Spatial Statistics

Computer lab – Session 3

Madlene Nussbaum

15 Oct 2024

Table of contents

1 Spatial prediction of binary response by random forest

1.1 Explore data

We will spatially predict landslide incidents in dataset `lsl` from R package `spDataLarge` (see excercieses *session 1*). The response is a binary variable `lslpts`. Further, five covariates derived from the elevation model are available.

> Task 1
>
> Investigate the dataset: Compute the percentage of each class. Is the dataset balanced?

Task 2

Create boxplots of response vs. covariates. Which relationships do you see?

Solution Task 1

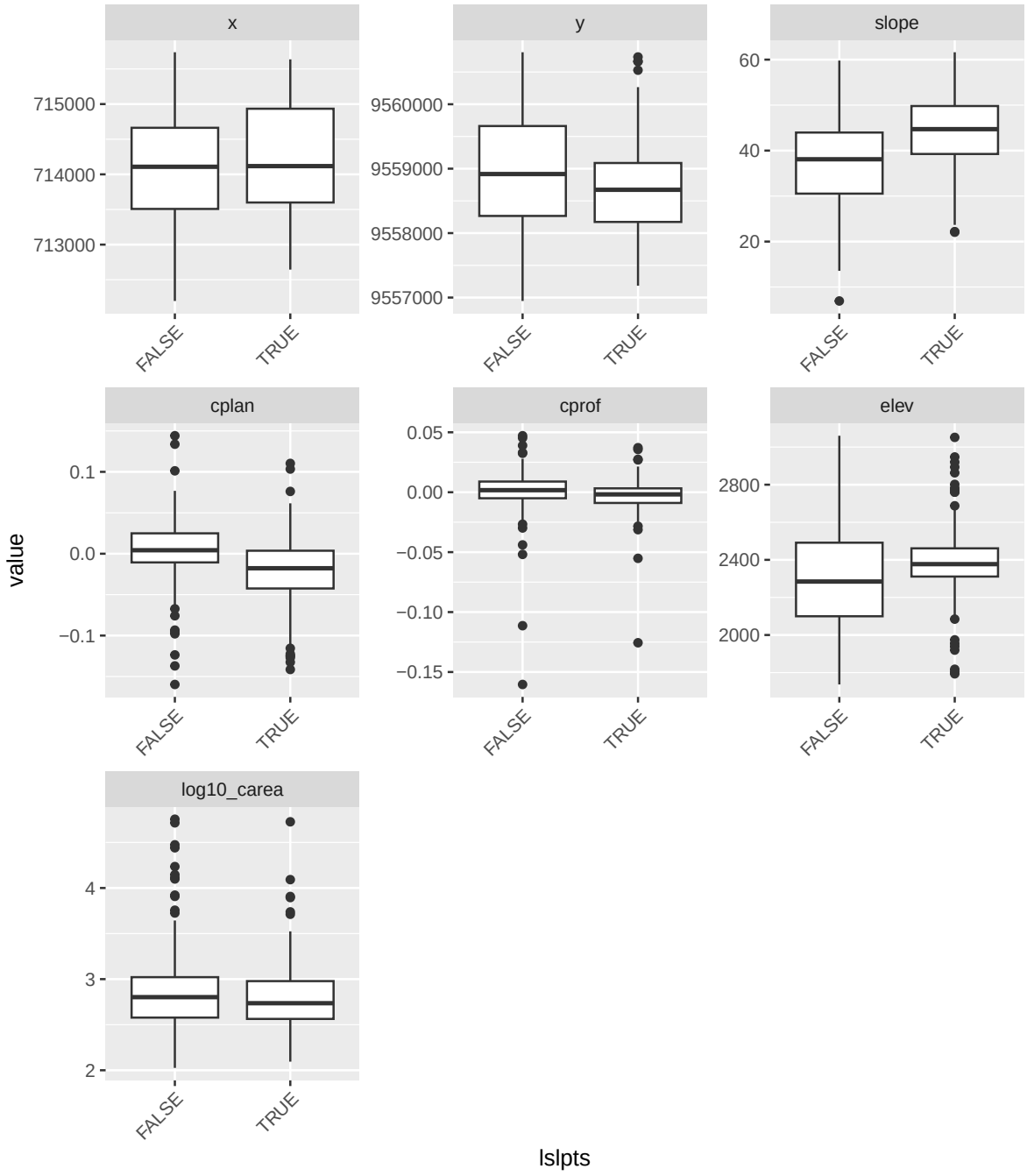
```
library(spDataLarge)
library(data.table)
data(lsl, package = "spDataLarge")
table(lsl$lslpts)/nrow(lsl)*100
```

```
FALSE  TRUE
   50    50
```

The response is exactly balanced. No steps have to be taken to account for uneven distribution of the response classes.

Solution Task 1

```
library(ggplot2)
# transform from wide to long table format
dt.lsl <- melt(data.table(lsl), id.vars = "lslpts")

par(mfrow = c(2,3))
ggplot(dt.lsl, aes(x = lslpts, y = value))+
  geom_boxplot()+
  facet_wrap(. ~variable, scales = "free")+
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



The value distributions for each class overlap for all of the covariates, but we do observe clear differences in some plots. While cprof (profile curvature of terrain) is about the same for both response classes, cplan (planar curvature) shows different median. Slope seems also to differentiate across the response classes well with landslides occurring in steeper terrains, also

at higher elevations. Along the coordinate axis (x, y) we do not see too much structure useful for prediction.

1.2 Fit random forest model

> Task 1
>
> Fit a random forest model using the terrain information as covariates (without spatial coordinates `x` and `y`).
> You can for example use package `ranger`. Use arguments `importance = "permutation"` to obtain permuted covariate importance and `probability = T` to obtain probability predictions for the next tasks.

> Task 2
>
> Evaluate covariate importance using the function `importance()`. Create a barplot for better visibility. Is the performance of the covariates as expected from the exploratory analysis?

Solution Task 1

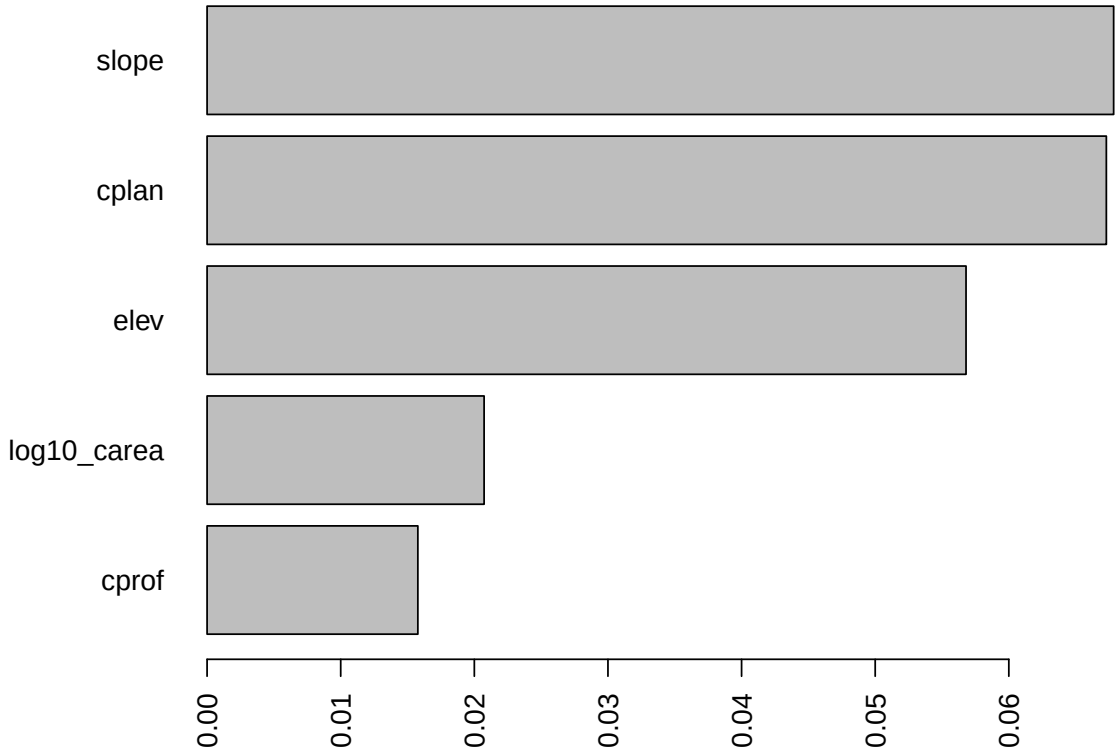
```
library(ranger)
rf.1 <- ranger(y = lsl$lslpts,
               x = lsl[4:8],
               importance = "permutation",
               probability = T, seed = 13)
rf.1
```

```
Ranger result

Call:
 ranger(y = lsl$lslpts, x = lsl[4:8], importance = "permutation",      probability = T, seed

Type:                             Probability estimation
Number of trees:                  500
Sample size:                      350
Number of independent variables:  5
Mtry:                             2
Target node size:                 10
Variable importance mode:         permutation
```

```
Splitrule:                          gini
OOB prediction error (Brier s.):    0.1553703
```

Solution Task 2

