Tom Hengl, Leandro Parente and Ichsani Wheeler

Spatial sampling and resampling for Machine Learning

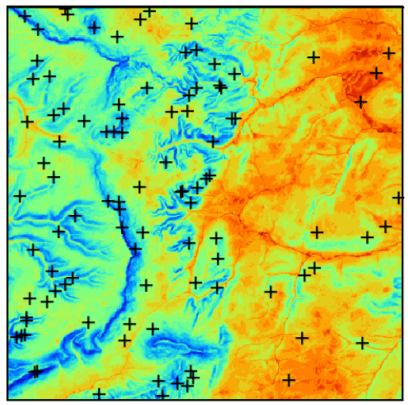
OpenGeoHub foundation, Wageningen, Netherlands

Contents

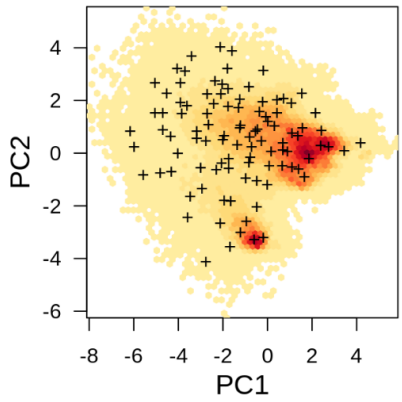
Introduction

Overview

```
if(!knitr:::is_latex_output()){
  knitr::include_graphics("https://zenodo.org/badge/doi/10.5281/zenodo.5886677.svg")
}
```



PCA Ebergotzen FSCS



[1] This Rmarkdown tutorial[2] provides practical instructions, illustrated with sample dataset, on how to generate and evaluate sampling plans using your own data. The specific focus is put on preparing sampling designs for predictive mapping, running analysis and interpretation on existing point data and planning 2nd and 3rd round sampling (based on initial models). A similar tutorial focusing on Spatial and spatiotemporal interpolation using Ensemble Machine Learning[3] is also available.

We use several key R packages and existing tutorials including:

- sp[4] package,
- clhs[5] package,
- mlr[6] package,
- ranger[7] package,

[1] https://opengeohub.github.io/spatial-sampling-ml/
[2] https://opengeohub.github.io/spatial-sampling-ml/
[3] https://opengeohub.github.io/spatial-prediction-eml/
[4] https://github.com/edzer/sp
[5] https://github.com/pierreroudier/clhs
[6] https://mlr.mlr-org.com/
[7] https://github.com/imbs-hl/ranger/

- forestError[8] package,

Other packages of interest for producing spatial sampling:

- SamplingBigData[9] package,
- sf[10] package,
- spatstat[11] package(s),

For an introduction to Spatial Data Science and Machine Learning with R we recommend studying first:

- Baddeley, A., Rubak, E. and Turner, R.: **"Spatial Point Patterns: Methodology and Applications with R"**[12];
- Becker, M. et al.: **"mlr3 book"**[13];
- Irizarry, R.A.: **"Introduction to Data Science: Data Analysis and Prediction Algorithms with R"**[14];
- Molnar, C.: **"Interpretable Machine Learning: A Guide for Making Black Box Models Explainable"**[15];
- Lovelace, R., Nowosad, J. and Muenchow, J.: **"Geocomputation with R"**[16];
- Pebesma, E. and Bivand, R: **"Spatial Data Science: with applications in R"**[17];

If you are looking for a more gentle introduction to spatial sampling methods in R please refer to **?**, **?**, **?** and **?**. The *"Spatial sampling with R"* book by Dick Brus and R code examples are available via https://github.com/DickBrus/SpatialSamplingwithR.

For an introduction to **Predictive Soil Mapping** using R refer to https://soilmapper.org.

Machine Learning in **python** with resampling can be best implemented via the scikit-learn library[18], which matches in functionality what is available via the mlr package in R.

To install the most recent **landmap**, **ranger**, **forestError** and **clhs** packages from Github use:

[8] https://github.com/benjilu/forestError
[9] https://github.com/jlisic/SamplingBigData
[10] https://keen-swartz-3146c4.netlify.app/pointpatterns.html#spatial-sampling-and-simulating-a-point-process
[11] https://spatstat.org/
[12] https://spatstat.org/
[13] https://mlr3book.mlr-org.com/
[14] https://rafalab.github.io/dsbook/
[15] https://christophm.github.io/interpretable-ml-book/
[16] https://geocompr.robinlovelace.net/
[17] https://keen-swartz-3146c4.netlify.app/
[18] https://scikit-learn.org/stable/

```
library(devtools)
devtools::install_github("envirometrix/landmap")
devtools::install_github("imbs-hl/ranger")
devtools::install_github("benjilu/forestError")
devtools::install_github("pierreroudier/clhs")
```

License

[19]

Acknowledgements


This tutorial is based on the **"R for Data Science"**[21] book by Hadley Wickham and contributors.

OpenLandMap[22] is a collaborative effort and many people have contributed data, software, fixes and improvements via pull request. OpenGeoHub[23] is an independent not-for-profit research foundation promoting Open Source and Open Data solutions. **EnvirometriX Ltd.**[24] is the commercial branch of the group responsible for designing soil sampling designs for the **AgriCapture**[25] and similar soil monitoring projects.

[26]

AgriCaptureCO2[27] receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. **101004282**[28].

[19] http://creativecommons.org/licenses/by-sa/4.0/
[20] http://creativecommons.org/licenses/by-sa/4.0/
[21] https://r4ds.had.co.nz/
[22] https://openlandmap.org
[23] https://opengeohub.org
[24] https://envirometrix.nl
[25] https://agricaptureco2.eu/
[26] https://envirometrix.nl
[27] https://agricaptureco2.eu/
[28] https://cordis.europa.eu/project/id/101004282

Chapter 1
Generating spatial sampling

You are reading the work-in-progress Spatial Sampling and Resampling for Machine Learning. This chapter is currently draft version, a peer-review publication is pending. You can find the polished first edition at https://opengeohub.github.io/spatial-sampling-ml/.

1.1 Spatial sampling

Sampling in statistics is done for the purpose of estimating population parameters and/or for testing of experiments. If Observations and Measurements (O&M) are collected in space i.e. as geographical variables this is referred to as **spatial sampling** and is often materialized as **a point map** with points representing locations of planned or implemented O&M. Preparing a spatial sampling plan is a type of **design of experiment** and hence it is important to do it right to avoid any potential bias.

Spatial sampling or producing and implementing sampling designs are common in various fields including physical geography, soil science, geology, vegetation science, ecology and similar. Imagine an area that potentially has problems with soil pollution by heavy metals. If we collect enough samples, we can overlay points vs covariate layers, then train spatial interpolation / spatial prediction models and produce predictions of the target variable. For example, to map soil pollution by heavy metals or soil organic carbon stock, we can collect soil samples on e.g. a few hundred predefined locations, then take the samples to the lab, measure individual values and then interpolate them to produce a map of concentrations. This is one of the most common methods of interest of **geostatistics** where e.g. various kriging methods[1] are used to produce predictions of the target variable (see e.g. **?**).

There are many sampling design algorithms that can be used to spatial sampling locations. In principle, all spatial sampling approaches can be grouped based on the following four aspects:

1. *How objective is it?* Here two groups exist:

[1] http://www.leg.ufpr.br/geor/

   a. Objective sampling designs which are either ***probability sampling***[2] or some experimental designs from spatial statistics;
   b. Subjective or **convenience sampling** which means that the inclusion probabilities are unknown and are often based on convenience e.g. distance to roads / accessibility;

2. *How much identically distributed is it*? Here at least three groups exist:

   a. **Independent Identically Distributed (IID)**[3] sampling designs,
   b. Clustered sampling i.e. non-equal probability sampling,
   c. Censored sampling,

3. *Is it based on geographical or feature space?* Here at least three groups exist:

   a. Geographical sampling i.e. taking into account only geographical dimensions + time;
   b. **Feature-space sampling** i.e. taking into account only distribution of points in feature space;
   c. Hybrid sampling i.e. taking both feature and geographical space into account;

4. *How optimized is it?* Here at least two groups exist:

   a. **Optimized sampling** so that the target optimization criteria reaches minimum / maximum i.e. it can be proven as being optimized,
   b. *Unoptimized sampling*, when either optimization criteria can not be tested or is unknown,

Doing sampling using objective sampling designs is important as it allows us to test hypotheses and produce **unbiased estimation** of population parameters or similar. Many spatial statisticians argue that only previously prepared, strictly followed randomized probability sampling can be used to provide an unbiased estimate of the accuracy of the spatial predictions (**?**). In the case of probability sampling, calculation of population parameters is derived mathematically i.e. that estimation process is unbiased and independent of the spatial properties of the target variable (e.g. spatial dependence structure and/or statistical distribution). For example, if we generate sampling locations using **Simple Random Sampling (SRS)**, this sampling design has the following properties:

1. It is an IID with each spatial location with exactly the same **inclusion probability**,
2. It is symmetrical in geographical space meaning that about the same number of points can be found in each quadrant of the study area,
3. It can be used to derive population parameters (e.g. mean) and these measures are per definition unbiased,
4. Any random subset of the SRS is also a SRS,

[2] https://towardsdatascience.com/an-introduction-to-probability-sampling-methods-7a936e486b5

[3] https://xzhu0027.gitbook.io/blog/ml-system/sys-ml-index/learning-from-non-iid-data

SRS is in principle both objective sampling and IID sampling and can be easily generated provided some polygon map representing the study area. Two other somewhat more complex sampling algorithms with similar properties as SRS are for example different versions of tessellated sampling. The **generalized random tessellation stratified (GRTS) design** was for example used in the USA to produce sampling locations for the purpose of geochemical mapping[4]; a **multi-stage stratified random sampling** design was used to produce LUCAS soil monitoring network of points[5].

Large point datasets representing observations and/or measurements *in-situ* can be used to generate maps by fitting regression and classification models using e.g. Machine Learning algorithms, then applying those models to predict values at all pixels. This is referred to as ***Predictive Mapping***[6]. In reality, many point datasets we use in Machine Learning for predictive mapping do not have ideal properties i.e. are neither IID nor are probabilities of inclusion known. Many are in fact **purposive** and/or **convenience sampling**[7] and hence potentially over-represent some geographic features, are potentially censored and can lead to significant bias in estimation.

1.2 Response surface designs

If the objective of modeling is to build regression models (correlating the target variable with a number of spatial layers representing e.g. soil forming factors), then we are looking at the problem in statistics known as the **response-surface experimental designs**[8]. Consider the following case of one target variable (Y) and one covariate variable (X). Assuming that the two are correlated linearly (i.e. $Y = b_0 + b_1 \cdot X$), one can easily prove that the optimal experimental design is to: (a) determine min and max of X, then put half of the point at X_{min} and the other half at X_{max} (**?**). This design is called the **D1 optimal design**[9] and indeed it looks relatively simple to implement. The problem is that it is the optimal design ONLY if the relationship between Y and X is perfectly linear. If the relationship is maybe close to quadratic than the D1 design is much worse than for example the D2 design (**?**).

In practice we may not know what is the nature of the relationship between Y and X, i.e. we do not wish to take a risk and produce biased estimation. Hence we can assume that it could be a curvilinear relationship and so we need to sample uniformly in the feature space.

[4] https://pubs.usgs.gov/ds/801/

[5] https://esdac.jrc.ec.europa.eu/projects/lucas

[6] https://soilmapper.org

[7] https://methods.sagepub.com/reference/encyclopedia-of-survey-research-methods/n105.xml

[8] https://en.wikipedia.org/wiki/Response_surface_methodology

[9] https://en.wikipedia.org/wiki/Optimal_design#D-optimality

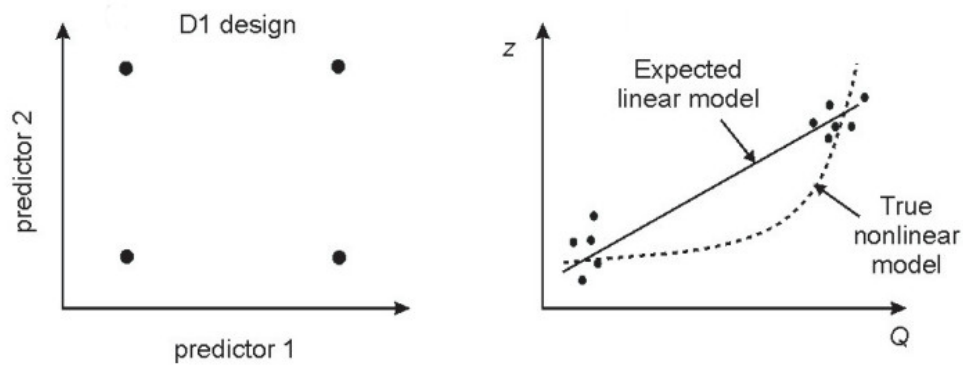


Fig. 1.1 Example of D1 design: (left) D1 design in 2D feature space, (right) D1 design is optimal only for linear model, if the model is curvilinear, it is in fact the worse design than simple random sampling.

1.3 Spatial sampling algorithms of interest

This chapter reviews some common approaches for preparing point samples for a study area that you are visiting for the first time and/or no previous samples or models are available. We focus on the following spatial sampling methods:

- Subjective or convenience sampling,
- Simple Random Sampling (**SRS**) (**??**),
- Latin Hypercube Sampling (**LHS**) and its variants e.g. Conditioned LHS (**??**),
- Feature Space Coverage Sampling (**FSCS**) (**?**) and fuzzy k-means clustering (**?**),
- 2nd round sampling (**?**),

Our interest in this tutorials is in producing predictions (maps) of the target variable by employing regression / correlation between the target variable and multitude of features (raster layers), and where various Machine Learning techniques are used for training and prediction.

