

Figure 4: Modelled smooth spatial surface based on the coordinates.

2.3 Geoadditive model selection using boosting

Gradient boosting with splines results still in quite large covariate sets. Moreover, factors often do not have the same degrees of freedom as the other baseleaners (here set to `df = 5`). Selection of factors might be biased.

GeoGAM ueses boosting to select covariates and their structure to finally fit a (geo)additive model (`gam` object from R package `mgcv`). "geo" refers to the smooth spatial surface that is possibly selected if spatial correlation is relevant (judged by decrease in cross validation error). Factors are added as offset for boosting and – if relevant – added to the final model.

NOTE: geoGAM is alpha / experimental version. Bug reports to: madlene.nussbaum@bfh.ch.

```
ph.geogam <- geoGAM(response = "ph.0.10",
                    covariates = l.covar,
                    coords = c("x", "y"),
                    data = d.ph10, seed = 1)
```

Formula of chosen model:

```
ph.geogam$gam.final$formula

## ph.0.10 ~ ge_lgm + sl_retention_fil + s(tr_be_twi2m_s60_tcilow,
##     bs = "ps", k = 16, m = c(3, 2)) + s(tr_se_alti10m_c, bs = "ps",
##     k = 16, m = c(3, 2)) + timesetag + ge_caco3ag + ge_gt_ch_filag +
##     sl_skelett_r_filag

##  plot summary and model selection information
# summary(ph.geogam)
# summary(ph.geogam, what = "path")
```

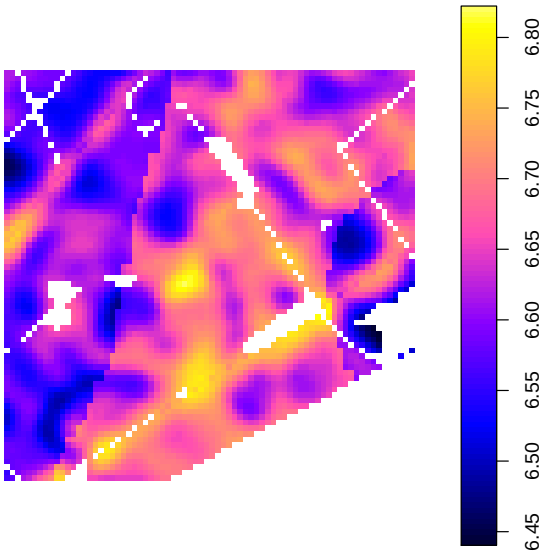

Figure 5: GeogAM predictions for topsoil pH for the berne.grid section of the study area.

3 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 3.4.1 (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, parallel, stats, utils
- Other packages: foreach 1.4.3, geoGAM 0.1-1, glmnet 2.0-10, grpreg 3.1-2, knitr 1.16, Matrix 1.2-7.1, mboost 2.8-0, randomForest 4.6-12, raster 2.5-8, sp 1.2-4, stabs 0.6-2
- 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, highr 0.6, iterators 1.0.8, lattice 0.20-34, magrittr 1.5, MASS 7.3-47, mgcv 1.8-17, modeltools 0.2-21, multcomp 1.4-6, mvtnorm 1.0-6, nlme 3.1-129, nnls 1.4, party 1.2-3, quadprog 1.5-5, Rcpp 0.12.11, rgdal 1.2-7, sandwich 2.3-4, splines 3.4.1, 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