```r
t.i <- importance(rf.1)
t.i <- t.i[order(t.i)]
par(mar=c(3,6,2,2))
barplot(t.i, horiz = T, las = 2)
```



Largest importance is reported for slope, cplan and elev. This alings with the different distribution per response class observed in the boxplots above. However, log10_carea (log of flow accumulation area) and cprof still contribute to the model fit, most likely in tails of their distributions or in by (non-linear) interactions with the other covariates.

1.3 Compute predictions and covariate interpretation

Load the area to be predicted containing the covariates:

```
library(terra)
ta <- rast(system.file("raster/ta.tif", package = "spDataLarge"))
```

> Task 1
>
> Create predictions for the entire extent of the `ta.tif`. Use `predict()` with the random forest object from the section above. Use `as.data.frame()` to transform the raster values into a `data.frame`. What information/values does the function `predict()` return?

> Task 2
>
> Create a `SpatRaster` object with the function `rast()` containing the predictions for the class `TRUE`. Plot the resulting map.

Hint

You can get `x` and `y` coordinates for each pixel using `as.data.frame(, xy = TRUE)`.

The predictions were now created for the entire extent. We only have observations in the central part. Therefore, we need crop the predictions by the study area to avoid spatial extrapolation.

Note: this could also have been done before predicting to increase computational efficiency.

Load the study area:

```
data("study_mask", package = "spDataLarge")
```

> Task 3
>
> Set the pixels outside of the study area to `NA` using the function `mask()` from `terra`.

Solution Task 1

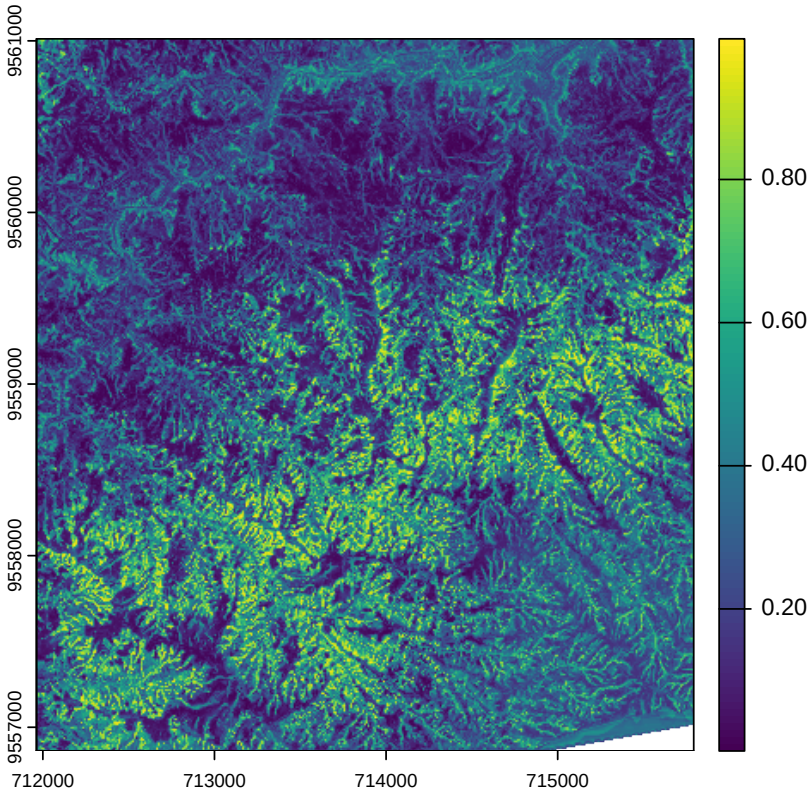
```
d.pred <- predict(object = rf.1, data = as.data.frame(ta))
str(d.pred)
```

```
List of 5
 $ predictions              : num [1:158326, 1:2] 0.375 0.344 0.3 0.16 0.251 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:2] "FALSE" "TRUE"
 $ num.trees                : num 500
 $ num.independent.variables: num 5
 $ num.samples              : int 158326
 $ treetype                 : chr "Probability estimation"
 - attr(*, "class")= chr "ranger.prediction"
```

A list is returned with some general information. The first slot contains a matrix of the predictions with probability for each class. The order of the classes is given in the attributes.

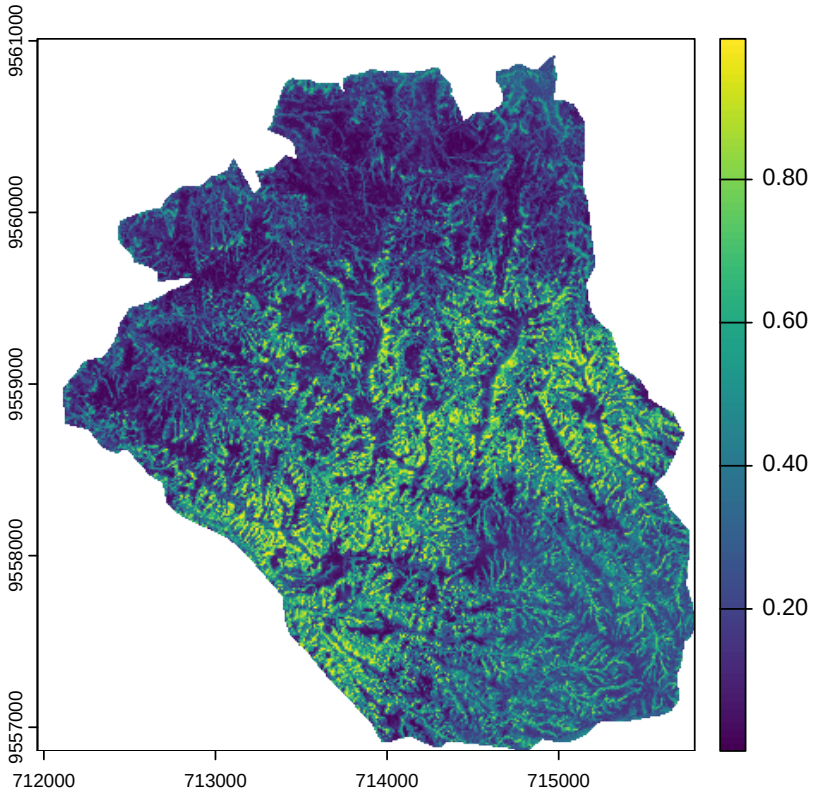
Solution Task 2

```
d.xy <- as.data.frame(ta, xy = TRUE)
d.xy$predTRUE <- d.pred$predictions[,2]
r.pred <- rast(d.xy, type = "xyz", crs = crs(ta))
plot(r.pred[["predTRUE"]])
```



Solution Task 3

```
r.pred <- mask(r.pred, study_mask)
plot(r.pred[["predTRUE"]])
```



1.4 Model interpretation (optional)

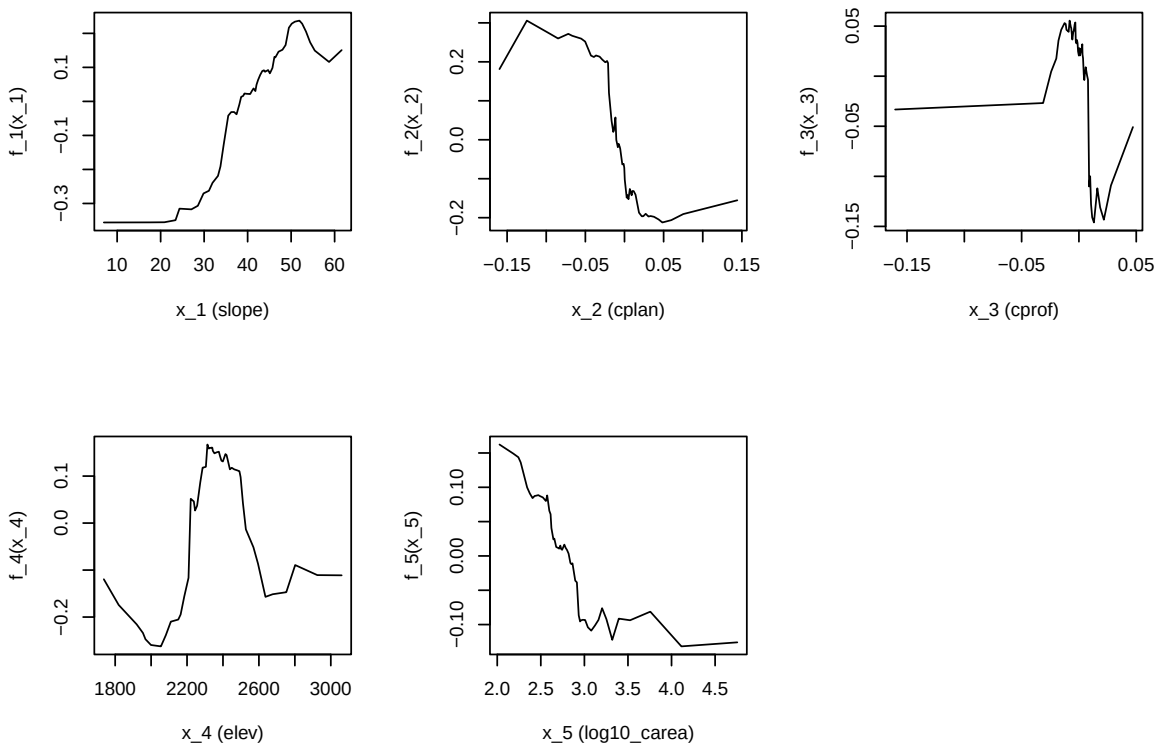
Inform yourself on accumulated local effects plots: Chapter 8.2, Molnar, 2024.

> Task 1
>
> Print accumulated local effects plots for all covariates using the function `ALEPlot()` from the package `ALEPlot`. What is the interpretation of these plots?
> Note: You will need to define a prediction function (`yhat` on the help page of `ALEPlot()`). Make sure this function returns a vector of the probabilities just for one class, see output of `predict()` above.

Solution Task 1