Once we collect enough training points in an area we can overlay points and GIS layers to produce a **regression matrix** or **classification matrix**, and which can then be used to generate spatial predictions i.e. produce maps. As a state-of-the-art we use the mlr framework for Ensemble Machine Learning as the key Machine Learning framework for predictive mapping. For an introduction to Ensemble Machine Learning for Predictive Mapping please refer to this tutorial[10].

1.4 Ebergotzen dataset

To test various sampling and mapping algorithms, we can use the Ebergotzen dataset available also via the **plotKML** package (**?**):

[10] https://gitlab.com/openlandmap/spatial-predictions-using-eml

```
set.seed(100)
library(plotKML)
library(sp)
library(viridis)
#> Loading required package: viridisLite
library(raster)
library(ggplot2)
data("eberg_grid25")
gridded(eberg_grid25) <- ~x+y
proj4string(eberg_grid25) <- CRS("+init=epsg:31467")
```

This dataset is described in detail in **?**. It is a soil survey dataset with ground observations and measurements of soil properties including soil types. The study area is a perfect square 10×10 km in size.

We have previously derived number of additional DEM parameters directly using SAGA GIS (**?**) and which we can add to the list of covariates:

```
eberg_grid25 = cbind(eberg_grid25, readRDS("./extdata/eberg_dtm_25m.rds"))
names(eberg_grid25)
#>  [1] "DEMTOPx"        "HBTSOLx"        "TWITOPx"        "NVILANx"
#>  [5] "eberg_dscurv"   "eberg_hshade"   "eberg_lsfactor" "eberg_pcurv"
#>  [9] "eberg_slope"    "eberg_stwi"     "eberg_twi"      "eberg_vdepth"
#> [13] "PMTZONES"
```

so a total of 11 layers at 25-m spatial resolution, from which two layers (`HBTSOLx` and `PMTZONES`) are factors representing soil units and parent material units. Next, for further analysis, and to reduce data overlap, we can convert all primary variables to (numeric) principal components using:

```
eberg_spc = landmap::spc(eberg_grid25[-c(2,3)])
#> Converting PMTZONES to indicators...
#> Converting covariates to principal components...
```

which gives the a total of 14 PCs. The patterns in the PC components reflect a complex combination of terrain variables, lithological discontinuities (`PMTZONES`) and surface vegetation (`NVILANx`):

```
spplot(eberg_spc@predicted[1:3], col.regions=SAGA_pal[[1]])
```

1.5 Simple Random Sampling

To generate a SRS with e.g. 100 points we can use the **sp** package `spsample` method:

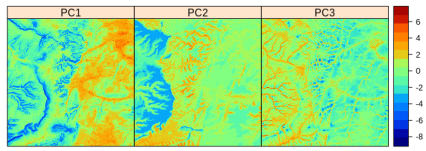


Fig. 1.2 Principal Components derived using Ebergotzen dataset.

```
rnd <- spsample(eberg_grid25[1], type="random", n=100)
```

```
plot(raster(eberg_spc@predicted[1]), col=SAGA_pal[[1]])
points(rnd, pch="+")
```

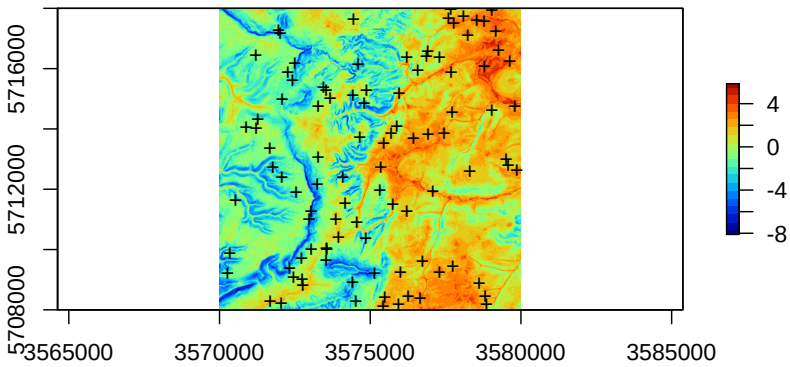


Fig. 1.3 An example of a Simple Random Sample (SRS).

This sample is generated purely based on the spatial domain, the feature space is completely ignored / not taken into account, hence we can check how well do these points represent the feature space using a density plot:

```
ov.rnd = sp::over(rnd, eberg_spc@predicted[,1:2])
library(hexbin)
library(grid)
reds = colorRampPalette(RColorBrewer::brewer.pal(9, "YlOrRd")[-1])
hb <- hexbin(eberg_spc@predicted@data[,1:2], xbins=60)
```

```
p <- plot(hb, colramp = reds, main='PCA Ebergotzen SRS')
pushHexport(p$plot.vp)
grid.points(ov.rnd[,1], ov.rnd[,2], pch="+")
```

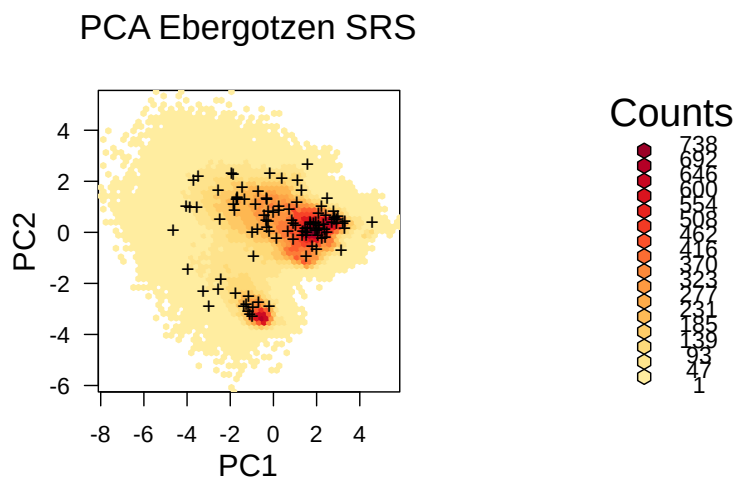


Fig. 1.4 Distribution of the SRS points from the previous example in the feature space.

Visually, we do not directly see from Fig. **??** that the generated SRS maybe misses some important feature space, however if we zoom in, then we can notice that some parts of feature space (in this specific randomization) with high density are somewhat under-represented. Imagine if we reduce number of sampling points then we run even higher risk of missing some areas in the feature space by using SRS.

Next, we are interested in evaluating the occurrence probability of the SRS points based on the PCA components. To derive the occurrence probability we can use the **maxlike** package method (**?**):

```
fm.cov <- stats::as.formula(paste("~", paste(names(eberg_spc@predicted[1:4]), collapse="+")))
ml <- maxlike::maxlike(formula=fm.cov, rasters=raster::stack(eberg_spc@predicted[1:4]),
                       points=rnd@coords, method="BFGS", removeDuplicates=TRUE, savedata=TRUE)
ml.prob <- predict(ml)
```

```
plot(ml.prob)
points(rnd@coords, pch="+")
```

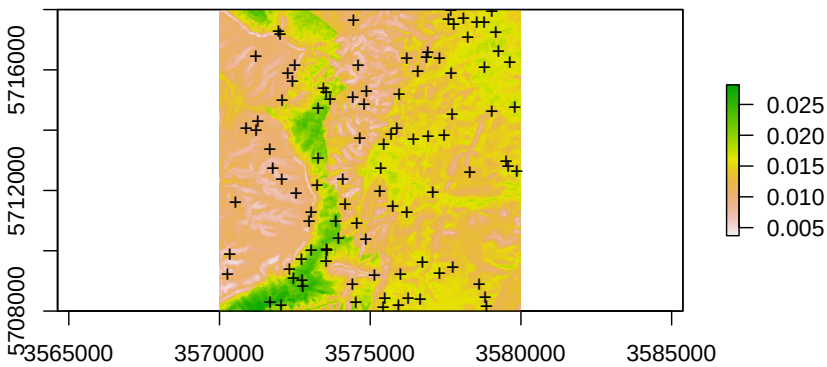


Fig. 1.5 Occurrence probability for SRS derived using the maxlike package.

Note: for the sake of reducing the computing intensity we focus on the first four PCs. In practice, feature space analysis can be quite computational and we recommend using parallelized versions within an High Performance Computing environments to run such analysis.

Fig. **??** shows that, by accident, some parts of the feature space might be somewhat under-represented (areas with low probability of occurrence on the map). Note however that occurrence probability for this dataset is overall *very low* (<0.05), indicating that distribution of points is not much correlated with the features. This specific SRS is thus probably satisfactory also for feature space analysis: the SRS points do not seem to group (which would in this case be by accident) and hence maxlike gives very low probability of occurrence.

We can repeat SRS many times and then see if the clustering of points gets more problematic, but as you can image, in practice for large number of samples it is a good chance that all features would get well represented also in the feature space even though we did not include feature space variables in the production of the sampling plan.

We can now also load the actual points collected for the Ebergotzen case study:

```
data(eberg)
eberg.xy <- eberg[,c("X","Y")]
coordinates(eberg.xy) <- ~X+Y
proj4string(eberg.xy) <- CRS("+init=epsg:31467")
ov.xy = sp::over(eberg.xy, eberg_grid25[1])
eberg.xy = eberg.xy[!is.na(ov.xy$DEMTOPx),]
sel <- sample.int(length(eberg.xy), 100)
eberg.smp = eberg.xy[sel,]
```

To quickly estimate spread of points in geographical and feature spaces, we can also use the function `spsample.prob` that calls both the kernel density function from the **spatstat**[11] package, and derives the probability of occurrence using the **maxlike**[12] package:

```
iprob <- landmap::spsample.prob(eberg.smp, eberg_spc@predicted[1:4])
#> Deriving kernel density map using sigma 1010 ...
#> Deriving inclusion probabilities using MaxLike analysis...
```

In this specific case, the actual sampling points are much more clustered, so if we plot the two occurrence probability maps derived using maxlike next to each other we get:

```
op <- par(mfrow=c(1,2))
plot(raster(iprob$maxlike), zlim=c(0,1))
points(eberg.smp@coords, pch="+")
plot(ml.prob, zlim=c(0,1))
points(rnd@coords, pch="+")
par(op)
```

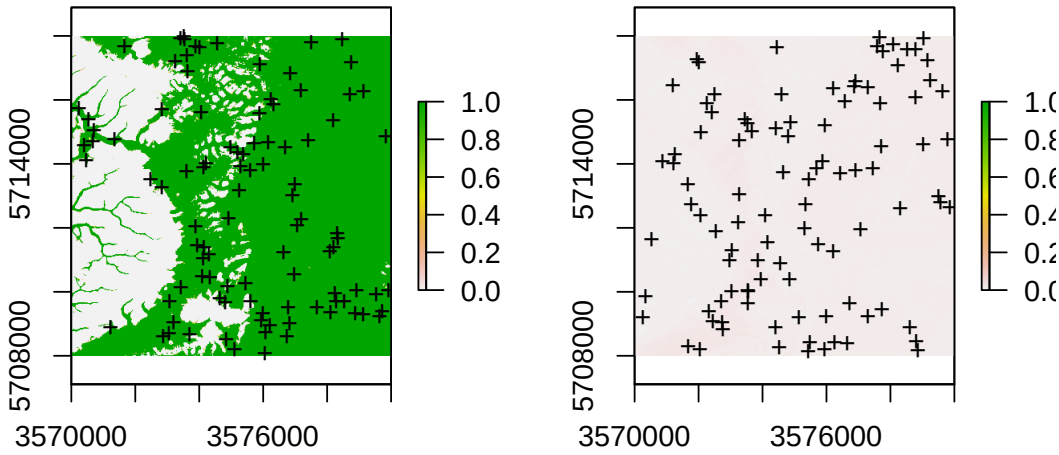


Fig. 1.6 Comparison occurrence probability for actual and SRS samples derived using the maxlike package.

[11] https://rdrr.io/cran/spatstat.core/man/density.ppp.html
[12] https://rdrr.io/github/rbchan/maxlike/man/maxlike-package.html

The map on the left clearly indicates that most of the sampling points are basically preferentially located in the plain area, while the hillands are systematically under-sampled. This we can also cross-check by reading the description of the dataset in **?**:

- the Ebergotzen soil survey points focus on agricultural land only,
- no objective sampling design has been used, hence some points are clustered,

This is also clearly visible from the *feature map* plot where one part of the feature space seem to be completely omitted from sampling:

```
ov2.rnd = sp::over(eberg.smp, eberg_spc@predicted[,1:2])
p <- plot(hb, colramp = reds, main='PCA Ebergotzen actual')
pushHexport(p$plot.vp)
grid.points(ov2.rnd[,1], ov2.rnd[,2], pch="+")
```

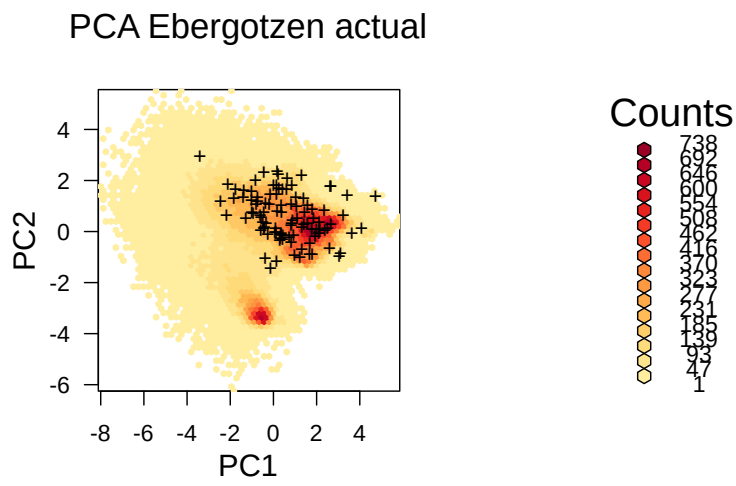


Fig. 1.7 Distribution of the actual survey points from the previous example displayed in the feature space.

