Bern University
of Applied Sciences

ISRIC
World Soil Information

ISRIC spring school – Hands on Digital Soil Mapping

Machine learning 2:
Understanding methods, model selection and interpretation

Solutions of the practical training

Madlene Nussbaum, 31 May 2018

Contents

1 Lasso – linear shrinkage method

1. Check the formula in the presentation slides. For large λ the lasso penalty becomes large (strong shrinking). Hence, many covariates are removed from the model and the remaining covariates have small coefficients. For small λ the penalty is small and the model becomes larger (many covariates) with larger coefficients. For very small λ the model will be similar to ordinary least squares.

2. If you do not have categorical covariates (factors with dummy coding) in your data, you do not need to use group lasso (`grpreg`). You can use standard lasso from `glmnet` instead. Moreover, if you have multinomial data (a categorical response with more than one level) you have to use standard lasso. As far as I am aware, there is no software that can fit a group lasso (maybe in the future).

3. The `model.matrix()` expands all the categorical covariates (factors) to dummy coding. For each such covariate columns are added (number of added columns = number of levels minus 1). Each column contains a 1 if this level is true for an observation or a 0 if it is

not true. Most R functions use `model.matrix()` in the background and do not require you to define it. Mostly, you just give a formula (e.g. $y \sim covar1 + covar2$). The lasso functions want to provide more flexible input possibilities, so they have another way of model specification.

4. The interpretation is exactly the same. Just the estimation of the coefficients is done by different rules. Positive coefficients mean a positive relationship with the response, negative coefficients a negative correlation with the response. Please note: coefficients of one single covariate depend on the other covariates in the model. The sign and the value of a coefficient is always "given the current model".

2 Selection of covariates by random forest

1. These covariates are monthly rainfall patterns and are correlated. At each split of the trees only a random subset of `mtry` (in the full fit `mtry` = 75) are tried and the best one is selected. If one of the `cl_mt_rr`-covariates is excluded and the other is included in the random subset the latter can easily substitute the first. Correlated covariates are therefore often simultaneously rated high importance.

2. The OOB error is calculated as the mean square error. If you compute the square root (`sqrt()`) you get the original units of the data (pH). So the model selection improved the OOB root mean square error from pH 0.505 to pH 0.491. This is only a very minor improvement and most likely not worth the effort. The complete study Berne area is quite big (see Nussbaum et al. 2018 in lecture), hence computing predictions with 226 covariates is more time consuming than with 60 only. Therefore, it was still useful to reduce the covariates.

3. Considering Figure 4 (path of OOB errors along the covariate selection) you could maybe exclude even more covariates. There is one model fit with similarly low OOB error, but with less covariates.

4. Compared to the non-zero coefficients do you find the same covariates?

   In the random forest model remain more covariates after model selection. The covariates selected by lasso are however not a subset of the ones selected by random forest (2 covariates are selected by lasso, but not by random forest).

   Why are there differences?

   Lasso and random forest model different structures in the data. Lasso models linear relationships without considering non-linearities and without considering interactions between the covariates. Tree based methods as random forest practically only model interactions (splitting on covariate A then on covariate B models an interaction between A and B; an interaction is a complexer kind of relationship where the value of the response are modelled by two interdependent covariates: e.g. pH is low in valley bottoms on geological substrate A, but not on substrate B).

How would you explain these differences to a soil surveyor without in depth statistical knowledge?

Very difficult! I have problems doing that... Model selection becomes a bit arbitrary for large sets of covariates and for a response where we expect rather week models (low R^2) as this is the case for the `berne` dataset. Therefore, please do not over-interpret the selected covariates. But, checking pedological feasibility with a soil surveyor is always a good idea. Maybe you did an error in the data preparation and you modelled impossible values.

3 Model interpretation

1. The coefficient of the OLS fit is larger (0.0193) than the lasso fit (0.0062). The OLS was not fitted with the same covariates and the lasso shrinks the coefficients (makes them smaller depending on the λ penalty).

2. Check the y-axis: `cl_mt_rr_3` splits the pH values between 6.19 to 6.47. `cl_mt_rr_y` only splits pH values between 6.28 to 6.37.

3. For `cl_mt_rr_3` the split is mostly done on a rainfall of 77 mm, for `cl_mt_rr_11` on 78 mm and for the yearly rainfall `cl_mt_rr_y` on 1 050 mm. This means predictions are calculated mostly with a binary predictor map showing rainfall above and below these values.