```
library(ALEPlot)
yhat <- function(X.model, newdata){ as.numeric(predict(object = X.model, data = newdata, )$pr
```

```
par(mfrow = c(2,3))
for( ii in 1:5){
  ALE.1=ALEPlot(lsl[,4:8], rf.1, pred.fun = yhat, J = ii, K = 50, NA.plot = TRUE)
}
```



Slope shows a nearly linear relationship starting from value of 25 degrees. cplan seems to have an inflection point at a value of 0 (no curvature), hence the covariate nearly operates as a binary class covariate. There are still some tree nodes split at other values than 0, hence it is still adisiable to use cplan as a continious covariate. elev seems to relate in a nearly quadratic way.

Note also the different range of the y-axis of the plots. The more important covariates according to the importance plot above show larger range compared to the less important ones.

1.5 Spatial coordinates as covariates

```
a <- 30 # rotation angle
x*cos(a/180*pi) - y*sin(a/180*pi)
x*sin(a/180*pi) + y*cos(a/180*pi)
```

Task 1

Add spatial coordinate axis including rotation by e.g. 30 and 60 degrees as additional covariates. Fit the above random forest model again and compute predictions and create the map.

Task 2

Compare the models by comparing the out-of-bag (OOB) model fit, covariate importance and the output maps. Create a difference map with `new.raster <- raster1 - raster2`.
If you have done so above, also compute accumulated local effects plots.

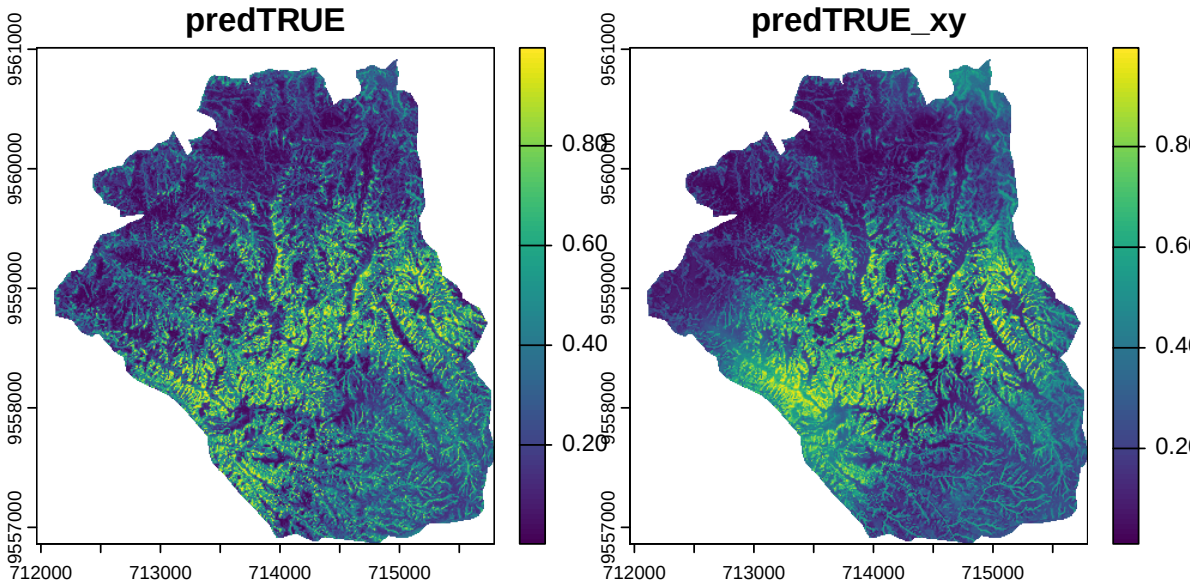
Solution Task 1

```
lsl$x30 <- lsl$x*cos(30/180*pi) - lsl$y*sin(30/180*pi)
lsl$y30 <- lsl$x*sin(30/180*pi) + lsl$y*cos(30/180*pi)
lsl$x60 <- lsl$x*cos(60/180*pi) - lsl$y*sin(60/180*pi)
lsl$y60 <- lsl$x*sin(60/180*pi) + lsl$y*cos(60/180*pi)

rf.2 <- ranger(y = lsl$lslpts,
               x = lsl[c(1:2,4:12)],
               importance = "permutation",
               probability = T, seed = 13)
rf.2
```

```
Ranger result

Call:
 ranger(y = lsl$lslpts, x = lsl[c(1:2, 4:12)], importance = "permutation",      probability =

Type:                             Probability estimation
Number of trees:                  500
```

```
Sample size:                        350
Number of independent variables:  11
Mtry:                               3
Target node size:                   10
Variable importance mode:           permutation
Splitrule:                          gini
OOB prediction error (Brier s.):  0.1500099
```

```r
# add rotated coordinate axis to prediction data.frame
d.xy$x30 <- d.xy$x*cos(30/180*pi) - d.xy$y*sin(30/180*pi)
d.xy$y30 <- d.xy$x*sin(30/180*pi) + d.xy$y*cos(30/180*pi)
d.xy$x60 <- d.xy$x*cos(60/180*pi) - d.xy$y*sin(60/180*pi)
d.xy$y60 <- d.xy$x*sin(60/180*pi) + d.xy$y*cos(60/180*pi)

# predict
d.pred2 <- predict(object = rf.2, data = d.xy)

# extract predictions and create raster
d.xy$predTRUE_xy <- d.pred2$predictions[,2]

r.pred <- rast(d.xy, type = "xyz", crs = crs(ta))
r.pred <- mask(r.pred, study_mask)
plot(r.pred, c(6, 11)) # plot multiple bands
```



Solution Task 2

Random forest model

```
rf.1
```

```
Ranger result

Call:
 ranger(y = lsl$lslpts, x = lsl[4:8], importance = "permutation",      probability = T, seed

Type:                             Probability estimation
Number of trees:                  500
Sample size:                      350
Number of independent variables:  5
Mtry:                             2
```

```
Target node size:                  10
Variable importance mode:          permutation
Splitrule:                         gini
OOB prediction error (Brier s.):   0.1553703
```

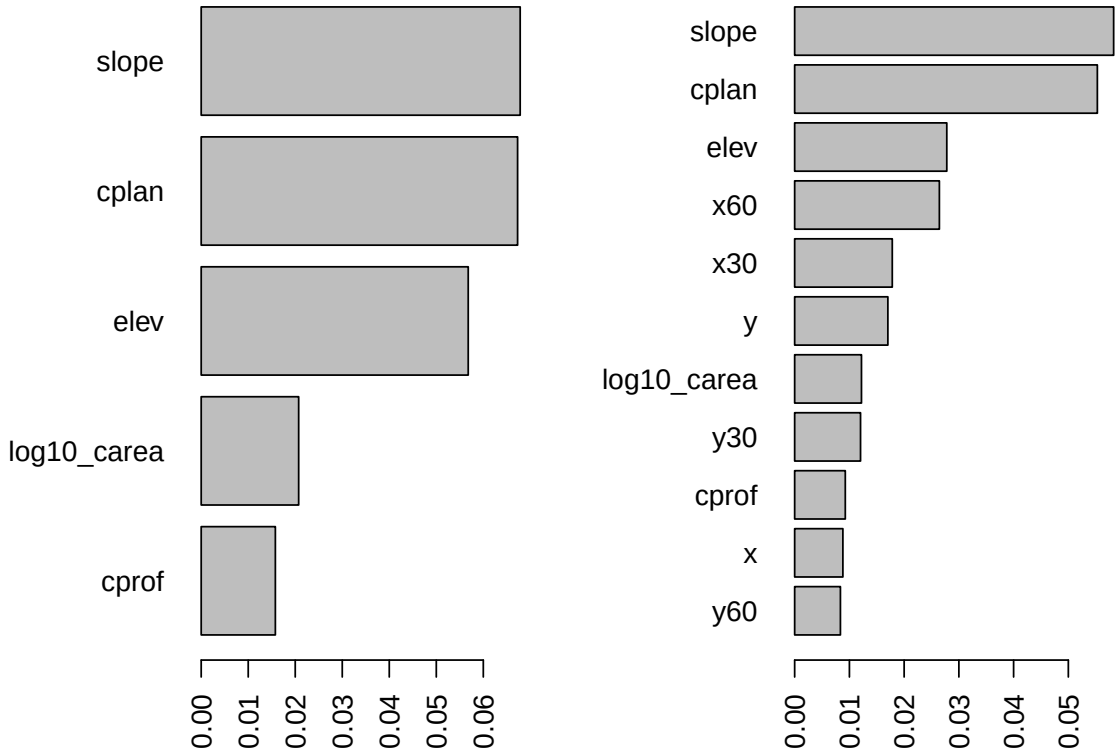
```
rf.2
```