1.6 Latin Hypercube Sampling

In the previous example we have shown how to implement SRS and then also evaluate it against feature layers. Often SRS represent very well feature space so it can be directly used for Machine Learning and with a guarantee of not making too much bias
in predictions. To avoid, however, risk of missing out some parts of the feature space, and also

to try to optimize allocation of points, we can generate a sample using the **Latin Hypercube Sampling**[13] (LHS) method. In a nutshell, LHS methods are based on dividing the **Cumulative Density Function** (CDF) into n equal partitions, and then choosing a random data point in each partition, consequently:

- CDF of LHS samples matches the CDF of population (hence unbiased representation),
- Extrapolation in the feature space should be minimized,

Latin Hypercube and sampling optimization using LHS is explained in detail in **?** and **?**. For **?**, LHS is just a special case of **balanced sampling**[14] i.e. sampling based on allocation in feature space. The lhs package[15] also contains numerous examples of how to implement LHS for (non-spatial) data.

Here we use an implementation of the LHS available in the **clhs** package (**?**):

```
library(clhs)
rnd.lhs = clhs::clhs(eberg_spc@predicted[1:4], size=100, iter=100, progress=FALSE)
```

This actually implements the so-called *"Conditional LHS"* (**?**), and can get quite computational for large stack of rasters, hence we manually limit the number of iterations to 100.

We can plot the LHS sampling plan:

```
plot(raster(eberg_spc@predicted[1]), col=SAGA_pal[[1]])
points(rnd.lhs@coords, pch="+")
```

```
p <- plot(hb, colramp = reds, main='PCA Ebergotzen LHS')
pushHexport(p$plot.vp)
grid.points(rnd.lhs$PC1, rnd.lhs$PC2, pch="+")
```

Although in principle we might not see any difference in the point pattern between SRS and LHS, the feature space plot clearly shows that LHS covers systematically feature space map, i.e. we would have a relatively low risk of missing out some important features as compared to Fig. **??**.

Thus the main advantages of the LHS are:

- it ensures that feature space is represented systematically i.e. it is optimized for Machine Learning using the specific feature layers;

[13] https://en.wikipedia.org/wiki/Latin_hypercube_sampling
[14] http://www.antongrafstrom.se/balancedsampling/
[15] https://github.com/bertcarnell/lhs

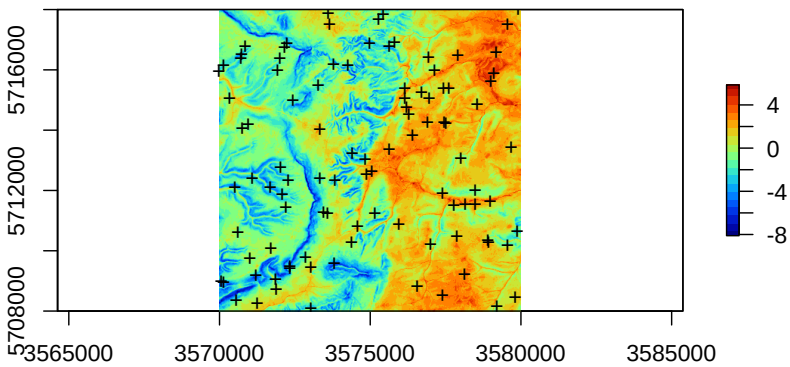


Fig. 1.8 An example of a Latin Hypercube Sample (LHS).

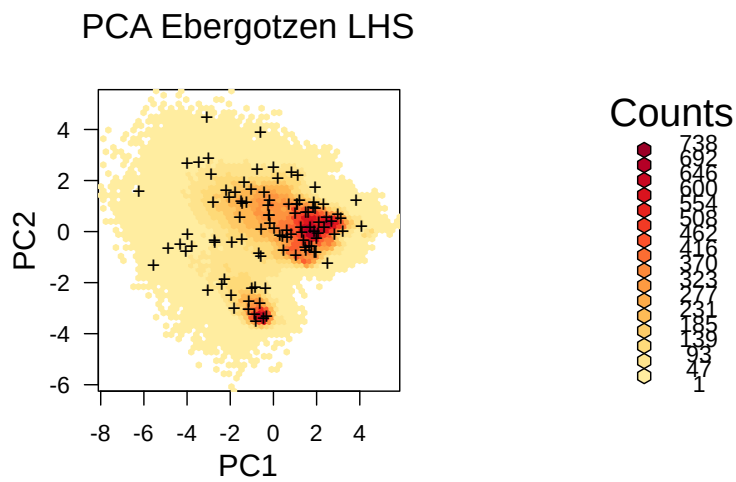


Fig. 1.9 Distribution of the LHS points from the previous example displayed in the feature space.

- it is an **Independent Identically Distributed (IID)**[16] sampling design;
- thanks to the **clhs** package, also the survey costs raster layer can be integrated to still keep systematic spread, but reduce survey costs as much as possible (**?**);

1.7 Feature Space Coverage Sampling

The **Feature Space Coverage Sampling** (FSCS) is described in detail in **?**. In a nutshell, FSCS aims at optimal coverage of the feature space which is achieved by minimizing the average distance of the population units (raster cells) to the nearest sampling units in the feature space represented by the raster layers (**?**).

To produce FSCS point sample we can use function `kmeanspp` of package **LICORS** (**?**). First we partition the feature space cube to e.g. 100 clusters. We then select raster cells with the shortest scaled Euclidean distance in covariate-space to the centers of the clusters as the sampling units:

```
library(LICORS)
library(fields)
fscs.clust <- kmeanspp(eberg_spc@predicted@data[,1:4], k=100, iter.max=100)
D <- fields::rdist(x1 = fscs.clust$centers, x2 = eberg_spc@predicted@data[,1:4])
units <- apply(D, MARGIN = 1, FUN = which.min)
rnd.fscs <- eberg_spc@predicted@coords[units,]
```

Note: the k-means++ algorithm is of most interest for small sample sizes: *"for large sample sizes the extra time needed for computing the initial centers can become substantial and may not outweigh the larger number of starts that can be afforded with the usual k-means algorithm for the same computing time"* (**?**). The `kmeanspp` algorithm from the LICORS package is unfortunately quite computational and is in principle not recommended for large grids / to generate large number of samples. Instead, we recommend clustering feature space using the `h2o.kmeans` function from the h2o package[17], which is also suitable for larger datasets with computing running in parallel:

```
library(h2o)
h2o.init(nthreads = -1)
#>
#> H2O is not running yet, starting it now...
#>
#> Note:  In case of errors look at the following log files:
#>     /tmp/RtmpxDss9K/file64a23b831553/h2o_tomislav_started_from_r.out
#>     /tmp/RtmpxDss9K/file64a27ea38241/h2o_tomislav_started_from_r.err
#>
```

[16] https://xzhu0027.gitbook.io/blog/ml-system/sys-ml-index/learning-from-non-iid-data
[17] https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/k-means.html

```
#>
#> Starting H2O JVM and connecting: .. Connection successful!
#>
#> R is connected to the H2O cluster:
#>     H2O cluster uptime:         1 seconds 950 milliseconds
#>     H2O cluster timezone:       Europe/Amsterdam
#>     H2O data parsing timezone:  UTC
#>     H2O cluster version:        3.30.0.1
#>     H2O cluster version age:    1 year, 9 months and 22 days !!!
#>     H2O cluster name:           H2O_started_from_R_tomislav_vru837
#>     H2O cluster total nodes:    1
#>     H2O cluster total memory:   15.71 GB
#>     H2O cluster total cores:    32
#>     H2O cluster allowed cores:  32
#>     H2O cluster healthy:        TRUE
#>     H2O Connection ip:          localhost
#>     H2O Connection port:        54321
#>     H2O Connection proxy:       NA
#>     H2O Internal Security:      FALSE
#>     H2O API Extensions:         Amazon S3, XGBoost, Algos, AutoML, Core V3, TargetEncoder, Core V4
#>     R Version:                  R version 4.0.2 (2020-06-22)
#> Warning in h2o.clusterInfo():
#> Your H2O cluster version is too old (1 year, 9 months and 22 days)!
#> Please download and install the latest version from http://h2o.ai/download/
df.hex <- as.h2o(eberg_spc@predicted@data[,1:4], destination_frame = "df")
#>
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
km.nut <- h2o.kmeans(training_frame=df.hex, k=100, keep_cross_validation_predictions = TRUE)
#>
  |
  |                                                                      |   0%
  |
  |=======                                                               |  10%
  |
  |======================================================================| 100%
#km.nut
```

Note: in the example above, we have manually set the number of clusters to 100, but the number of clusters could be also derived using some optimization procedure[18]. Next, we predict the clusters and plot the output:

[18] https://www.r-bloggers.com/2019/01/10-tips-for-choosing-the-optimal-number-of-clusters/

```
m.km <- as.data.frame(h2o.predict(km.nut, df.hex, na.action=na.pass))
#>
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

We can save the class centers (if needed for any further analysis):

```
class_df.c = as.data.frame(h2o.centers(km.nut))
names(class_df.c) = names(eberg_spc@predicted@data[,1:4])
str(class_df.c)
#> 'data.frame':    100 obs. of  4 variables:
#>  $ PC1: num  -1.01 -2.09 -3.01 -4.71 4 ...
#>  $ PC2: num  0.912 2.547 -3.995 3.034 0.631 ...
#>  $ PC3: num  1.12 5.43 -3.42 -1.42 2.45 ...
#>  $ PC4: num  -1.718 4.709 -2.428 2.168 0.654 ...
#write.csv(class_df.c, "NCluster_100_class_centers.csv")
```

To select sampling points we use the minimum distance to class centers (**?**):

```
D <- fields::rdist(x1 = class_df.c, x2 = eberg_spc@predicted@data[,1:4])
units <- apply(D, MARGIN = 1, FUN = which.min)
rnd.fscs <- eberg_spc@predicted@coords[units,]
```

```
plot(raster(eberg_spc@predicted[1]), col=SAGA_pal[[1]])
points(rnd.fscs, pch="+")
```

Visually, FSCS seem to add higher spatial density of points in areas where there is higher complexity. The `h2o.kmeans` algorithm stratifies area into most possible homogeneous units (in the example above, large plains in the right part of the study area are relatively homogeneous, hence the sampling intensity in there areas drops significantly when visualized in the geographical space), and the points are then allocated per each strata.

```
p <- plot(hb, colramp = reds, main='PCA Ebergotzen FSCS')
pushHexport(p$plot.vp)
grid.points(eberg_spc@predicted@data[units,"PC1"], eberg_spc@predicted@data[units,"PC2"], pch="+")
```

The FSCS sampling pattern in feature space looks almost as grid sampling in feature space. FSCS seems to put more effort on sampling at the edges on the feature space in comparison to LHS and

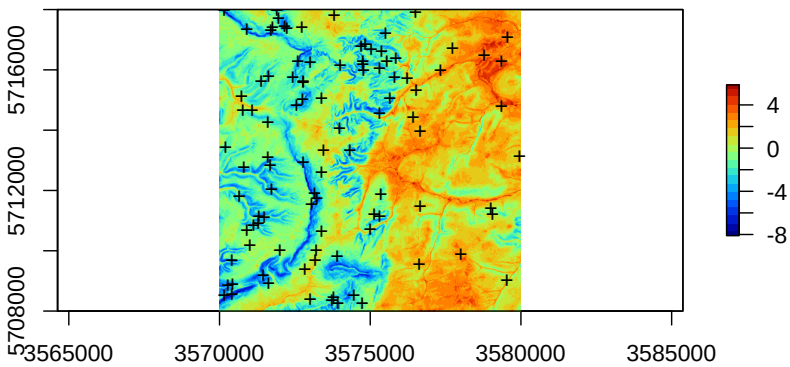


Fig. 1.10 An example of a Feature Space Coverage Sampling (FSCS).

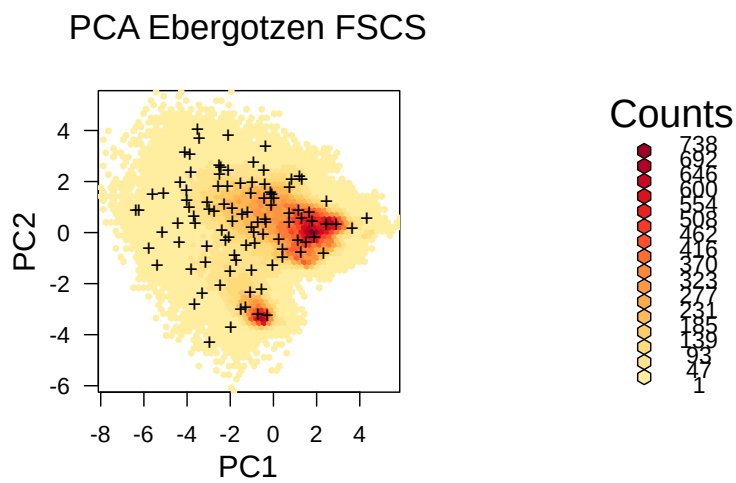


Fig. 1.11 Distribution of the FSCS points from the previous example displayed in the feature space.

SRS, and hence can be compared to classical response surface designs such as D-optimal designs[19] (**?**).