4. Partial residual plots: good covariates/predictors show very steep regression lines (red in the figure). Moreover, the residual points follow this red line closely. In Figure 5 the residual points are quite scattered and the regression line is not very steep, so the covariates perform "medium" to predict the pH. For a very bad predictor, the red line would follow the dashed horizontal line drawn at 0.

   Partial dependence plot: good covariates split over the whole range of values of the response and the covariate. The plot should range from pH 4.5 to 8.5 (y-axis). The graph should not drop immediately, but rather on a continuous rate as we increase the values of the covariate (x-axis). Here the splits in the trees are mainly done on the same rainfall value, more desirable would be a wide range of rainfall values. Hence again, we have medium predictors.

4 Advanced tasks

Disclaimer: run the code here at your own risk, some of it might take a long time.

Fit lasso to binary data

Select the lasso for a binary response (e.g. presence/absence of waterlogging `waterlog.100`):

```
library(grpreg) # grouped lasso
library(geoGAM) # for the berne dataset
# Binary response, presence of waterlogging down to 100 cm (yes/no)
data(berne)
d.wlog100 <- berne[berne$dataset=="calibration"&!is.na(berne$waterlog.100), ]
d.wlog100 <- d.wlog100[complete.cases(d.wlog100[13:ncol(d.wlog100)]), ]
l.covar <- names(d.wlog100[, 13:ncol(d.wlog100)])

# define groups (the same code as for topsoil ph)
l.factors <- names(d.wlog100[l.covar])[
  t.f <- unlist( lapply(d.wlog100[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.wlog100[, l.factors[n]])-1)
                 })
               )
)
# grpreg needs model matrix as input
XX <- model.matrix( ~., d.wlog100[, c(l.numeric, l.factors), F])[,-1]
# cross validation (CV) to find lambda
wlog.cvfit <- cv.grpreg(X = XX, y = d.wlog100$waterlog.100,
                        group = g.groups,
                        penalty = "grLasso",
                        returnY = T) # access CV results
plot(wlog.cvfit) # now, continue as with the topsoil pH example
```

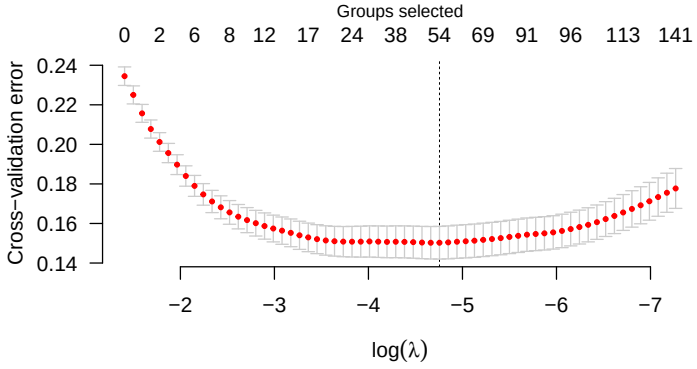

Figure 2: Cross validation results for lasso fit on binary response.

Better understand R coding

Replace the `apply`'s in the training by `for`-loops:

```
library(geoGAM) # for the berne dataset

data(berne)
d.ph10 <- berne[berne$dataset == "calibration" & !is.na(berne$ph.0.10), ]
l.covar <- names(d.ph10[, 13:ncol(d.ph10)])

## 1. apply-function
## version 1
l.factors <- c()
for( ii in l.covar){
  # skip if it is not a factor
  # (the loop is not further executed)
  if( !is.factor(d.ph10[, ii]) ) next

  # append to vector
  l.factors <- c(l.factors, ii)
}

## version 2
l.factors <- c()
for( ii in l.covar){
  # append to vector, if index on ii is getting TRUE
  l.factors <- c(l.factors, ii[is.factor(d.ph10[, ii] ) ] )
}
```

```
## 2. apply-function
l.numeric <-  names(t.f[ !t.f ])
g.groups <- 1:length(l.numeric)

for(ll in 1:length(l.factors)){
  # append number of levels of each factor to groups vector
  g.groups <- c(g.groups,
        rep(ll+length(l.numeric), nlevels(d.ph10[, l.factors[ll]])-1 ))

}
```