```
Ranger result

Call:
 ranger(y = lsl$lslpts, x = lsl[c(1:2, 4:12)], importance = "permutation",      probability =

Type:                              Probability estimation
Number of trees:                   500
Sample size:                       350
Number of independent variables:   11
Mtry:                              3
Target node size:                  10
Variable importance mode:          permutation
Splitrule:                         gini
OOB prediction error (Brier s.):   0.1500099
```

Brier score (similar to a RMSE, but for probabilistic predictions) is somewhat lower for the larger model including coordinate axis.

Covariate importance

```
t.i2 <- importance(rf.2)
t.i2 <- t.i2[order(t.i2)]
par(mar=c(3,6,2,2), mfrow = c(1,2))
barplot(t.i, horiz = T, las = 2)
barplot(t.i2, horiz = T, las = 2)
```



slope, cplan and elev still rank high in the importance plot. But, the covariates of lower importance (log10_carea, cprof) are outperformed by some of the rotated coordinate axis. This indicates there is some spatial trend not accounted for by the thematic covariates.

Difference map

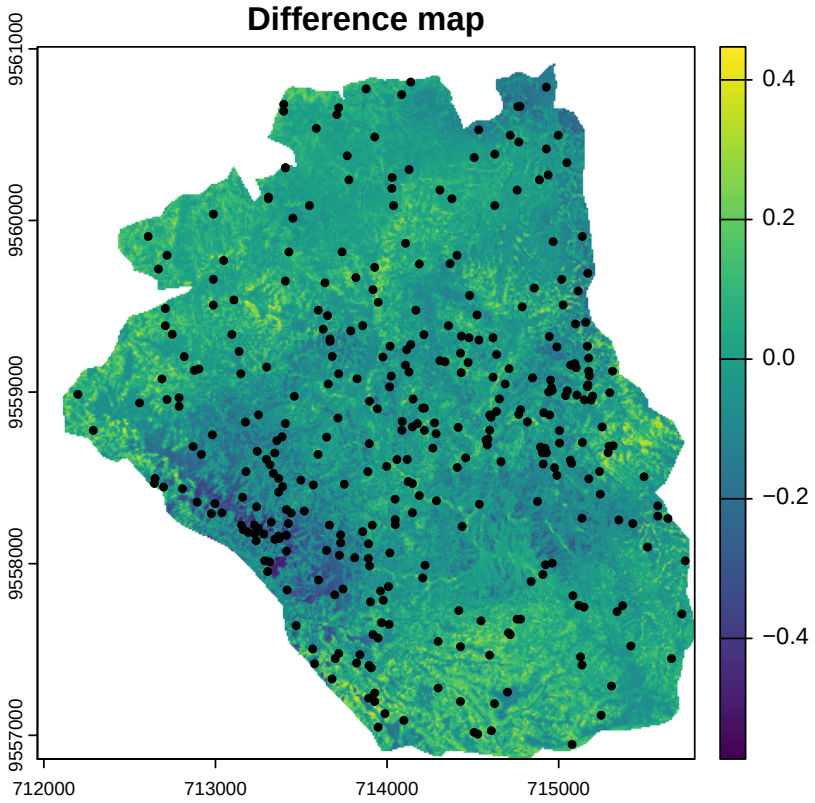
```
# summary of the difference of predicted probabilities
summary(d.xy$predTRUE - d.xy$predTRUE_xy)
```

```
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-0.572753 -0.075642 -0.003978 -0.010429  0.066360  0.527910
```

```
library(sf)
```

```
Linking to GEOS 3.11.1, GDAL 3.6.2, PROJ 9.1.1; sf_use_s2() is TRUE
```

```
r.pred$diff <- r.pred[["predTRUE"]] - r.pred[["predTRUE_xy"]]
plot(r.pred[["diff"]], main = "Difference map")
# add observed locations
v.lsl <- st_as_sf(lsl, coords = c("x", "y"), crs = 32717)
points(v.lsl)
```

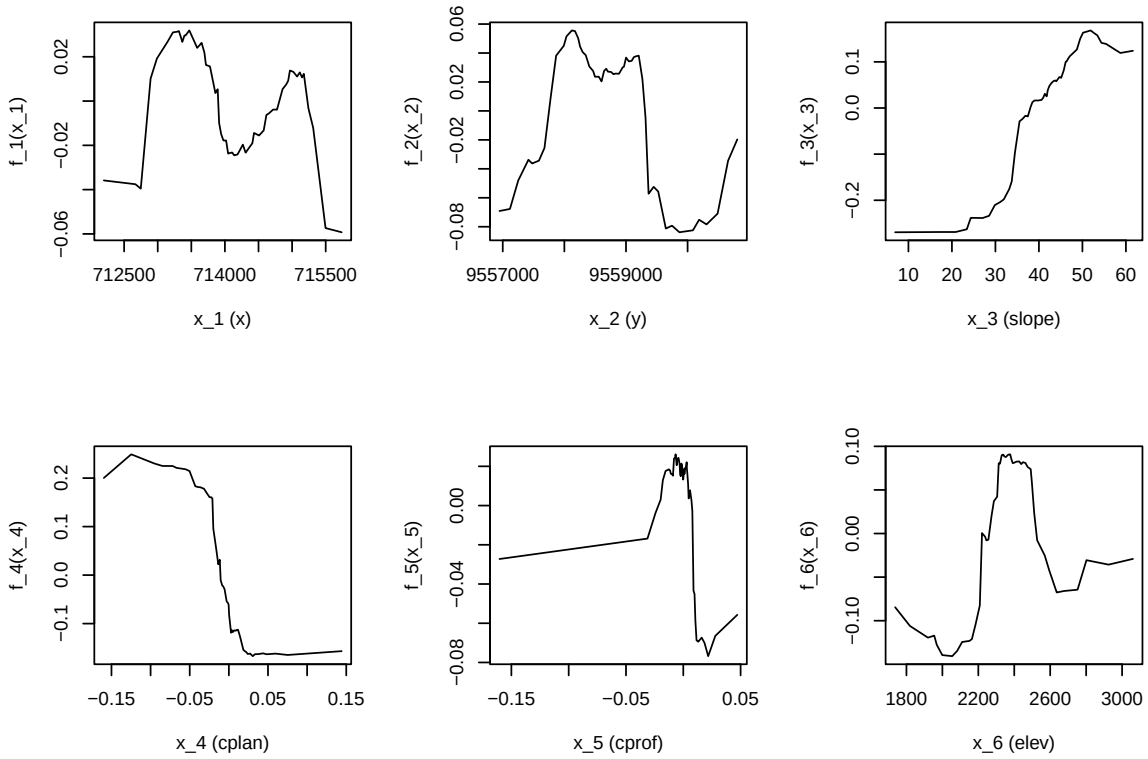


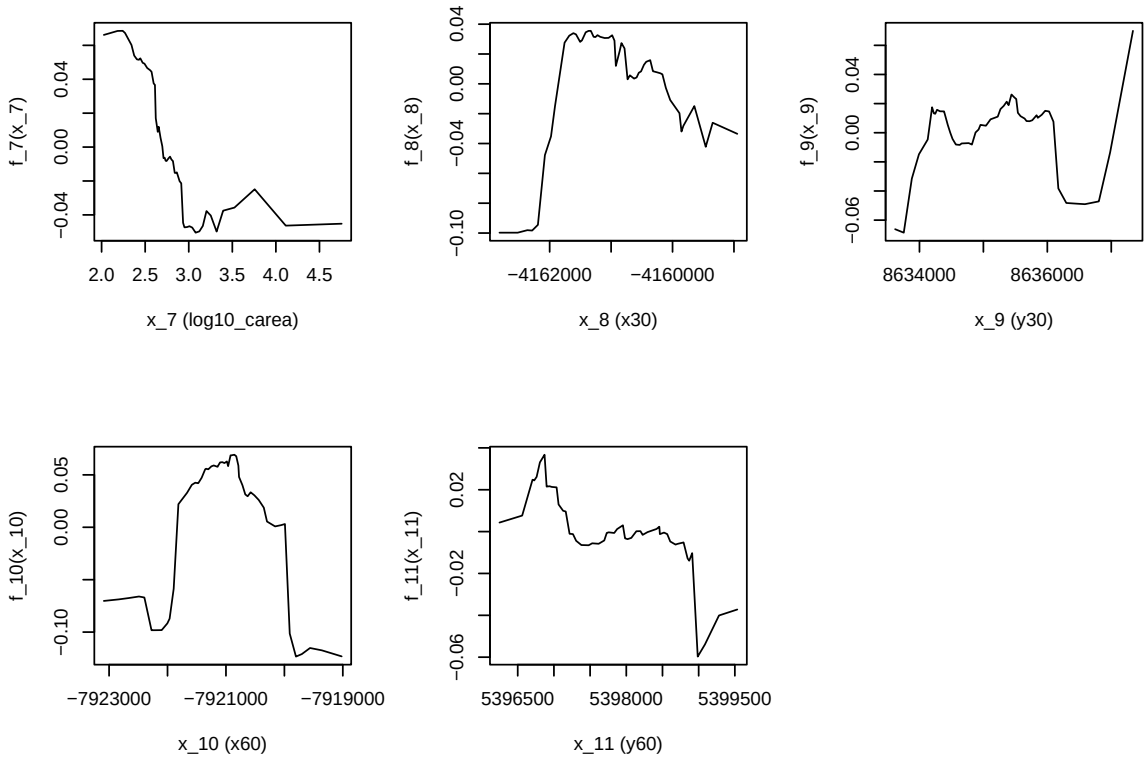
On average, the model with `x` and `y` coordinates predicts slightly larger landslide probabilities. By inspecting the maps, we see the pattern becoming more pronounced. In the North of the study are the probabilities become lower (bluer), in the Southeast the probabilities have become larger (yellower in the probability prediction map). The overall pattern remains the same, it seems including the `x` and `y` will lead to a clearer decision to assign the final class (discrete prediction).

ALE plots