```
h2o.shutdown(prompt = FALSE)
```

1.8 Summary points

In the previous examples we have shown differences between SRS, LHS and FSCS. SRS and LHS are IID sampling methods and as long as the number of samples is large and the study area is complex, it is often difficult to notice or quantify any under- or over-sampling. Depending on which variant of FSCS we implement, FSCS can result in higher spatial spreading especially if a study area consists of a combination of a relatively homogeneous and complex terrain units.

The Ebergotzen dataset (existing point samples) clearly shows that the *"convenience surveys"* can show significant clustering and under-representation of feature space (Fig. **??**). Consequently, these sampling bias could lead to:

- *Bias in estimating regression parameters* i.e. over-fitting;
- *Extrapolation problems* due to under-representation of feature space;
- *Bias* in estimating population parameters of the target variable;

The first step to deal with these problems is to detect them, second is to try to implement a strategy that prevents from model over-fitting. Some possible approaches to deal with such problems are addressed in the second part of the tutorial.

LHS and FSCS are recommended sampling methods if the purpose of sampling is to build regression or classification models using multitude of (terrain, climate, land cover etc) covariate layers. A generalization of LHS is the balanced sampling where users can select even variable inclusion probabilities (**??**).

? compared LHS to FSCS for mapping soil types and concluded that FSCS results in better mapping accuracy, most likely because FSCS spreads points better in feature space and hence in their case studies that seem to have helped with producing more accurate predictions. **?** also report that LHS helps improve accuracy only for the large size of points.

The `h2o.kmeans` algorithm is suited for large datasets, but nevertheless to generate 100 clusters using large number of raster layers could become RAM consuming and is maybe not practical for operational sampling. An alternative would be to reduce number of clusters and select multiple points per cluster.

In the case of doubt which method to use LHS or FSCS, we recommend the following simple rules of thumb:

[19] https://en.wikipedia.org/wiki/Optimal_design

- If your dataset contains relatively smaller-size rasters and the targeted number of sampling points is relatively small (e.g. 1000), we recommend using the FSCS algorithm;
- If your project requires large number of sampling points (100), then you should probably consider using the LHS algorithm;

In general, as the number of sampling points starts growing, differences between SRS (no feature space) and LHS becomes minor, which can also be witnessed visually (basically it becomes difficult to tell the difference between the two). SRS could, however, by accident miss some important parts of the feature space, so in principle it is still important to use either LHS or FSCS algorithms to prepare sampling locations for ML where objective is to fit regression and/or classification models using ML algorithms.

To evaluate potential sampling clustering and pin-point under-represented areas one can run multiple diagnostics:

1. In *geographical space*:
   (a) kernel density analysis[20] using the spatstat package, then determine if some parts of the study area have systematically higher density;
   (b) testing for **Complete Spatial Randomness** (**?**) using e.g. spatstat.core::mad.test[21] and/or dbmss::Ktest[22];
2. In *feature space*:
   (a) occurrence probability analysis using the maxlike package[23];
   (b) **unsupervised clustering** of the feature space using e.g. `h2o.kmeans`, then determining if any of the clusters are significantly under-represented / under-sampled;
   (c) estimating **Area of Applicability** based on similarities between training prediction and feature spaces (**?**);

Plotting generated sampling points both on a map and *feature space map* helps detect possible extrapolation problems in a sampling design (Fig. **??**). If you detect problems in feature space representation based on an existing point sampling set, you can try to reduce those problems by adding additional samples e.g. through **covariate space infill sampling** (**?**) or through 2nd round sampling and then re-analysis. Such methods are discussed in further chapters.

[20] https://rdrr.io/cran/spatstat.core/man/density.ppp.html
[21] https://rdrr.io/cran/spatstat.core/man/dclf.test.html
[22] https://rdrr.io/cran/dbmss/man/Ktest.html
[23] https://rdrr.io/github/rbchan/maxlike/man/maxlike-package.html

Chapter 2
Resampling methods for Machine Learning

You are reading the work-in-progress Spatial Sampling and Resampling for Machine Learning. This chapter is currently draft version, a peer-review publication is pending. You can find the polished first edition at https://opengeohub.github.io/spatial-sampling-ml/.

2.1 Resampling and Cross-Validation

In the previous examples we have demonstrated how to prepare sampling designs for your own study area assuming no other point data is available. We have also demonstrated how to run some sampling representation diagnostics to detect potential problems especially in the feature space. In the next sections we will focus on how to use different **resampling** methods i.e. **cross-validation strategies** (**?**) to help reduce problems such as:

- *overfitting* i.e. producing models that are biased and/or over-optimistic;
- *missing out covariates* that are important but possibly *shadowed* by the covariates over-selected due to overfitting;
- *producing poor extrapolation* i.e. generating artifacts or blunders in predictions;
- *over-/under-estimating mapping accuracy* i.e. producing a biased estimate of model performance;

Resamping methods are discussed in detail in **?**, **?** and **?**, and is also commonly implemented in many statistical and machine learning packages such as the caret[1] or mlr[2]. Spatial resampling methods are discussed in detail in **?**.

[1] https://topepo.github.io/caret/
[2] https://mlr3.mlr-org.com/

For an introduction to Cross-Validation please refer to this tutorial[3] and the **"Data Analysis and Prediction Algorithms with R"**[4] chapters on cross validation. For an introduction to **spatial Cross-Validation** refer to the **"Geocomputation with R"**[5] book.

It can be said that, in general, purpose of Machine Learning for predictive mapping is to try to produce **Best Unbiased Predictions** (BUPS) of the target variable and the associated uncertainty (e.g. prediction error). BUPS is commonly implemented through: (1) selecting the Best Unbiased Predictor (**?**), (2) selecting the optimal subset of covariates and model parameters (usually by iterations), (3) applying predictions and providing an estimate of the prediction uncertainty i.e. estimate of the prediction errors / prediction intervals for a given probability distribution. In this tutorial the path to BUPS we use includes:

- **Ensemble Machine Learning** using stacking approach with 5-fold Cross-Validation with a meta-learner i.e. an independent model correlating competing base-learners with the target variable (**??**),
- Estimating prediction errors using **quantile regression Random Forest** (**?**),

The reasoning for using Ensemble ML for predictive mapping is explained in detail in this tutorial[6], and it's advantages for minimizing extrapolation effects in this medium post[7].

2.2 Resampling training points using declustering

In the previous example we have shown that the actual soil survey points for Ebergotzen are somewhat spatially *clustered*, they under-sample forest areas / hillands. In statistical term these sampling points are geographically unbalanced i.e. non-IID. If we plot the original sampling points (N=2780) we see high spatial clustering of samples:

```
library(rgdal)
library(raster)
library(plotKML)
library(landmap)
library(mlr)
data("eberg_grid25")
gridded(eberg_grid25) <- ~x+y
proj4string(eberg_grid25) <- CRS("+init=epsg:31467")
eberg_grid25 = cbind(eberg_grid25, readRDS("./extdata/eberg_dtm_25m.rds"))
eberg_spc = landmap::spc(eberg_grid25[-c(2,3)])
#> Converting PMTZONES to indicators...
```

[3] https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right

[4] https://rafalab.github.io/dsbook/cross-validation.html

[5] https://geocompr.robinlovelace.net/spatial-cv.html

[6] https://opengeohub.github.io/spatial-prediction-eml/

[7] https://medium.com/nerd-for-tech/extrapolation-is-tough-for-trees-tree-based-learners-combining-learners-of-different-type-makes-659187a6f58d

```
#> Converting covariates to principal components...
data(eberg)
eberg.xy <- eberg[,c("X","Y")]
coordinates(eberg.xy) <- ~X+Y
proj4string(eberg.xy) <- CRS("+init=epsg:31467")
ov.xy = sp::over(eberg.xy, eberg_grid25[1])
eberg.xy = eberg.xy[!is.na(ov.xy$DEMTOPx),]
```

```
plot(raster(eberg_spc@predicted[1]), col=SAGA_pal[[1]])
points(eberg.xy, pch="+", cex=.5)
```

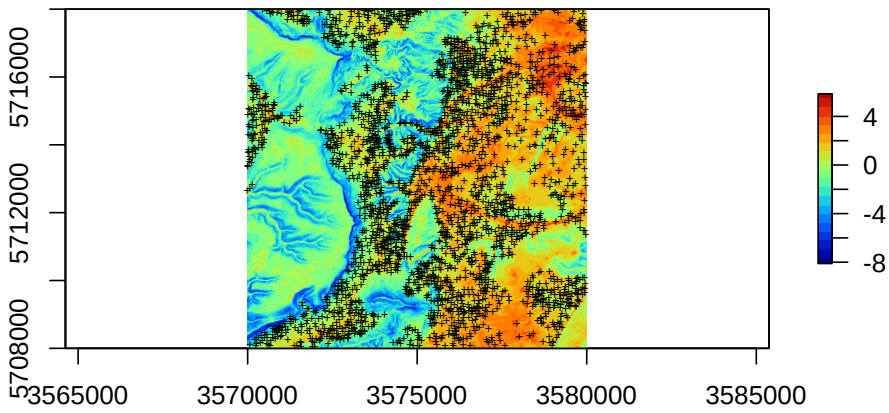


Fig. 2.1 All sampling points available for Ebergotzen case study.

If we ignore that property of the data and directly fit a predictive model for e.g. top-soil clay content using e.g. random forest (**?**) we get:

```
library(ranger)
rm.eberg = cbind(eberg[!is.na(ov.xy$DEMTOPx),], sp::over(eberg.xy, eberg_spc@predicted))
cly.fm = as.formula(paste0("CLYMHT_A ~ ", paste0("PC", 1:13, collapse = "+")))
sel.cly = complete.cases(rm.eberg[,all.vars(cly.fm)])
rm.cly = rm.eberg[sel.cly,]
rf.cly = ranger::ranger(cly.fm, data=rm.cly)
```

```
rf.cly
#> Ranger result
#>
#> Call:
#>  ranger::ranger(cly.fm, data = rm.cly)
#>
#> Type:                             Regression
#> Number of trees:                  500
#> Sample size:                      2776
#> Number of independent variables:  13
#> Mtry:                             3
#> Target node size:                 5
#> Variable importance mode:         none
#> Splitrule:                        variance
#> OOB prediction error (MSE):       53.23538
#> R squared (OOB):                  0.6081801
```

This shows an RMSE of about 7.3% and an R-square of about 0.61. The problem of this accuracy measure is that with this Random Forest model we ignore spatial clustering of points, hence both the model and the accuracy metric could be over-optimistic (**??**). Because we are typically interested in how does the model perform over the WHOLE area of interest, not only in comparison to out-of-bag points, to reduce overfitting or any bias in the BUPS we need to apply some adjustments.

Strategy #1 for producing more objective estimate of model parameters is to resample sampling points by forcing as much as possible equal sampling intensity (hence mimicking the SRS), then observe performance of the model accuracy. We can implement such spatial resampling using the `sample.grid` function, which basically resamples the existing point samples with an objective of producing a sample more similar to SRS. This type of subsetting can be run `M` times and then an ensemble model can be produced in which each individual model is based on spatially balanced samples. These are not true SRS samples but we can refer to them as the **pseudo-SRS samples** as they would probably pass all Spatial Randomness tests.

In R we can implement spatial resampling using the following three steps. First, we generate e.g. 10 random subsets where the sampling intensity of points is relatively homogeneous:

```
eberg.sp = SpatialPointsDataFrame(eberg.xy, rm.eberg[c("ID","CLYMHT_A")])
sub.lst = lapply(1:10, function(i){landmap::sample.grid(eberg.sp, c(500, 500), n=2)})
```

This randomly subsets each 500-m block to max 2 points i.e. trims down the densely sampled points to produce a relatively balanced spatial sampling intensity. We can check that the training point sample looks more like a SRS or similar.