Replace the `for` loop in the recursive backward eliminationby an apply:

Note: this is advanced stuff! If you do not understand everything, please do not get depressed.

```
library(randomForest) # for random forest
library(geoGAM) # to get the Berne data set

data(berne)
# continuous response, topsoil pH in 0-10 cm
d.ph10 <- berne[berne$dataset == "calibration" & !is.na(berne$ph.0.10), ]
d.ph10 <- d.ph10[complete.cases(d.ph10[13:ncol(d.ph10)]), ]
l.covar <- names(d.ph10[, 13:ncol(d.ph10)])
```

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

# simple example for the Fibonacci series
f.fibo <- function(n) {

  if ( n < 2 ) {
    n
  } else {
    f.fibo(n-1) + f.fibo(n-2)
  }
}
t.fibo <- sapply(0:26, f.fibo)


## now: backward elimination with a sapply
# create the sequence as before
s.seq <- sort( c( seq(5, 95, by = 5),
                  seq(100, length(l.covar), by = 10) ),
               decreasing = T)

# only to parts of the calculation to save time
s.seq <- s.seq[1:5]

# at one more number at the beginning because we add the starting
# model within the function
s.seq <- c(1,s.seq)

## first create a function, that we can pass to sapply
# start.model = rf.ph -> hand over the first model to the function
# (would also work without, but the namespace in a function is a
# different one than outside of it, this makes sure the model is found
# inside the function)
f.back.elim <- function(ss, start.model = rf.ph){

  set.seed(1) # make random forest reproducible
  # if you want to compare with the loop, refit with calling the same seed before

  if( ss == 1){
    # for the first iteration collect the starting model in a list and return
    return( list( qrf.elim = start.model, oob.mse = tail(start.model$mse, n=1) ))

  } else {

    # now just do the same as in the loop
    t.imp <- importance( f.back.elim(ss-1)[[1]] )
    t.imp <- t.imp[ order(t.imp[,1], decreasing = T),]

    rf.fit <- randomForest(x = d.ph10[, names(t.imp[1:s.seq[ss]])],
                           y = d.ph10$ph.0.10 )
```

```
    # collect the results in a list and return
    return( list( qrf.elim = rf.fit, oob.mse = tail(rf.fit$mse,n=1) ))
  }
}

# apply this function to the sequence
# do not simplify the results, so we can access it with the given names later
result.list <- sapply( 1:length(s.seq), FUN = f.back.elim, simplify = FALSE)

# the random forest model is then stored in:
result.list[[2]]$qrf.elim

##
## Call:
##  randomForest(x = d.ph10[, names(t.imp[1:s.seq[ss]])], y = d.ph10$ph.0.10)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 73
##
##           Mean of squared residuals: 0.2519095
##                     % Var explained: 55.07

# and the OOB error is stored in:
result.list[[2]]$oob.mse

## [1] 0.2519095
```

Optimize random forest before covariate selection

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?

```
library(randomForest) # for random forest
library(geoGAM) # to get the Berne data set
library(caret) # for the tuning with train

data(berne)
# continuous response, topsoil pH in 0-10 cm
d.ph10 <- berne[berne$dataset == "calibration" & !is.na(berne$ph.0.10), ]
d.ph10 <- d.ph10[complete.cases(d.ph10[13:ncol(d.ph10)]), ]
l.covar <- names(d.ph10[, 13:ncol(d.ph10)])

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

# create grid of all possible mtry (maximum = number of covariates)
# rf.grid <- expand.grid(.mtry = 1:length(l.covar))
# only test part of it, otherwise too slow..
rf.grid <- expand.grid(.mtry = seq(5, length(l.covar), by = 50))

# do tuning with caret based on cross validation
rf.ph.tuned <- train(x = d.ph10[, l.covar],
                     y = d.ph10$ph.0.10,
                     method="rf",
                     tuneGrid = rf.grid,
                     trControl = trainControl(method = "cv",number = 10) )

# print result
rf.ph.tuned$bestTune

##   mtry
## 4  155

# -> hint: mtry could also be choosen by the OOB errors yielded from every
#    randomForest fit. As far as I am aware, this is not implemented in caret,
#    but can easily be done by a for loop (or apply function), see the code
#    of the recursive backward elimination
```

Gradient boosting with trees

There are various R packages to fit boosting models (e.g. `mboost`, `xgboost`). We use `gbm` here. We can tune it with `caret`. Fit gradient boosting with trees:

```
# load data an packages
library(gbm) # for the boosting model

## Error in library(gbm):  there is no package called 'gbm'

library(caret) # for the tuning with train

## Error in library(caret):  there is no package called 'caret'

library(geoGAM) # for the berne dataset
library(raster) # for plotting as a raster

## Error:  package 'sp' could not be loaded

data(berne)
d.ph10 <- berne[berne$dataset == "calibration" & !is.na(berne$ph.0.10), ]
d.ph10 <- d.ph10[complete.cases(d.ph10[13:ncol(d.ph10)]), ]
l.covar <- names(d.ph10[, 13:ncol(d.ph10)])
```

```
# 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"))
```

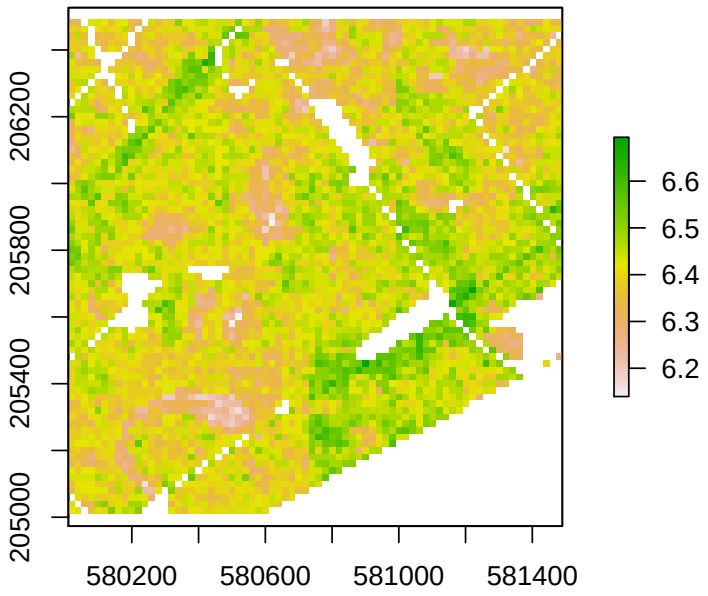

Figure 3: Predictions computed with an optimized boosted trees model of topsoil pH (0–10 cm) for a small part of the Berne study region (white areas are streets, developped areas or forests).

```
# 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
# add the Swiss projection (see ?berne.grid)
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)
```

```
## Error in varImp(gbm.model$finalModel):  could not find function "varImp"

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

## Error in eval(expr, envir, enclos):  object 't.imp' not found

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

## Error in eval(expr, envir, enclos):  object 't.imp' not found

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)
}

## Error in eval(expr, envir, enclos):  object 't.names' not found

# -> 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 = )
```

Automated documentation of R analysis

Mini example, put the following in an `.Rnw`-file and compile with `KnitR` in RStudio.

```
# Remove the #s
# \documentclass{article}
# \begin{document}
# \section{Hello World}

# Calculate some basic operation and print the result:

# <<junk-with-some-code>>=
# a <- 1
# b <- 2
# (a + b)^b
# @

# We use the result in the text; it was \Sexpr{(a + b)^b}.
# \bigskip

# We create a random plot:
# <<junk-with-a-silly-figure>>=
# plot(rnorm(10),pch=1:10,col=1:10)
# @

# \end{document}
```

R session information

```
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/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 0.1-2, grpreg 3.1-3, knitr 1.20, Matrix 1.2-14
- Loaded via a namespace (and not attached): codetools 0.2-15, coin 1.2-2, compiler 3.5.0, evaluate 0.10.1, grid 3.5.0, highr 0.6, lattice 0.20-35, magrittr 1.5, MASS 7.3-50, mboost 2.8-1, mgcv 1.8-23, modeltools 0.2-21, multcomp 1.4-8, mvtnorm 1.0-7, nlme 3.1-137, nnls 1.4, parallel 3.5.0, party 1.3-0, quadprog 1.5-5, sandwich 2.4-0, splines 3.5.0, stabs 0.6-3, stats4 3.5.0, stringi 1.2.2, stringr 1.2.0, strucchange 1.5-1, survival 2.42-3, TH.data 1.0-8, tools 3.5.0, zoo 1.8-1