```
par(mfrow = c(2,3))
for( ii in 1:(ncol(lsl)-1)){
```

```
  ALE.1=ALEPlot(lsl[,-c(3)], rf.2, pred.fun = yhat, J = ii, K = 50, NA.plot = TRUE)
}
```





The modelled relationships of the thematic covariates remain roughly the same. The rotated coordinate axis show highly non-linear behaviour. This indicates overall non-linear trends, but most likely more modelling of local fluctuations. To know more about the contribution to local prediction we would need to investiate the interactions of the coordinate axis to see how the study area gets *segmeted* by the axis.

2 Spatial predict continuous response with random forest

We will use the `berne` soil mapping dataset from the R package `geoGAM`. Due to size limitations of CRAN `berne.grid` for predictions only covers a very small part of the study area. We can still use it, to exemplify the prediction process.

Use topsoil pH `ph.0.10` as response. Starting from column `cl_mt_etap_pe` you find a large number of potential covariates in the dataset.

```
library(geoGAM)
data(berne)
data(berne.grid)
```

2.1 Explore dataset

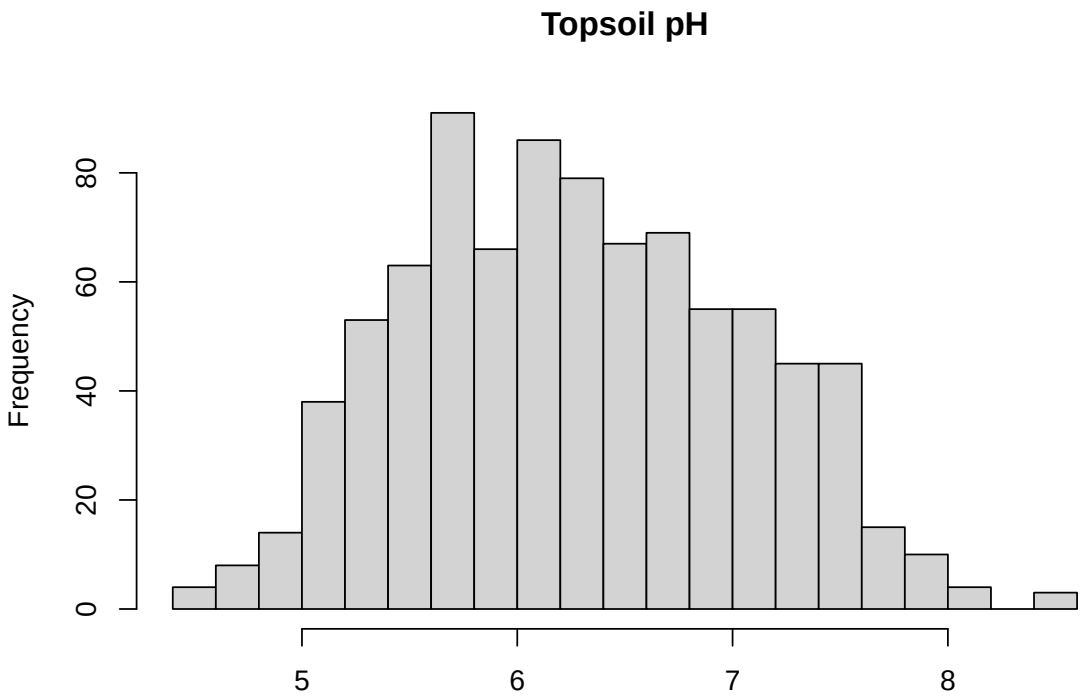
> Task 1
>
> Plot a histogram of the response. Investigate the value range and the completeness of the data.

> Task 2
>
> Screen the covariates. What data types are there?

> Task 3
>
> Create an `sf` object and display the data points with the response using `mapview`. The coordinate reference system information can be found on the help page of the dataset.

Solution Task 1

```
hist(berne$ph.0.10, breaks = 20, main = "Topsoil pH", xlab = "")
```




```
summary(berne$ph.0.10)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   4.50    5.80    6.30    6.33    6.90    8.50     182
```

The distribution of the response is about normal. There are 182 missing values in the response.

Solution Task 2

```
# ?berne # for more information
table( unlist( lapply(berne[,c(14:ncol(berne))], class) ) )
```

```
 factor numeric
     11     214
```

There are mostly numeric covariates in the data, but also 11 factors. They originate from geological maps or overview soil maps.

```
table(complete.cases(berne[,c(14:ncol(berne))]))
```

```
FALSE  TRUE
   13  1039
```

```
table(NAcovars = !complete.cases(berne[,c(14:ncol(berne))]),
      NAresponse = is.na(berne$ph.0.10))
```

```
        NAresponse
NAcovars FALSE TRUE
   FALSE   862  177
   TRUE      8    5
```

There are some additional rows with missing values in the covariates. `berne` is a legacy soil dataset with soil data sampled in the 1970ties. Not all of the observed soil properties are available for all locations. We will remove the rows with missing values for the subsequent analysis.

```
table(complete.cases(berne.grid))
```

```
FALSE
 4594
```

```
f.na.sum <- function(x){sum(is.na(x))}
l <- unlist(lapply(berne.grid, f.na.sum))

l[l>0]
```

```
ge_geo500h1id   tr_se_twi2m
         4594            39
```

```
berne$ge_geo500h1id <- NULL
berne$tr_se_twi2m <- NULL
```

The prediction grid seems also to have missing values. Since we do not have any further knowledge about the dataset, we for now remove these columns from the analysis.

Solution Task 3

```
library(mapview)
v.berne <- st_as_sf(berne, coords = c("x", "y"), crs = 21781)
mapview(v.berne, zcol = "ph.0.10")
```

2.2 Fit random forest model

Task 1

Fit a random forest model using `ranger`. Remove rows with missing values in the response or the covariates. We use the predefined calibration and validation datasets. Hence, only use the rows labeled "calibration" in row `dataset`.

Task 2

Run the Boruta model selection algorithm with function `Boruta()` from the R package `Boruta`. Remove covariates that are considered unimportant according to the results.

Task 3 (optional)

Tune the number of covariates randomly chosen for testing at each tree split (`mtry`). You can either implement this yourself (`apply` or `for` over the possible `mtry` values which are `1:number.covariates`, then select `mtry` of the model with the lowest OOB mean squared error) or use a meta-package like `caret`.

Hint

Here some input how it can be done with `caret`:

```
# create a matrix with all settings to test
# we limit ourself to mtry
options.to.test <- expand.grid(
  .splitrule = c("variance"),
  .min.node.size = c(5),
  .mtry = c(5, 15, 20, ... )
)
# model training with different settings
rf.caret <- train(
  x = d.ph[, names .. of covariates still in the game ],
```