```
l1 <- list("sp.points", sub.lst[[1]]$subset, pch="+", col="black")
spplot(sub.lst[[1]]$grid, scales=list(draw=TRUE),
   col.regions="grey", sp.layout=list(l1), colorkey=FALSE)
```

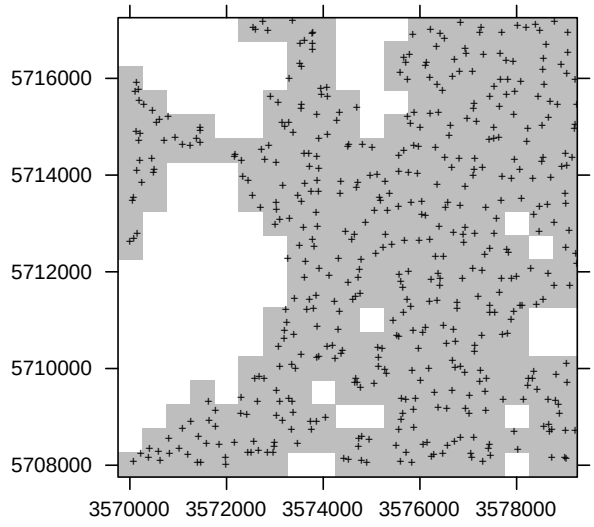


Fig. 2.2 Resampling original points using 'sample.grid' function, which produces a sample with similar properties such as SRS.

Second, we can fit a list of random forest models using 10 random draws mimicking some IID sampling:

```
rf.cly.lst = lapply(1:length(sub.lst), function(i){
        x <- rm.eberg[which(rm.eberg$ID %in% sub.lst[[i]]$subset$ID),];
        x <- x[complete.cases(x[,all.vars(cly.fm)]),];
        y <- ranger::ranger(cly.fm, data=x, num.trees = 50);
        return(y)
      }
)
```

Third, we produce an Ensemble model that combines all predictions by simple averaging (**?**). To produce final predictions, we can use simple averaging because all subsets are symmetrical i.e. have exactly the same inputs and settings, hence all models fitted have equal importance for the ensemble model.

The out-of-bag accuracy of ranger now shows a somewhat higher RMSE, which is also probably more realistic:

```
mean(sapply(rf.cly.lst, function(i){sqrt(i[["prediction.error"]])}))
#> [1] 8.003919
```

In summary, the actual error is most likely about 20% higher than if we ignore clustering and the previous model was likely over-optimistic. The model is too much influenced by the clustered point samples and this should be taken into account during model fitting (**??**). If we visually compare the predictions between the original and ensemble models we see:

```
cn = rf.cly$forest$independent.variable.names
pred.cly = predict(rf.cly, eberg_spc@predicted@data[,cn])
eberg_grid25$pred.cly.rf = pred.cly$predictions
pred.cly.lst = lapply(rf.cly.lst, function(i){
    predict(i, eberg_spc@predicted@data[,cn])$predictions })
eberg_grid25$pred.cly.erf = rowMeans(do.call(cbind, pred.cly.lst), na.rm=TRUE)
```

```
#zlim.cly = quantile(rm.eberg$CLYMHT_A, c(0.05, 0.95), na.rm=TRUE)
spplot(eberg_grid25[c("pred.cly.rf", "pred.cly.erf")], col.regions=SAGA_pal[[1]])
```

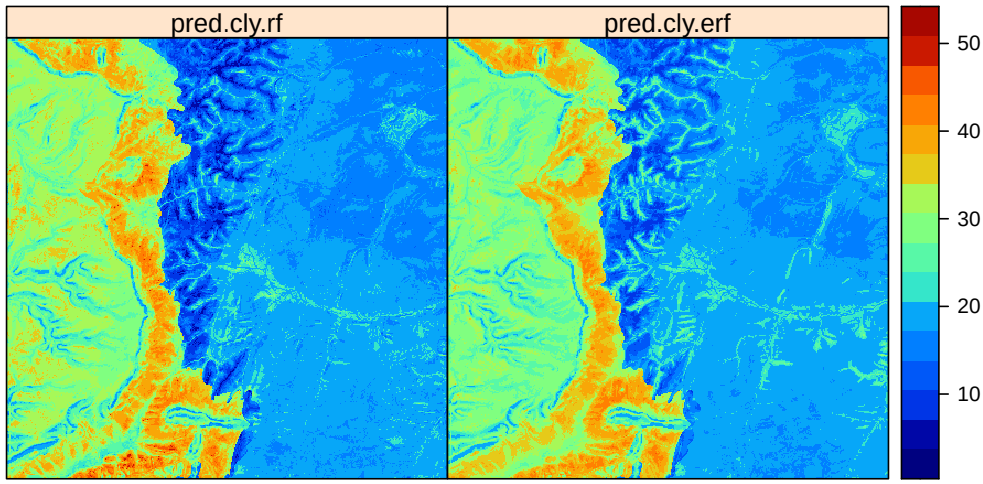


Fig. 2.3 Predictions of clay content: (left) ignoring spatial clustering effect on model, (right) after de-clustering of points.

Overall, the difference is small but there is certainly visible difference in predictions. To the end users we would probably suggest to use `pred.cly.erf` (predictions produced using the de-clustered points) map because the other model completely ignores spatial clustering of points and this could have resulted in bias estimate of the regression parameters. This solution to producing predictions is fully scalable and relatively easy to implement, it only requires from user to decide on (1) size

of the block for subsampling, (2) max sampling intensity per block. In practice, both could be determined by iterations.

2.3 Weighted Machine Learning

Note that the function `grid.sample` per definition draws points that are relatively isolated (Fig. **??**) with higher probability. Hence, a more generic approach (Strategy #2) from declustering, is to use the `case.weights` parameter to instruct `ranger` to put more emphasis on isolated points / remove impact of clustered points. The weighted estimation of model parameters is common in regression, and also in spatial statistics (see e.g. the nlme::gls[8] Generalized Least Square (GLS) function). Theoretical basis for GLS / weighted regression is that the points that are clustered might also be spatially correlated, and that means that they would introduce bias in estimation of the regression parameters. Ideally, regression residuals should be uncorrelated and uniform, hence a correction is needed that helps ensure these properties.

Weighted Machine Learning where bias in the sampling intensity is incorporated in the modeling can be implemented in two steps. First, we derive the occurrence probability (0–1) using the `spsample.prob` method:

```
iprob.all <- landmap::spsample.prob(eberg.sp, eberg_spc@predicted[1:4])
#> Deriving kernel density map using sigma 157 ...
#> Deriving inclusion probabilities using MaxLike analysis...
```

```
plot(raster(iprob.all$prob), zlim=c(0,1))
points(iprob.all$observations, pch="+", cex=.5)
```

Second, we fit a model using all points, but this time we set `case.weights` to be reversely proportional to probability of occurrence (hence points with lower occurrence probability get higher weights):

```
ov.weigths = 1/sp::over(eberg.xy, iprob.all$prob)[]
rf.clyI = ranger::ranger(cly.fm, data=rm.cly, case.weights = ov.weigths[sel.cly,])
```

Weighted regression[9] is a common technique in statistics and in this case the weights are used to helps reduce clustering effect i.e. give weights to points proportionally to the sampling bias. This method is in principle similar to the nlme::gls[10] Generalized Least Square (GLS) procedure, but with the difference that we do not estimate any spatial autocorrelation structure, but instead incorporate the probability of occurrence to reduce effect of spatial clustering.

[8] https://stat.ethz.ch/R-manual/R-devel/library/nlme/html/gls.html

[9] https://www.statology.org/weighted-least-squares-in-r/

[10] https://stat.ethz.ch/R-manual/R-devel/library/nlme/html/gls.html

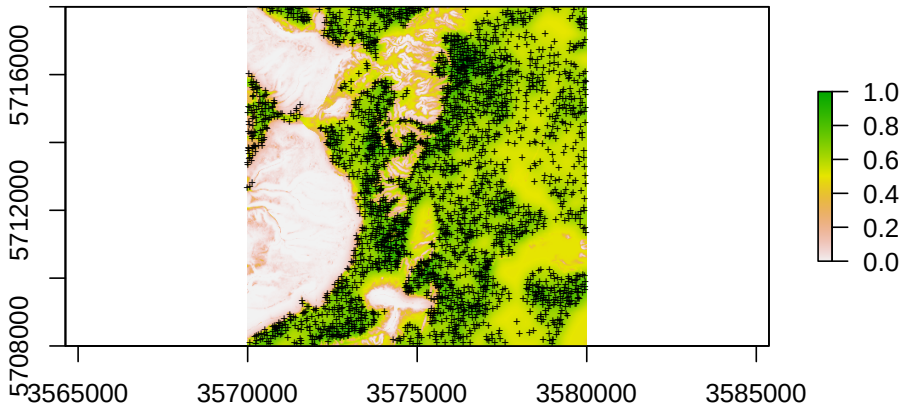


Fig. 2.4 Occurrence probability for existing point samples for the Ebergotzen case study derived as an average between the kernel density and maxlike occurrence probabilities.

2.4 Resampling using Ensemble ML

Another approach to improve generating BUPS from clustered point data is to switch to Ensemble ML i.e. use a multitude of ML methods (so called **base-learners**), then estimate final predictions using robust resampling and blocking. Ensemble ML has shown to help increase mapping accuracy, but also helps with reducing *over-shooting* effects due to extrapolation[11].

One way to reduce effects of point clustering for predictive is to use *spatial blocking* i.e. to make sure that spatially clustered points are not used both for training and internal validation (**?**). First, we need to define a spatial grid that we will use as a blocking parameter. We set here arbitrarily size of spatial blocks to 500-m, in practice the block size can be determined more objectively by e.g. analyzing at which distances is clustering reduced:

```
grd <- sp::GridTopology(cellcentre.offset=eberg_grid25@bbox[,1], cellsize=rep(500,2),
                        cells.dim=c(ceiling(abs(diff(eberg_grid25@bbox[1,])/500))+1,
                        ceiling(abs(diff(eberg_grid25@bbox[2,])/500))+1))
r.sp <- sp::SpatialGridDataFrame(grd, proj4string = eberg_grid25@proj4string,
                     data=data.frame(gid=1:(grd@cells.dim[1] * grd@cells.dim[2])))
id <- sp::over(eberg.xy, r.sp)$gid
#summary(as.factor(id))
```

[11] https://medium.com/nerd-for-tech/extrapolation-is-tough-for-trees-tree-based-learners-combining-learners-of-different-type-makes-659187a6f58d

```
plot(r.sp)
points(eberg.xy, pch="+", cex=.5)
```

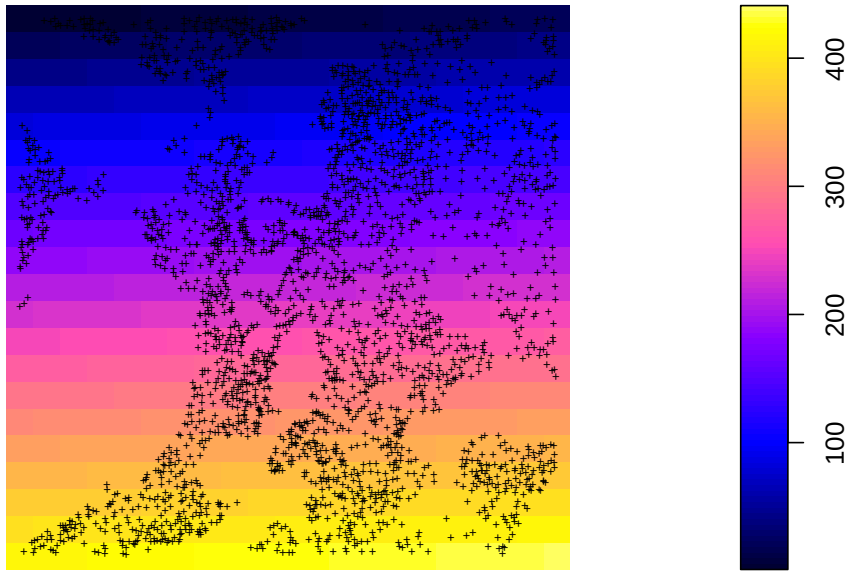


Fig. 2.5 500 m grid for spatial resampling.

This shows that many blocks basically have no training points, and some blocks are densely sampled with e.g. 15–20 points. Next, we can compare models fitted using spatial blocking vs no special settings. First, we fit ensemble ML model using no blocking:

```
parallelMap::parallelStartSocket(parallel::detectCores())
#> Starting parallelization in mode=socket with cpus=32.
```

```
library(mlr)
library(glmnet)
library(Cubist)
lrns <- list(mlr::makeLearner("regr.ranger",
                num.threads = parallel::detectCores(), num.trees=150, importance="impurity"),
             mlr::makeLearner("regr.glm"), mlr::makeLearner("regr.cubist"),
             mlr::makeLearner("regr.cvglmnet"))
tsk0 <- mlr::makeRegrTask(data = rm.cly[,all.vars(cly.fm)], target = "CLYMHT_A")
init0.m <- mlr::makeStackedLearner(lrns, method = "stack.cv",
                                   super.learner = "regr.lm",
```

```
                                resampling=mlr::makeResampleDesc(method = "CV"))
eml0 = train(init0.m, tsk0)
#> Exporting objects to slaves for mode socket: .mlr.slave.options
#> Mapping in parallel: mode = socket; level = mlr.resample; cpus = 32; elements = 10.
#> Exporting objects to slaves for mode socket: .mlr.slave.options
#> Mapping in parallel: mode = socket; level = mlr.resample; cpus = 32; elements = 10.
#> Exporting objects to slaves for mode socket: .mlr.slave.options
#> Mapping in parallel: mode = socket; level = mlr.resample; cpus = 32; elements = 10.
#> Exporting objects to slaves for mode socket: .mlr.slave.options
#> Mapping in parallel: mode = socket; level = mlr.resample; cpus = 32; elements = 10.
summary(eml0$learner.model$super.model$learner.model)
#>
#> Call:
#> stats::lm(formula = f, data = d)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -33.228  -4.005   0.298   3.054  37.818
#>
#> Coefficients:
#>               Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   -0.55143    0.47112  -1.170 0.241906
#> regr.ranger    0.80513    0.05689  14.152  < 2e-16 ***
#> regr.glm       0.24905    0.11067   2.250 0.024502 *
#> regr.cubist    0.13973    0.04208   3.320 0.000911 ***
#> regr.cvglmnet -0.17004    0.11544  -1.473 0.140868
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 7.339 on 2771 degrees of freedom
#> Multiple R-squared:  0.6041, Adjusted R-squared:  0.6036
#> F-statistic:  1057 on 4 and 2771 DF,  p-value: < 2.2e-16
```

This shows that `ranger` i.e. random forest and `cubist` are the most important learners and the overall model performance matches the previously fitted model using ranger. Next, we fit an ensemble ML model with the spatial blocking (500-m):

```
tsk1 <- mlr::makeRegrTask(data = rm.cly[,all.vars(cly.fm)], target = "CLYMHT_A",
                          blocking = as.factor(id[sel.cly]))
init1.m <- mlr::makeStackedLearner(lrns, method = "stack.cv",
                  super.learner = "regr.lm",
                  resampling=mlr::makeResampleDesc(method = "CV", blocking.cv=TRUE))
eml1 = train(init1.m, tsk1)
#> Exporting objects to slaves for mode socket: .mlr.slave.options
#> Mapping in parallel: mode = socket; level = mlr.resample; cpus = 32; elements = 10.
#> Exporting objects to slaves for mode socket: .mlr.slave.options
```

```
#> Mapping in parallel: mode = socket; level = mlr.resample; cpus = 32; elements = 10.
#> Exporting objects to slaves for mode socket: .mlr.slave.options
#> Mapping in parallel: mode = socket; level = mlr.resample; cpus = 32; elements = 10.
#> Exporting objects to slaves for mode socket: .mlr.slave.options
#> Mapping in parallel: mode = socket; level = mlr.resample; cpus = 32; elements = 10.
summary(eml1$learner.model$super.model$learner.model)
#>
#> Call:
#> stats::lm(formula = f, data = d)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -31.676  -4.201   0.443   3.109  40.386
#>
#> Coefficients:
#>               Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   -0.36065    0.48143  -0.749   0.4539
#> regr.ranger    0.63875    0.05688  11.229  < 2e-16 ***
#> regr.glm       0.21547    0.11185   1.926   0.0542 .
#> regr.cubist    0.23915    0.04124   5.800  7.4e-09 ***
#> regr.cvglmnet -0.08255    0.11581  -0.713   0.4760
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 7.514 on 2771 degrees of freedom
#> Multiple R-squared:  0.5851, Adjusted R-squared:  0.5845
#> F-statistic: 976.9 on 4 and 2771 DF,  p-value: < 2.2e-16
```

which shows a difference: the RMSE drops for 10–20% and the `regr.glm` learner is now also significant. In the previous example, it is possible that the `regr.glm` model was possibly *shadowed* by the fitting power of ranger and Cubist, while after more strict CV, also more simple models seem to perform with a comparable accuracy.

We can produce predictions using the Ensemble ML model developed with blocking by running:

```
pred.cly.eml = predict(eml1, newdata=eberg_spc@predicted@data[,eml1$features])
eberg_grid25$pred.cly.eml = pred.cly.eml$data$response
```

```
spplot(eberg_grid25[c("pred.cly.eml", "pred.cly.erf")], col.regions=SAGA_pal[[1]])
```

Visual comparison with the predictions produced in previous section, show that the Ensemble method `pred.cly.eml` predicts somewhat higher clay content in the extrapolation area, but also smooths out some higher values in the plains. Again, if we have to choose we would suggest users to use the map on the left for two main reasons:

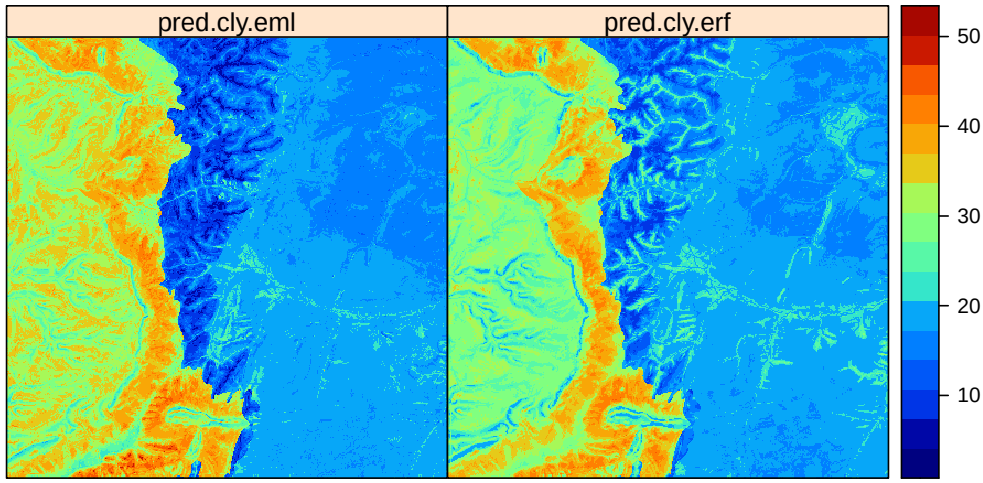


Fig. 2.6 Predictions of clay content: (left) Ensemble ML with spatial blocking, (right) after de-clustring of points.

1. It is based on multiple base-learners, not only on Random Forest and results show that both Cubist and glmnet package produce comparable results to RF.

2. It is probably more sensible to use the predictions produced by the meta-learner, especially in the extrapolation space.

2.5 Estimating the Area of Applicability

? have developed a method to estimate so-called "Area of Applicability"[12] using a fitted model and feature space analysis. This method can be used for post-modeling analysis and helps users realize what are the true extrapolation areas and where the predictions are critically poor. The users can then choose to e.g. limit predictions only to combinations of pixels that are NOT too risky (extrapolation).

For the RF model fitted above we can derive AoA using:

```
library(CAST)
train.df = rm.cly[,all.vars(cly.fm)[-1]]
```

[12] https://cran.r-project.org/web/packages/CAST/vignettes/AOA-tutorial.html

```
weight = as.data.frame(mlr::getFeatureImportance(eml1$learner.model$base.models[[1]])$res)
AOA <- CAST::aoa(train=train.df, predictors=eberg_spc@predicted@data[,eml1$features], weight=weight)
```

This method can be computational so it is probably not recommended for larger datasets. Some examples of the Area of Applicability can be found in the CAST package tutorial[13].

2.6 Estimating per-pixel mapping accuracy

Using the Ensemble Model we can also estimate the mapping accuracy per pixel i.e. by deriving the **Mean Square Prediction Error** (MSPE). The **forestError** package currently provides a *"Unified Framework for Random Forest Prediction Error Estimation"* (**?**) and is probably the most worked-out procedure for deriving prediction errors and estimating potential bias. This requires two steps, (1) first, we need to fit an additional quantile Regression RF model (using the four base learners from the previous section), (2) second, we can then estimate complete error statistics per pixel using the quantForestError[14] function.

Because derivation of prediction errors per pixel can often be computational (even at the order of magnitude more computational than predictions), it is important to use a method that is computationally efficient and precise enough. In the landmap package the uncertainty is derived using base learners instead of using ALL raster layers which could be hundreds. This approach of using (few) base learners instead of (many) original covariates helps compress the complexity of model and significantly speed-up computing. The base learners can be accessed from the mlr object:

```
eml.t = eml1$learner.model$super.model$learner.model$terms
paste(eml.t)
#> [1] "~"
#> [2] "CLYMHT_A"
#> [3] "regr.ranger + regr.glm + regr.cubist + regr.cvglmnet"
eml.m = eml1$learner.model$super.model$learner.model$model
```

We use the spatially resampled base-learners to fit (an independent) quantile RF:

```
eml.qr <- ranger::ranger(eml.t, eml.m, num.trees=85, importance="impurity",
                         quantreg=TRUE, keep.inbag = TRUE)
#eml.qr
```

Next, we can use the `forestError` package to derive prediction errors by:

[13] https://cran.r-project.org/web/packages/CAST/vignettes/AOA-tutorial.html
[14] https://rdrr.io/cran/forestError/man/quantForestError.html

```
library(forestError)
quantiles = c((1-.682)/2, 1-(1-.682)/2)
n.cores = parallel::detectCores()
out.c <- as.data.frame(mlr::getStackedBaseLearnerPredictions(eml1,
          newdata=eberg_spc@predicted@data[,eml1$features]))
pred.q = forestError::quantForestError(eml.qr,
                    X.train = eml.m[,all.vars(eml.t)[-1]],
                    X.test = out.c,
                    Y.train = eml.m[,all.vars(eml.t)[1]],
                    alpha = (1-(quantiles[2]-quantiles[1])), n.cores=n.cores)
```

We could have also subset e.g. 10% of the input points and keep them ONLY for estimating the prediction errors using the `quantForestError`, which is also recommended by the authors of the forestError package. In practice, because base learners have been fitted using 5-fold Cross-Validation with blocking, they are already out-of-bag samples hence taking out extra OOB samples is probably not required, but you can also test this with your own data.

The `quantForestError` function runs a complete uncertainty assessment and includes both MSPE, bias and upper and lower confidence intervals:

```
str(pred.q)
#> List of 3
#>  $ estimates:'data.frame':   160000 obs. of  5 variables:
#>   ..$ pred       : num [1:160000] 35.2 37.5 31.7 36 35.9 ...
#>   ..$ mspe       : num [1:160000] 138.1 104.1 82.7 90.9 76.2 ...
#>   ..$ bias       : num [1:160000] -1.77 -1.22 -1.13 -1.18 -1.45 ...
#>   ..$ lower_0.318: num [1:160000] 23.5 27.8 21.5 29.6 29.5 ...
#>   ..$ upper_0.318: num [1:160000] 52.7 47.7 40.2 47.7 42.3 ...
#>  $ perror   :function (q, xs = 1:n.test)
#>  $ qerror   :function (p, xs = 1:n.test)
```

In this case, for the lower and upper confidence intervals we use 1-standard deviation probability which is about 2/3 probability and hence the lower value is 0.159 and upper 0.841. The RMSPE should match the half of the difference between the lower and upper interval in this case, although there will be difference in exact numbers.

The mean RMSPE for the whole study area (mean of all pixels) should be as expected somewhat higher than the RMSE we get from model fitting:

```
mean(sqrt(pred.q$estimates$mspe), na.rm=TRUE)
#> [1] 7.780946
```

This is because we are also extrapolating in the large part of the area. The map in Fig. **??** correctly depicts the extrapolation areas as having much higher RMSPE (compare with Fig. **??**):

```
eberg_grid25$rmspe.cly.eml = sqrt(pred.q$estimates$mspe)
plot(raster(eberg_grid25[c("rmspe.cly.eml")]), col=SAGA_pal[[16]])
points(eberg.sp, pch="+", cex=.6)
```

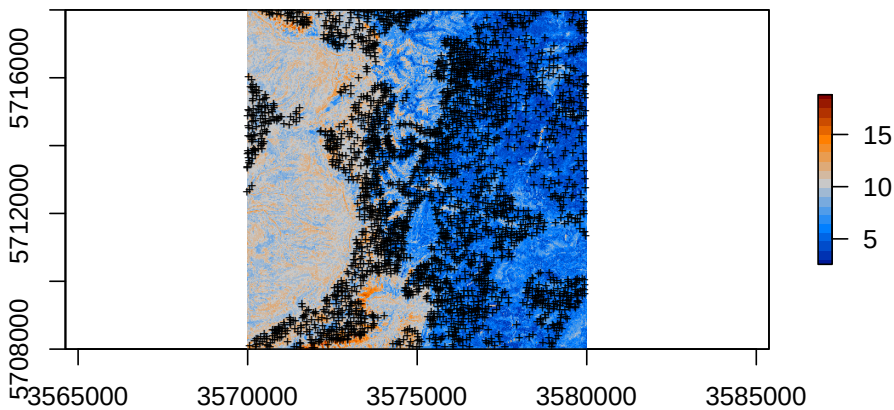


Fig. 2.7 Prediction errors for the clay content based on the forestError package and Ensemble ML.

In the previous examples we have shown that the actual samples from the Ebergotzen dataset are actually clustered and cover only agricultural land. Can we still use these samples to estimate the mapping accuracy for the whole area? The answer is yes, but we need to be aware that our estimate might be biased (usually over-optimistic) and we need to do our best to reduce the over-fitting effects by implementing some of the strategies above e.g.: assign different weights to training points, and/or implement blocking settings.

Assuming that forest soils are possibly very different from agricultural soils, once we collect new sampling in the forest part of the study area we might discovering that the actual mapping accuracy we estimated for the whole study area using only agricultural soil samples is significantly lower than what we have estimated in Fig. **??**.

Fig. **??** shows the usual effect of spatial blocking (for varying buffer size) on the Cross-Validation accuracy (**?**). Note that the accuracy tends to stabilize at some distance, although too strict blocking can also lead to over-pessimistic estimates of accuracy hence bias in predictions (@ **?**). In the Ebergotzen case, prediction error could be over-pessimistic although the block size is relatively small considering the size of the study area. On the other hand, if the training points are clustered, blocking becomes important because otherwise the estimate of error and choice of model parameters could get over-optimistic (**??**). If in doubt of whether to produce over-optimistic or over-pessimistic estimates of uncertainty, it is of course ideal to avoid both, but if necessary

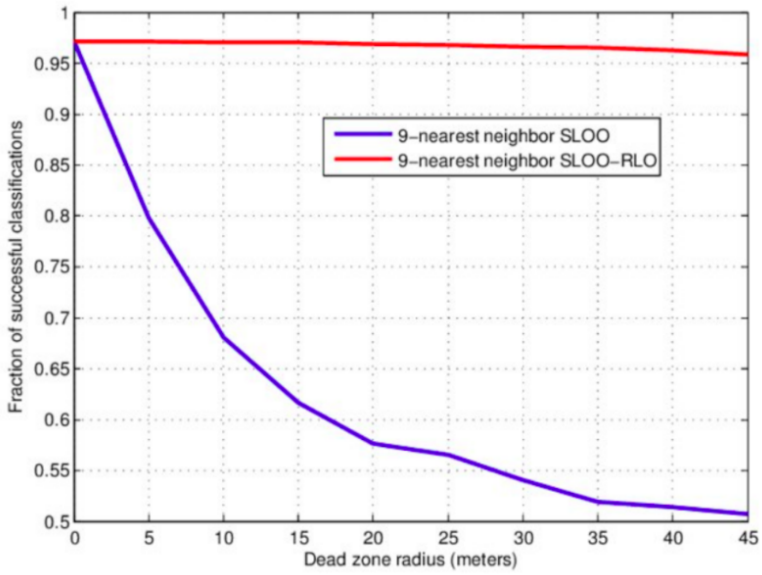


Fig. 2.8 Example of effects of the size of the spatial blocking (buffer) on mapping accuracy.

consider that somewhat over-pessimistic estimate of accuracy could be slightly more *on a safe side* (**?**).

2.7 Testing mapping accuracy using resampling and blocking

We can switch to the Edgeroi dataset (**?**) that is originally based on *designed* sampling and as such is more interesting for assessing effects of various blocking strategies on overall mapping accuracy. We can load the dataset and prepare a regression matrix by using (**?**):