```
  y = d.ph$ph.0.10,
  tuneGrid = options.to.test,
  method = "ranger",
  trControl = trainControl(method = "cv", number = 10)
)
```

Solution Task 1

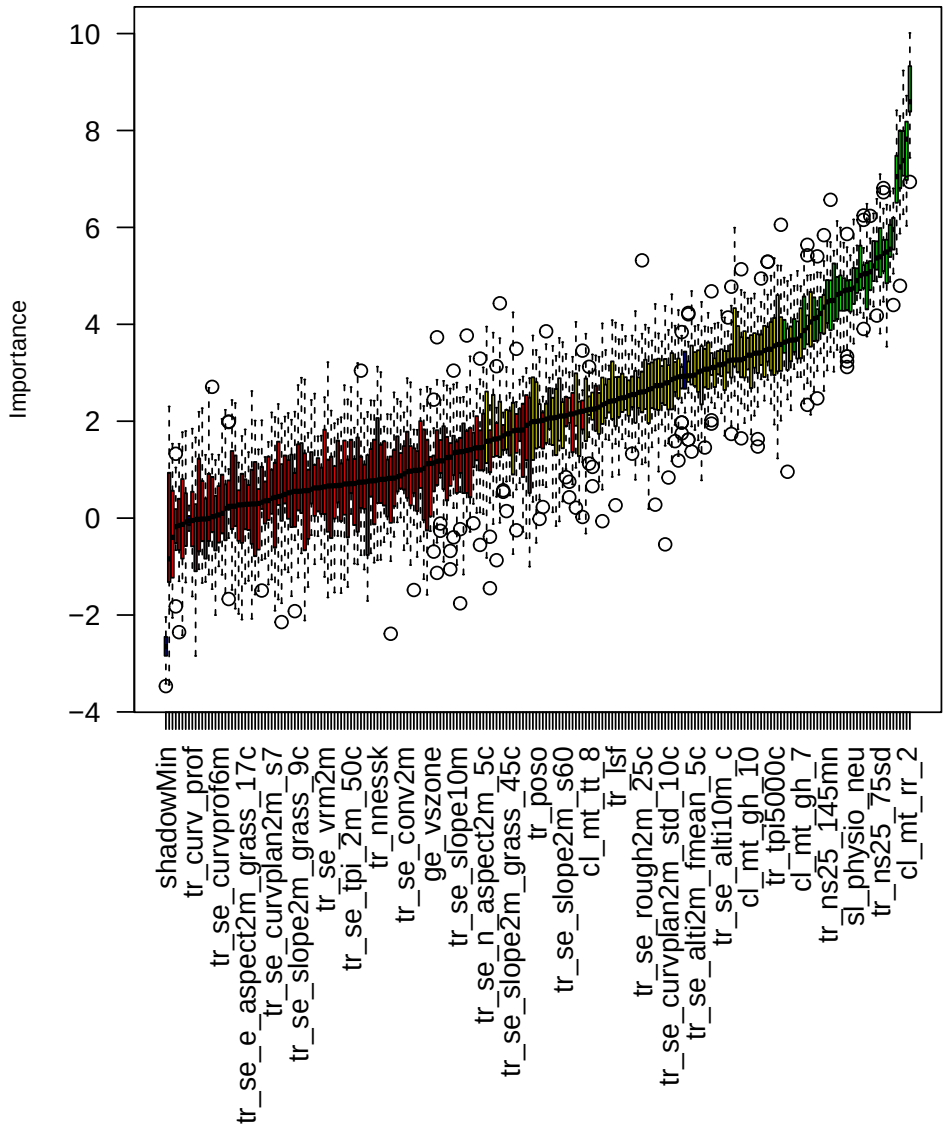
```
d.ph <- berne[berne$dataset == "calibration" & !is.na(berne$ph.0.10), ]
d.ph <- d.ph[complete.cases(d.ph[14:ncol(d.ph)]), ]

rf.ph <- ranger(x = d.ph[, 14:ncol(d.ph) ],
                y = d.ph$ph.0.10,
                seed = 13)
rf.ph
```

```
Ranger result

Call:
 ranger(x = d.ph[, 14:ncol(d.ph)], y = d.ph$ph.0.10, seed = 13)

Type:                             Regression
Number of trees:                  500
Sample size:                      668
Number of independent variables:  223
Mtry:                             14
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.2863646
R squared (OOB):                  0.4860319
```

Solution Task 2

```
library(Boruta)
set.seed(13)
boruta.model <- Boruta(x = d.ph[, 14:ncol(d.ph) ],
                       y = d.ph$ph.0.10,
```

```
                        maxRuns = 30)

# Check ranking compared to shadow variables
par(oma = c(8,3,2,2))
plot(boruta.model, las = 2, xlab = "", cex.lab = 0.8)
```



Selected covariates by Boruta

```
# use for example "Confirmed" (green) and "Tentative" (yellow) > Max.Shadow.
names.cov.selected <- names(boruta.model$finalDecision[ boruta.model$finalDecision %in% c("Te
names.cov.selected
```

```
 [1] "cl_mt_etap_ro"   "cl_mt_gh_2"      "cl_mt_gh_3"      "cl_mt_gh_4"
 [5] "cl_mt_gh_5"      "cl_mt_gh_6"      "cl_mt_gh_8"      "cl_mt_gh_9"
 [9] "cl_mt_pet_pe"    "cl_mt_pet_ro"    "cl_mt_rr_1"      "cl_mt_rr_10"
[13] "cl_mt_rr_11"     "cl_mt_rr_12"     "cl_mt_rr_2"      "cl_mt_rr_3"
[17] "cl_mt_rr_4"      "cl_mt_rr_5"      "cl_mt_rr_7"      "cl_mt_rr_8"
[21] "cl_mt_rr_9"      "cl_mt_rr_y"      "cl_mt_swb_pe"    "cl_mt_swb_ro"
[25] "sl_physio_neu"   "tr_cindx50_25"   "tr_nego"         "tr_ns25_145mn"
[29] "tr_ns25_75sd"    "tr_se_twi2m_s30" "tr_se_twi2m_s7"  "tr_tsc25_40"
```

Random forest with subset of covariates

```
rf.ph.sel <- ranger(x = d.ph[, names.cov.selected],
                    y = d.ph$ph.0.10,
                    seed = 13)
rf.ph.sel
```

```
Ranger result

Call:
 ranger(x = d.ph[, names.cov.selected], y = d.ph$ph.0.10, seed = 13)

Type:                             Regression
Number of trees:                  500
Sample size:                      668
Number of independent variables:  32
Mtry:                             5
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.2921226
R squared (OOB):                  0.4756975
```

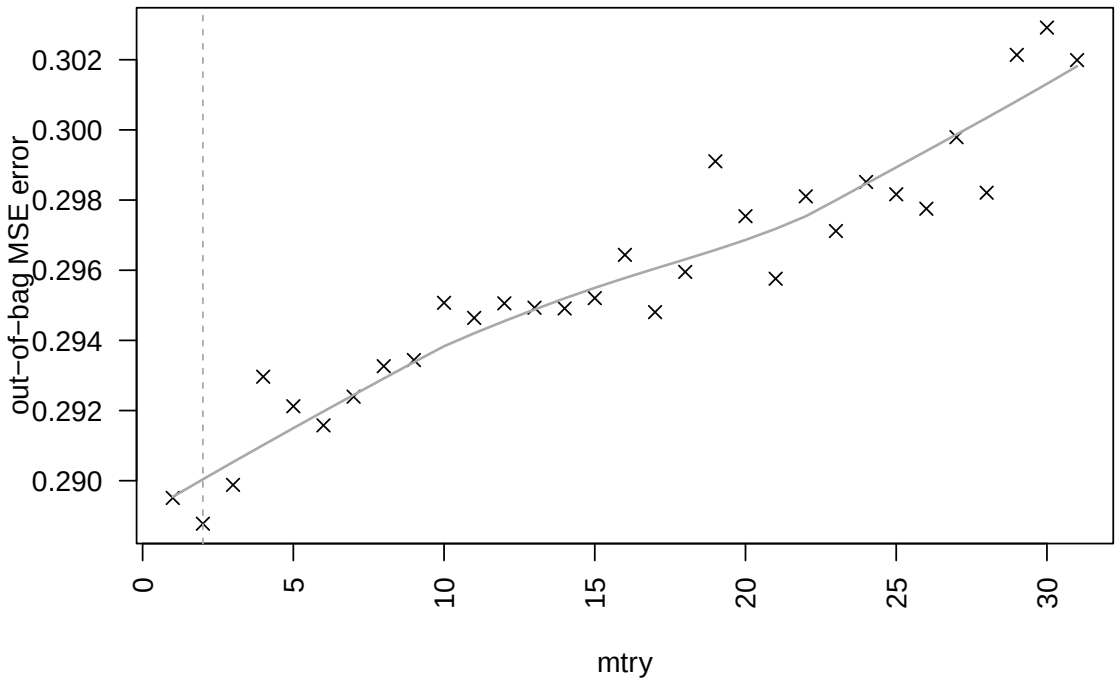
Solution Task 3

Own implementation based on OOB MSE

```r
library(parallel)
# Define function to use below
f.tune.randomforest <- function(test.mtry, # mtry to test
                                d.cal,     # calibration data.frame
                                l.covariates ){ # list of covariates
  # fit random forest with mtry = test.mtry
  rf.tune <- ranger(x = d.cal[, l.covariates ],
                    y = d.cal[, "ph.0.10"],
                    mtry = test.mtry,
                    seed = 13)
  # return the mean squared error (mse) of this model fit
  return( rf.tune$prediction.error )
}

# Vector of mtry to test
seq.mtry <- 1:(length(names.cov.selected) - 1)
# Only take every fifth mtry to speed up tuning
# seq.mtry <- seq.mtry[ seq.mtry %% 5 == 0 ]

# Apply function to sequence.
# (without parallel computing this takes a while)
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.ph,
                     l.covariates = names.cov.selected )

# Hint: Who is not comfortable with "mclapply"
# the same could be achieved with
# for(test.mtry in seq.mtry){
#    .. 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) ]
# Optimal mtry
s.mtry
```

```
[1] 2
```

```
# plot result
plot( mtry.oob$mtry.n, mtry.oob$mtry.OOBe, pch = 4, las = 2,
      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")
```



Use caret with cross-validation

```
library(caret)
# create a matrix with all settings to test
# we limit ourself to mtry
options.to.test <- expand.grid(
  .splitrule = c("variance"),
  .min.node.size = c(5),
  .mtry = 1:(length(names.cov.selected) - 1)
)
```

```r
# model training with different settings
rf.caret <- train(
  x = d.ph[, names.cov.selected ],
  y = d.ph$ph.0.10,
  tuneGrid = options.to.test,
  method = "ranger",
  trControl = trainControl(method = "cv", number = 10)
)
# print best settings
rf.caret$bestTune
```

```
  mtry splitrule min.node.size
1    1  variance             5
```

Fit random forest with optimal `mtry`

```r
rf.ph.sel.tuned <- ranger(x = d.ph[, names.cov.selected ],
                          y = d.ph$ph.0.10,
                          mtry = rf.caret$bestTune$mtry,
                          seed = 13)
rf.ph.sel.tuned
```

```
Ranger result

Call:
 ranger(x = d.ph[, names.cov.selected], y = d.ph$ph.0.10, mtry = rf.caret$bestTune$mtry,

Type:                             Regression
Number of trees:                  500
Sample size:                      668
Number of independent variables:  32
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.2895098
R squared (OOB):                  0.4803869
```

Optimal `mtry` is very low. Depending on how it is precisely evaluated optimal `mtry` is 1 or 2. Low `mtry` means a large degree of randomization is used to fit the regression trees as only 1 or 2 randomly chosen covariates are tried at each tree node to split the data.

2.3 Investigate spatial structure of residuals

Task 1

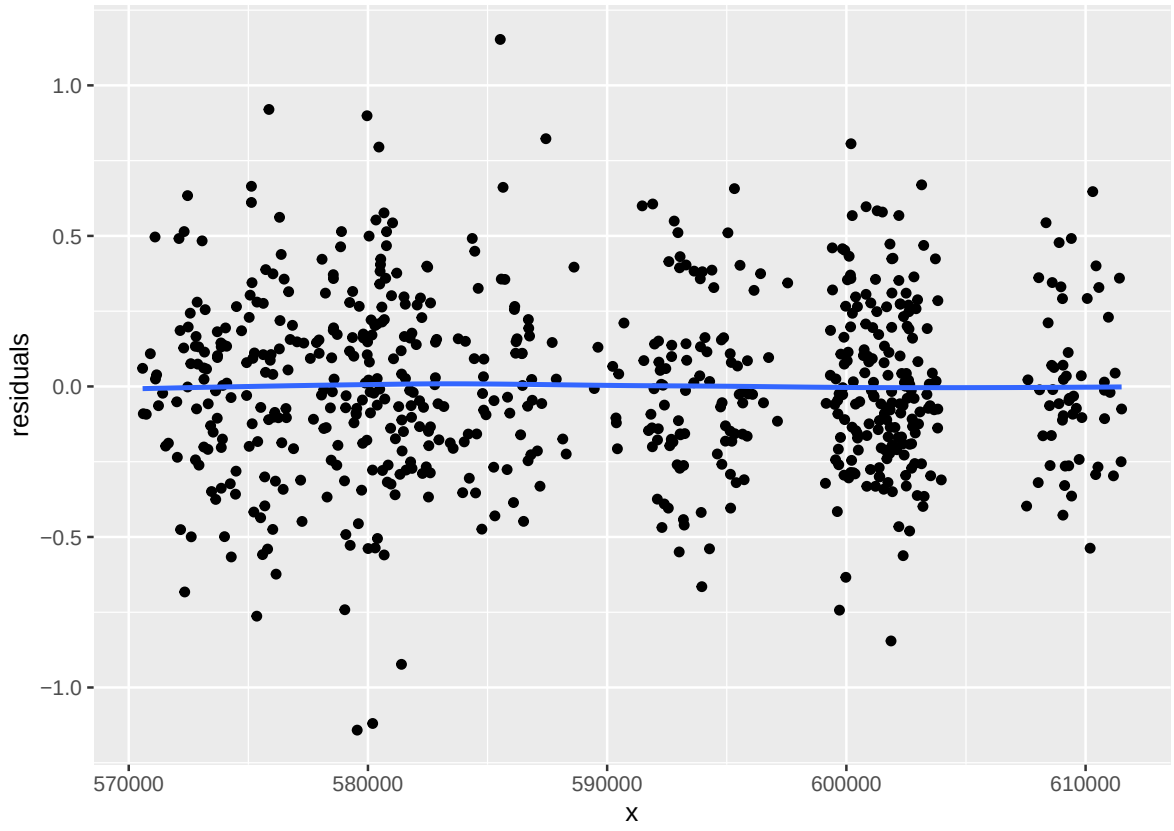
Compute the random forest residuals (use `predict()` on the calibration data). Investigate the residuals for spatial structure. Plot the residuals against `x` and `y` axis and create a sample variogram.

Task 2

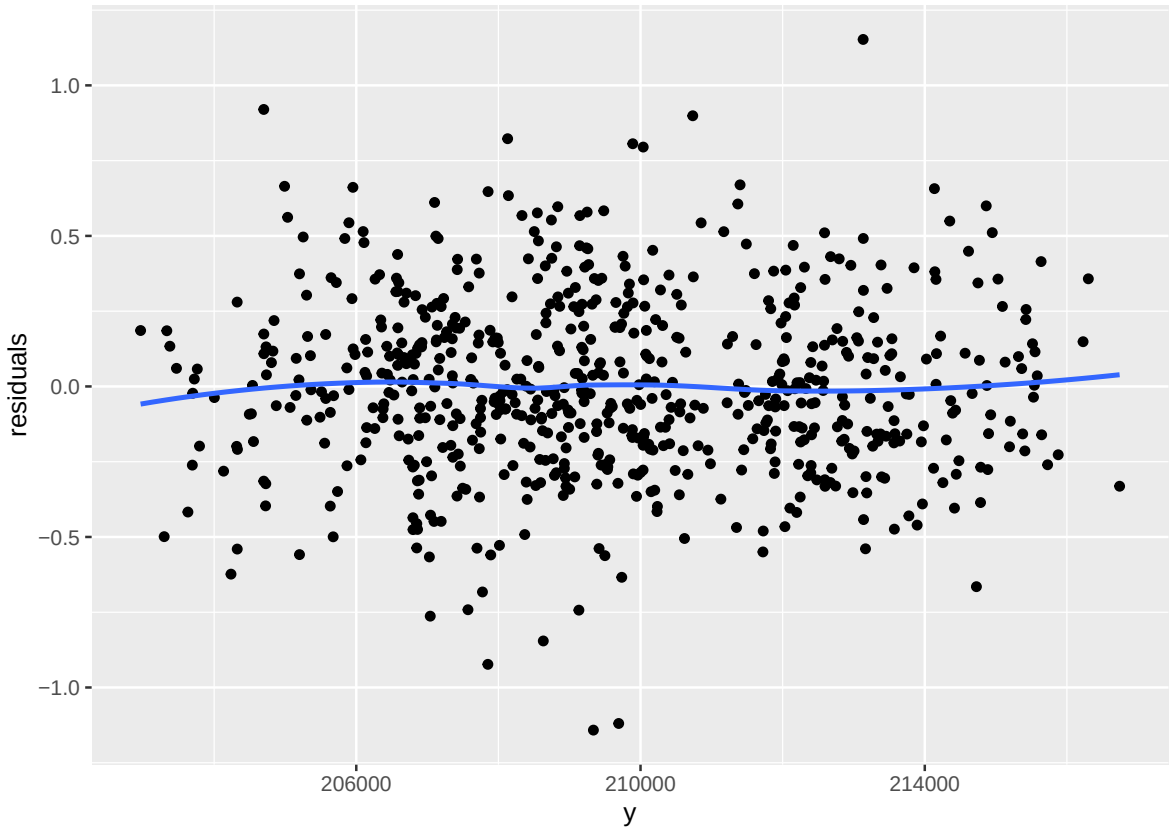
Test if the model can be improved by adding rotated coordinate axis. Compare the OOB mean squared error of the model with and without the coordinates.

Solution Task 1

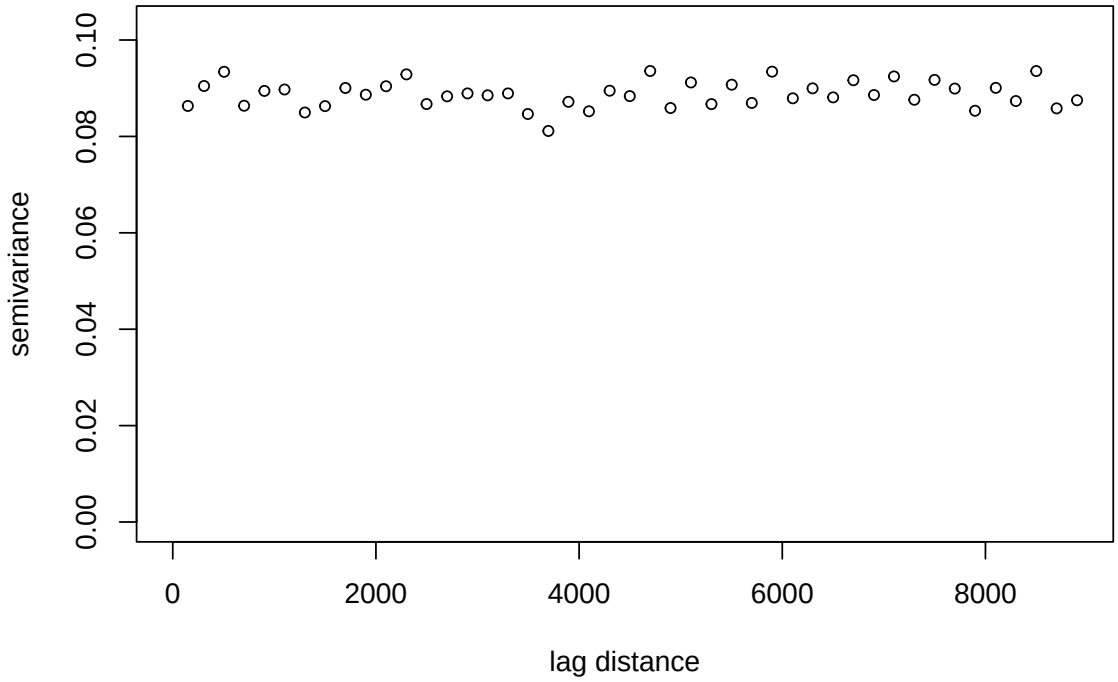
```
fitted <- predict(rf.ph.sel.tuned, data = d.ph[, names.cov.selected ])
d.ph$residuals <- d.ph$ph.0.10  - fitted$predictions

ggplot(d.ph, aes(x = x, y = residuals)) +
  geom_point() + geom_smooth(se=F)
```




```
ggplot(d.ph, aes(x = y, y = residuals)) +
  geom_point() + geom_smooth(se=F)
```




```
library(georob)
plot(sample.variogram(d.ph$residuals, locations = d.ph[, c("x", "y")],
  estimator = "matheron", lag.dist.def = 200, max.lag = 9000))
```



According to the lowess scatterplot smoother we do not observe any remaining trend along the coordinate axis.

The sample variogram shows a "pure nugget" variogram, i.e. there seems to be not short range auto-correlation in the residuals.

From this result, we do not expect that the predictions improve when including spatial coordinates.

Solution Task 2

```
# add rotated coordinate axis to training data.frame
d.ph$x30 <- d.ph$x*cos(30/180*pi) - d.ph$y*sin(30/180*pi)
d.ph$y30 <- d.ph$x*sin(30/180*pi) + d.ph$y*cos(30/180*pi)
d.ph$x60 <- d.ph$x*cos(60/180*pi) - d.ph$y*sin(60/180*pi)
d.ph$y60 <- d.ph$x*sin(60/180*pi) + d.ph$y*cos(60/180*pi)

names.xy <- c(names.cov.selected, paste0( rep(c("x", "y"), 3), c("", "30", "60")) )
```

```
rf.ph.sel.tuned.xy <- ranger(x = d.ph[, names.xy],
                             y = d.ph$ph.0.10,
                             mtry = rf.caret$bestTune$mtry,
                             seed = 13)
rf.ph.sel.tuned.xy
```

```
Ranger result

Call:
 ranger(x = d.ph[, names.xy], y = d.ph$ph.0.10, mtry = rf.caret$bestTune$mtry,      seed = 13

Type:                             Regression
Number of trees:                  500
Sample size:                      668
Number of independent variables:  38
Mtry:                             1
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.2873347
R squared (OOB):                  0.4842908
```

According to OOB metrics the model fit improves slightly. We observe an R^2 of 0.480 and a MSE of 0.29 without and a slightly larger R^2 of 0.484 with a slightly lower MSE of 0.287 with the rotated coordinate axis.

This is opposed to the plots (scatterplots and variogram) above. Therefore, rotated `x` and `y` coordinate axis likely form interactions with other covariates. Due to the hierarchical structure of regression trees, they designed to model interactions. We would need to investigate the modeled interaction structure by for example using second-order ALE plots, see Chapter 8.2, Molnar, 2024.

This shows that classical "diagnostics" used in the linear model context may not be reliable indicators for machine learning models.

2.4 Create predictions

Task 1

Compute predictions for `berne.grid` and plot the result. Are you happy with the visual result?

Task 2

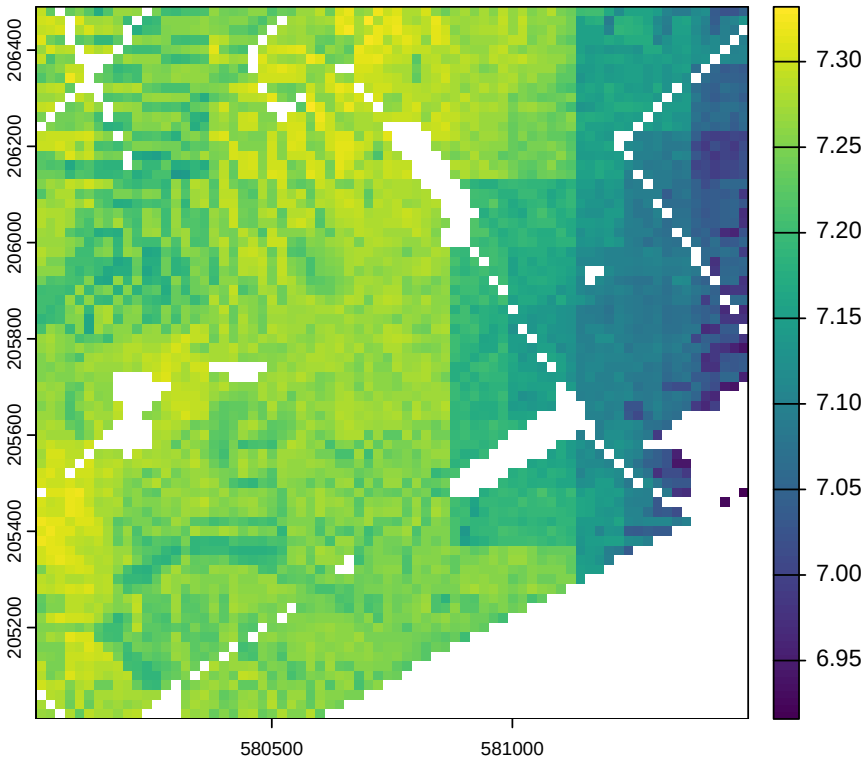
Compute predictions for the rows labeled with "validation" in the column `dataset` in the `berne` dataset. Compute the R^2 and compare it with the OOB R^2.

Task 3

Save the predictions for the "validation" rows together with the observed values and coordinates. We will inspect them in the next lab session.

Solution Task 1

```
library(terra)
berne.grid <- berne.grid[ complete.cases(berne.grid[names.cov.selected]), ]

# add rotated coordinate axis to prediction data.frame
berne.grid$x30 <- berne.grid$x*cos(30/180*pi) - berne.grid$y*sin(30/180*pi)
berne.grid$y30 <- berne.grid$x*sin(30/180*pi) + berne.grid$y*cos(30/180*pi)
berne.grid$x60 <- berne.grid$x*cos(60/180*pi) - berne.grid$y*sin(60/180*pi)
berne.grid$y60 <- berne.grid$x*sin(60/180*pi) + berne.grid$y*cos(60/180*pi)


berne.grid$pred.ph <- predict(rf.ph.sel.tuned.xy, berne.grid)$predictions
r.ph <- rast(berne.grid[, c("x", "y", "pred.ph")], type = "xyz")
plot(r.ph)
```



The predictions created for the small section of the study area available in `berne.grid` seem to be rather "ugly". There are two types of artefacts:

- sort of small scale flow lines, mostly in the Northwest of the map section,
- large vertical lines and large rectangular areas, mainly in the east of the map section.

The latter is certainly not typical for the spatial variation of soil pH. The small scale pattern most likely originates from a derivative of the elevation model. Since the color scale only covers a part of the value range of our response (min: 4.5, max: 8.8) these patterns get exaggerated by the current display.

The large rectangular structures originate from the covariates starting with "cl_mt_..". Those are climatic maps (e.g. average yearly rainfall) that were only available at a large pixel resolution. The pixels are now directly visible in the prediction map. To avoid this outcome we could use data with higher resolution (if available) or smooth the raster with a filter before using it as covariate (e.g. function `terra::focal()`). Smoothing is a practical solution, but the uncertainty in the covariates is not accounted for.

Closely investigating the resulting maps is a required step of spatial prediction to avoid publishing data sets with non-feasible content.

Solution Task 2

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

# add rotated coordinate axis to validation data.frame
d.ph.val$x30 <- d.ph.val$x*cos(30/180*pi) - d.ph.val$y*sin(30/180*pi)
d.ph.val$y30 <- d.ph.val$x*sin(30/180*pi) + d.ph.val$y*cos(30/180*pi)
d.ph.val$x60 <- d.ph.val$x*cos(60/180*pi) - d.ph.val$y*sin(60/180*pi)
d.ph.val$y60 <- d.ph.val$x*sin(60/180*pi) + d.ph.val$y*cos(60/180*pi)

d.ph.val$pred.ph.0.10 <- predict(rf.ph.sel.tuned.xy, d.ph.val)$predictions

# R2 = 1 - RSS / TSS (RSS: residual sum of squares, TSS: total sum of squares)
1 - sum( (d.ph.val$pred.ph.0.10 - d.ph.val$ph.0.10)^2) / sum((d.ph.val$ph.0.10 - mean(d.ph.va
```

```
[1] 0.4519854
```

```
rf.ph.sel.tuned.xy$r.squared
```

```
[1] 0.4842908
```

The R^2 computed on the validation set is only slightly lower compared to the out-of-bag error. Therefore, it seems the prediction model is generalizing well to predict new locations unseen to the model.

However, we would need to investigate the locations of the validation data and if those allow for a valid conclusion regarding the accuracy of the resulting map.

Solution Task 3

```
d.ph.val <- d.ph.val[, c("site_id_unique", "x", "y", "ph.0.10","pred.ph.0.10")]
write.csv(d.ph.val, file = "lab3-validation-set-ph-predictions.csv")
```