```
data(edgeroi)
edgeroi.sp <- edgeroi$sites
coordinates(edgeroi.sp) <- ~ LONGDA94 + LATGDA94
proj4string(edgeroi.sp) <- CRS("+proj=longlat +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +no_defs")
edgeroi.sp <- spTransform(edgeroi.sp, CRS("+init=epsg:28355"))
length(edgeroi.sp)
#> [1] 359
h2 <- hor2xyd(edgeroi$horizons)
edgeroi.grids.100m = readRDS("./extdata/edgeroi.grids.100m.rds")
edgeroi.grids = landmap::spc(edgeroi.grids.100m[-1])
#> Converting covariates to principal components...
```

```
plot(raster(edgeroi.grids@predicted[1]))
points(edgeroi.sp, pch="+")
```

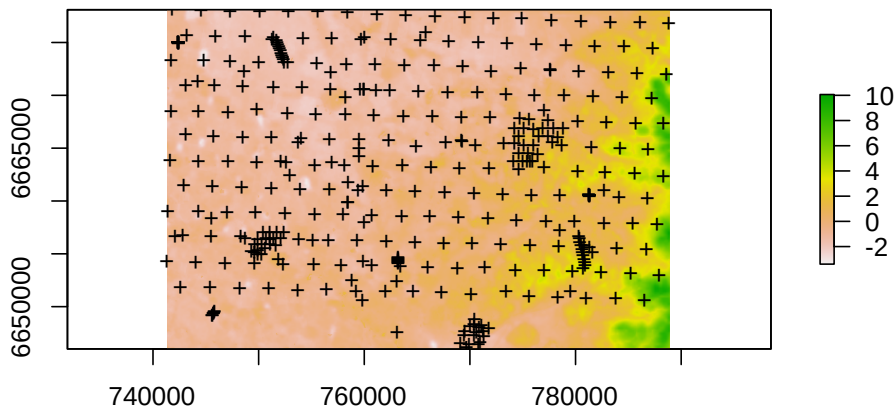


Fig. 2.9 The Edgeroi dataset consisting of 359 soil profiles.

The dataset documentation indicates that from a total of 359 profiles, 210 soil profiles were sampled on a systematic, equilateral triangular grid with a spacing of 2.8 km between sites; the further 131 soil profiles are distributed more irregularly or on transects (**?**). This is hence a **hybrid sampling design** but in general satisfying IID, and hence any subsample of these points should give an unbiased estimate of the mapping accuracy. We first prepare a regression matrix that includes all covariates, location IDs and we also add a spatial grid of 500-m size:

```
grdE <- sp::GridTopology(cellcentre.offset=edgeroi.grids.100m@bbox[,1], cellsize=rep(500,2),
                         cells.dim=c(ceiling(abs(diff(edgeroi.grids.100m@bbox[1,])/500))+1,
                         ceiling(abs(diff(edgeroi.grids.100m@bbox[2,])/500))+1))
rE.sp <- sp::SpatialGridDataFrame(grdE, proj4string = edgeroi.grids.100m@proj4string,
                    data=data.frame(gid=1:(grdE@cells.dim[1] * grdE@cells.dim[2])))
ovF <- sp::over(edgeroi.sp, edgeroi.grids@predicted)
ovF$SOURCEID <- edgeroi.sp$SOURCEID
ovF$gid <- sp::over(edgeroi.sp, rE.sp)$gid
ovF$x = edgeroi.sp@coords[,1]
ovF$y = edgeroi.sp@coords[,2]
rmF <- plyr::join_all(dfs = list(edgeroi$sites, h2, ovF))
```

```
#> Joining by: SOURCEID
#> Joining by: SOURCEID
```

This produces a regression matrix with unique IDs of profiles `SOURCEID`, spatial block IDs (`gid`) and all target and covariate layers.

We can now fit a model to predict soil organic carbon content (in g/kg) in 3D:

```
rmF$log.ORCDRC = log1p(rmF$ORCDRC)
formulaStringPF <- as.formula(paste0("log.ORCDRC ~ DEPTH + ", paste0("PC", 1:10, collapse = "+")))
rmPF <- rmF[complete.cases(rmF[,all.vars(formulaStringPF)]),]
#str(rmPF[,all.vars(formulaStringPF)])
```

We first fit a model distribution of soil organic carbon ignoring any spatial clustering, overlap in 3rd dimension (soil depth) or similar:

```
soc.rf = ranger(formulaStringPF, rmPF)
soc.rf
#> Ranger result
#>
#> Call:
#>  ranger(formulaStringPF, rmPF)
#>
#> Type:                             Regression
#> Number of trees:                  500
#> Sample size:                      5001
#> Number of independent variables:  11
#> Mtry:                             3
#> Target node size:                 5
#> Variable importance mode:         none
#> Splitrule:                        variance
#> OOB prediction error (MSE):       0.1116369
#> R squared (OOB):                  0.8166682
```

If we compare this model with an Ensemble ML where whole blocks i.e. including also whole soil profiles are taken out from modeling:

```
SL.library2 = c("regr.ranger", "regr.glm", "regr.cvglmnet", "regr.xgboost", "regr.ksvm")
lrnsE <- lapply(SL.library2, mlr::makeLearner)
tskE <- mlr::makeRegrTask(data = rmPF[,all.vars(formulaStringPF)], target="log.ORCDRC",
                          blocking = as.factor(rmPF$gid))
initE.m <- mlr::makeStackedLearner(lrnsE, method = "stack.cv",
                  super.learner = "regr.lm",
```

```
                  resampling=mlr::makeResampleDesc(method = "CV", blocking.cv=TRUE))
emlE = train(initE.m, tskE)
#> Exporting objects to slaves for mode socket: .mlr.slave.options
#> Mapping in parallel: mode = socket; level = mlr.resample; cpus = 32; elements = 10.
#> Exporting objects to slaves for mode socket: .mlr.slave.options
#> Mapping in parallel: mode = socket; level = mlr.resample; cpus = 32; elements = 10.
#> Exporting objects to slaves for mode socket: .mlr.slave.options
#> Mapping in parallel: mode = socket; level = mlr.resample; cpus = 32; elements = 10.
#> Exporting objects to slaves for mode socket: .mlr.slave.options
#> Mapping in parallel: mode = socket; level = mlr.resample; cpus = 32; elements = 10.
#> [12:18:24] WARNING: amalgamation/../src/objective/regression_obj.cu:170: reg:linear is now deprecated in favor o
#> Exporting objects to slaves for mode socket: .mlr.slave.options
#> Mapping in parallel: mode = socket; level = mlr.resample; cpus = 32; elements = 10.
summary(emlE$learner.model$super.model$learner.model)
#>
#> Call:
#> stats::lm(formula = f, data = d)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -2.1642 -0.2550 -0.0232  0.2376  3.1830
#>
#> Coefficients:
#>               Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   -0.17386    0.03557  -4.887 1.05e-06 ***
#> regr.ranger    0.83042    0.04432  18.736  < 2e-16 ***
#> regr.glm       0.42906    0.07066   6.072 1.36e-09 ***
#> regr.cvglmnet -0.37021    0.07626  -4.855 1.24e-06 ***
#> regr.xgboost   0.20284    0.09370   2.165   0.0305 *
#> regr.ksvm      0.12077    0.02840   4.253 2.15e-05 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.449 on 4995 degrees of freedom
#> Multiple R-squared:  0.6693, Adjusted R-squared:  0.6689
#> F-statistic:  2022 on 5 and 4995 DF,  p-value: < 2.2e-16
```

Notice a large difference in the model accuracy with R-square dropping from about 0.82 to 0.65. How can we check which of the two models is more accurate / which shows a more realistic mapping accuracy? We can again generate pseudo-grid samples e.g. 10 subsets where in each subset we make sure that the points are at least 3.5-km apart and are taken selected randomly:

```
subE.lst = lapply(1:10, function(i){landmap::sample.grid(edgeroi.sp, c(3.5e3, 3.5e3), n=1)})
```

```
lE1 <- list("sp.points", subE.lst[[1]]$subset, pch="+", col="black")
spplot(subE.lst[[1]]$grid, scales=list(draw=TRUE),
   col.regions="grey", sp.layout=list(lE1), colorkey=FALSE)
```

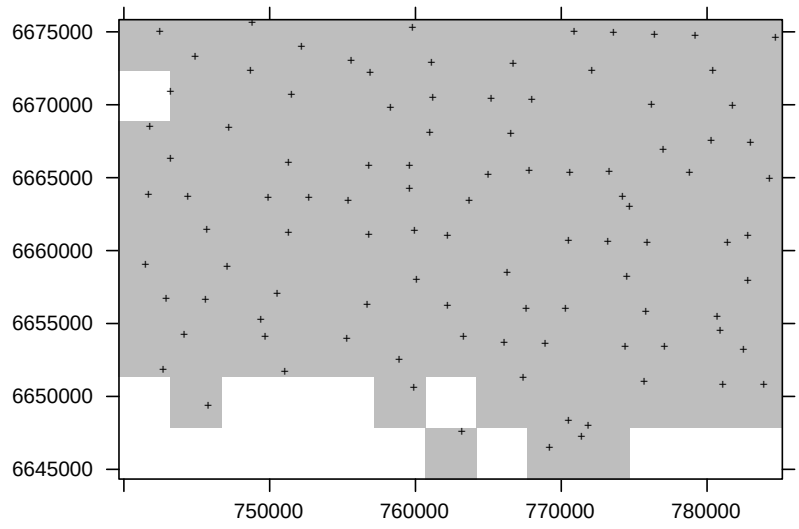


Fig. 2.10 Resampling original points using 'sample.grid' function for the Edgeroi dataset.

So in any random pseudo-grid subset we take out about 100 profiles from 359 and keep for validation only. We can next repeatedly fit models using the two approaches and derive prediction errors. First, for simple model ignoring any spatial clustering / soil profile locations:

```
rf.soc.lst = lapply(1:length(subE.lst), function(i){
        sel <- !rmPF$SOURCEID %in% subE.lst[[i]]$subset$SOURCEID;
        y <- ranger::ranger(formulaStringPF, rmPF[sel,]);
        out <- data.frame(meas=rmPF[!sel,"log.ORCDRC"], pred=predict(y, rmPF[!sel,])$predictions);
        return(out)
      }
)
rf.cv = do.call(rbind, rf.soc.lst)
Metrics::rmse(rf.cv$meas, rf.cv$pred)
#> [1] 0.4343792
```

This gives an RMSPE of 0.44, which is higher than what is reported by the OOB for RF without blocking. The accuracy plot shows that the Concordance Correlation Coefficient (CCC)[15] is about

[15] https://rdrr.io/cran/yardstick/man/ccc.html

0.79 (corresponding to a R-square of about 0.62 and thus significantly less than what is reported by ranger):

```
t.b = quantile(log1p(rmPF$ORCDRC), c(0.001, 0.01, 0.999), na.rm=TRUE)
plot_hexbin(varn="SOC_RF", breaks=c(t.b[1], seq(t.b[2], t.b[3], length=25)),
      meas=rf.cv$meas, pred=rf.cv$pred, main="SOC [RF]")
```

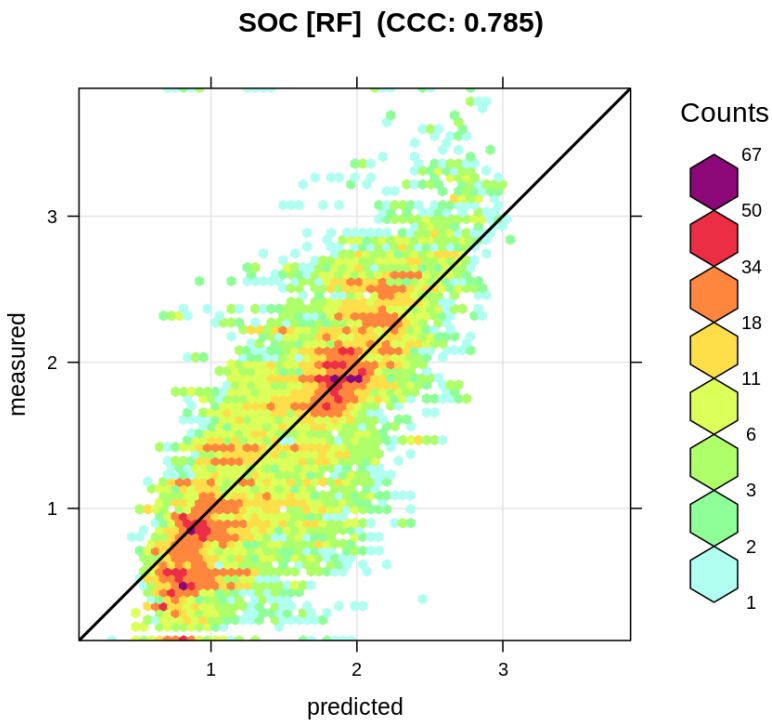


Fig. 2.11 Accuracy plot for soil organic carbon fitted using RF.

We repeat the same process of re-fitting the model using Ensemble ML with spatial blocking:

```
eml.soc.lst = lapply(1:length(subE.lst), function(i){
        sel <- !rmPF$SOURCEID %in% subE.lst[[i]]$subset$SOURCEID;
        x <- mlr::makeRegrTask(data = rmPF[sel,all.vars(formulaStringPF)],
                  target="log.ORCDRC", blocking = as.factor(rmPF$gid[sel]));
        y <- train(initE.m, x)
        out <- data.frame(meas=rmPF[!sel,"log.ORCDRC"], pred=predict(y, newdata=rmPF[!sel, y$features])$data$respor
        return(out)
      }
```

```
)
eml.cv = do.call(rbind, eml.soc.lst)
Metrics::rmse(eml.cv$meas, eml.cv$pred)
```

```
plot_hexbin(varn="SOC_EML", breaks=c(t.b[1], seq(t.b[2], t.b[3], length=25)),
      meas=eml.cv$meas, pred=eml.cv$pred, main="SOC [EML]")
```

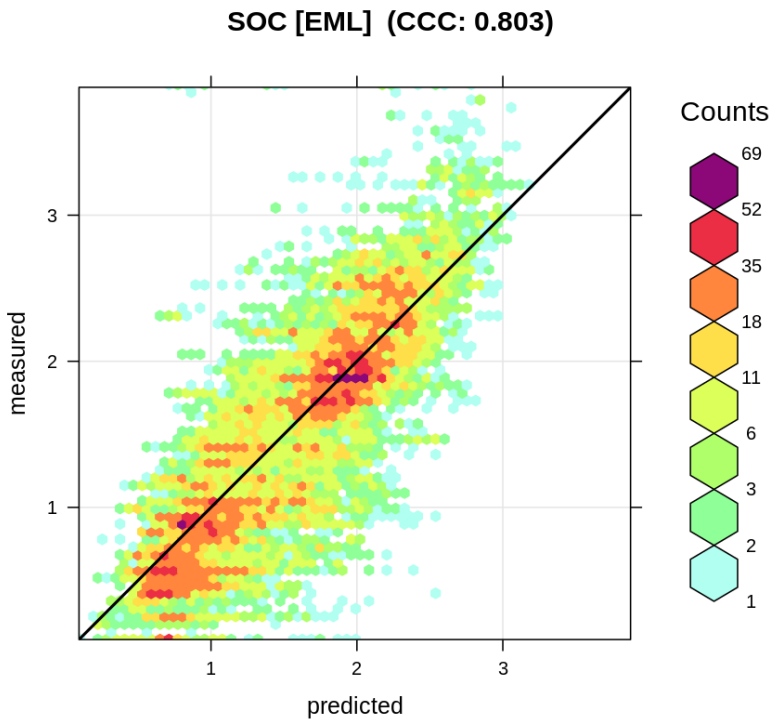


Fig. 2.12 Accuracy plot for soil organic carbon fitted using Ensemble Machine Learning with spatial blocking.

So in summary, independent validation using pseudo-probability samples indicates that the Ensemble ML produces more accurate predictions (in this case only slightly better) and the RMSE estimated by the meta-learner the Ensemble ML approach is more realistic (`Residual standard error: 0.45`). This clearly demonstrates that spatial blocking is important to (a) prevent from over-fitting, (b) produce a more realistic estimate of the uncertainty / mapping accuracy. Ensemble ML comes at costs of at the order of magnitude higher computing costs however.

We can again plot the predictions produced by two methods next to each other:

```
newdata = edgeroi.grids@predicted@data[,paste0("PC", 1:10)]
newdata$DEPTH = 5
```

```
edgeroi.grids.100m$rf_soc_5cm = predict(soc.rf, newdata)$predictions
edgeroi.grids.100m$eml_soc_5cm = predict(emlE, newdata=newdata[,emlE$features])$data$response
l.pnts <- list("sp.points", edgeroi.sp, pch="+", col="black")
spplot(edgeroi.grids.100m[c("rf_soc_5cm", "eml_soc_5cm")],
       sp.layout = list(l.pnts), col.regions=SAGA_pal[[1]])
```

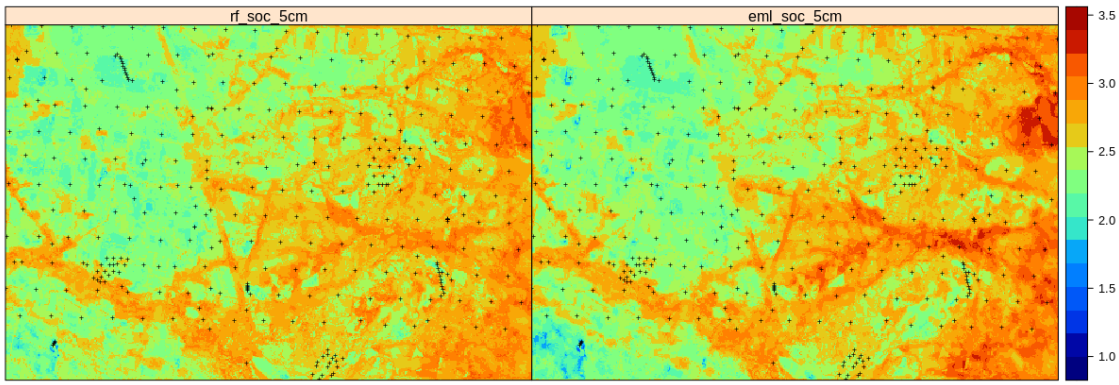


Fig. 2.13 Predictions of soil organic carbon (in log-scale) based on Random Forest (RF) and Ensemble ML (EML).

Which shows that the Ensemble ML seems to predict significantly higher SOC in the hillands (right part of the study area), so again significant difference in predictions between the two models. Even though the Ensemble ML with spatial blocking is only slightly better in accuracy (RMSE based on 10-times out-of-bag declustered validation points), these results confirm that it helps produce a more realistic map of RMSPE. This matches the result of **?** and **?** who suggest that block cross-validation is nearly universally more appropriate than random cross-validation if the goal is predicting to new data or predictor space, or for selecting causal predictors.

We can also map the prediction errors:

```
emlE.t = emlE$learner.model$super.model$learner.model$terms
paste(emlE.t)
#> [1] "~"
#> [2] "log.ORCDRC"
#> [3] "regr.ranger + regr.glm + regr.cvglmnet + regr.xgboost + regr.ksvm"
emlE.m = emlE$learner.model$super.model$learner.model$model
emlE.qr <- ranger::ranger(emlE.t, emlE.m, num.trees=85, importance="impurity",
                          quantreg=TRUE, keep.inbag = TRUE)
```

```
outE.c <- as.data.frame(mlr::getStackedBaseLearnerPredictions(emlE,
                                    newdata=newdata[,emlE$features]))
predE.q = forestError::quantForestError(emlE.qr,
                    X.train = emlE.m[,all.vars(emlE.t)[-1]],
                    X.test = outE.c,
                    Y.train = emlE.m[,all.vars(emlE.t)[1]],
                    alpha = (1-(quantiles[2]-quantiles[1])), n.cores=n.cores)
```

Which again shows where could be the main extrapolation problems i.e. where multiple base learners perform poorly:

```
edgeroi.grids.100m$rmspe.soc.eml = sqrt(predE.q$estimates$mspe)
plot(raster(edgeroi.grids.100m[c("rmspe.soc.eml")]), col=SAGA_pal[[16]])
points(edgeroi.sp, pch="+", cex=.8)
```

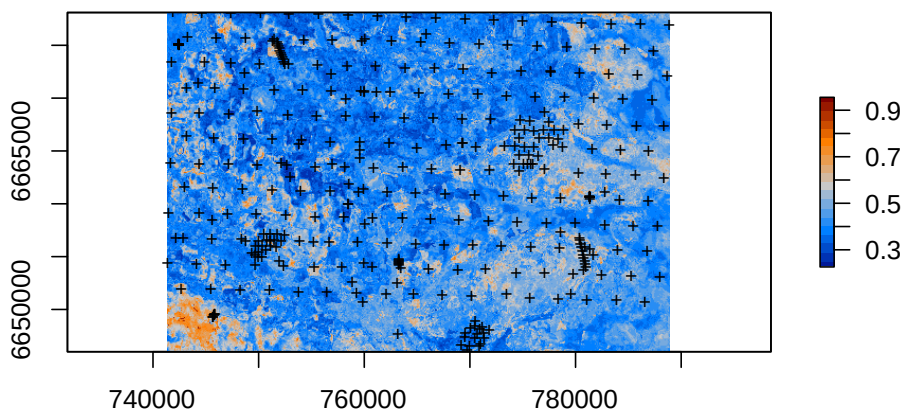


Fig. 2.14 Prediction errors for the soil organic carbon content based on the forestError package and Ensemble ML.

```
rgdal::writeGDAL(edgeroi.grids.100m[c("rmspe.soc.eml")],
                 "./output/edgeroi_soc_rmspe.tif",
                 options=c("COMPRESS=DEFLATE"))
```

The overall mapping accuracy for Edgeroi based on the mean prediction error is thus:

```
mean(sqrt(predE.q$estimates$mspe), na.rm=TRUE)
#> [1] 0.4352432
```

which in general matches what we get through repeated validation using pseudo-SRS subsampling.

From this experiment we can conclude that the mapping accuracy estimated using ranger and out-of-bag samples and ignoring locations of profiles was probably over-optimistic and hence ranger has possibly over-fitted the target variable. This is in fact common problem observed with many 3D predictive soil mapping models where soil profiles basically have ALL the same values of covariates and Random Forest thus easier predicts values due to overlap in covariate data. For a discussion on why is important to run internal training and Cross-Validation using spatial blocking refer also to **?** and **?**.

```
parallelMap::parallelStop()
#> Stopped parallelization. All cleaned up.
```

Chapter 3
Resampling for spatiotemporal Machine Learning

You are reading the work-in-progress Spatial Sampling and Resampling for Machine Learning. This chapter is currently draft version, a peer-review publication is pending. You can find the polished first edition at https://opengeohub.github.io/spatial-sampling-ml/.

3.1 Case study: Cookfarm dataset

We next look at the Cookfarm dataset[1], which is available via the landmap package and described in detail in **?**:

```
library(landmap)
#?landmap::cookfarm
data("cookfarm")
```

This dataset contains spatio-temporal (3D+T) measurements of three soil properties and a number of spatial and temporal regression covariates. In this example multiple covariates are used to fit a spatiotemporal model to predict soil moisture, soil temperature and electrical conductivity in 3D+T (hence 2 extra dimension beyond spatial dimensions i.e. a 2D model).

We can load the prediction locations and regression-matrix from:

```
library(rgdal)
library(ranger)
cookfarm.rm = readRDS('extdata/cookfarm_st.rds')
cookfarm.grid = readRDS('extdata/cookfarm_grid10m.rds')
```

We are specifically interested in modeling soil moisture (`VW`) as a function of soil depth (`altitude`), elevation (`DEM`), Topographic Wetness Index (`TWI`), Normalized Difference Red Edge Index (`NDRE.M`),

[1] https://rdrr.io/cran/landmap/man/cookfarm.html

Normalized Difference Red Edge Index (NDRE.sd), Cumulative precipitation in mm (Precip_cum), Maximum measured temperature (MaxT_wrcc), Minimum measured temperature (MinT_wrcc) and the transformed cumulative day (cdayt):

```
fm <- VW ~ altitude+DEM+TWI+NDRE.M+NDRE.Sd+Precip_cum+MaxT_wrcc+MinT_wrcc+cdayt
```

We can use the ranger package to fit a random forest model:

```
m.vw = ranger(fm, cookfarm.rm, num.trees = 100)
m.vw
#> Ranger result
#>
#> Call:
#>  ranger(fm, cookfarm.rm, num.trees = 100)
#>
#> Type:                             Regression
#> Number of trees:                  100
#> Sample size:                      107851
#> Number of independent variables:  9
#> Mtry:                             3
#> Target node size:                 5
#> Variable importance mode:         none
#> Splitrule:                        variance
#> OOB prediction error (MSE):       0.0009826038
#> R squared (OOB):                  0.8479968
```

which shows that a significant model can be fitting using this data with R-square about 0.85. The accuracy plot shows that the Concordance Correlation Coefficient (CCC)[2] is high:

```
vw.b = quantile(cookfarm.rm$VW, c(0.001, 0.01, 0.999), na.rm=TRUE)
plot_hexbin(varn="VW_RF", breaks=c(vw.b[1], seq(vw.b[2], vw.b[3], length=25)),
      meas=cookfarm.rm$VW, pred=m.vw$predictions, main="VW [RF]")
```

This model, however, as shown in **?**, ignores the fact that many VW measurements have exactly the same location (monitoring station with four depths), hence ranger over-fits data and gives unrealistic R-square. We can instead fit an Ensemble ML model, but we will also use a **blocking parameter** that should protect from over-fitting: the unique code of the station (SOURCEID). This means that **complete stations** will be either used for training or for validation. This satisfies the requirement of **?** for predicting to new data or predictor space by blocking clustered or overlapping measurements.

We use the same procedure in mlr as in the previous example:

[2] https://rdrr.io/cran/yardstick/man/ccc.html

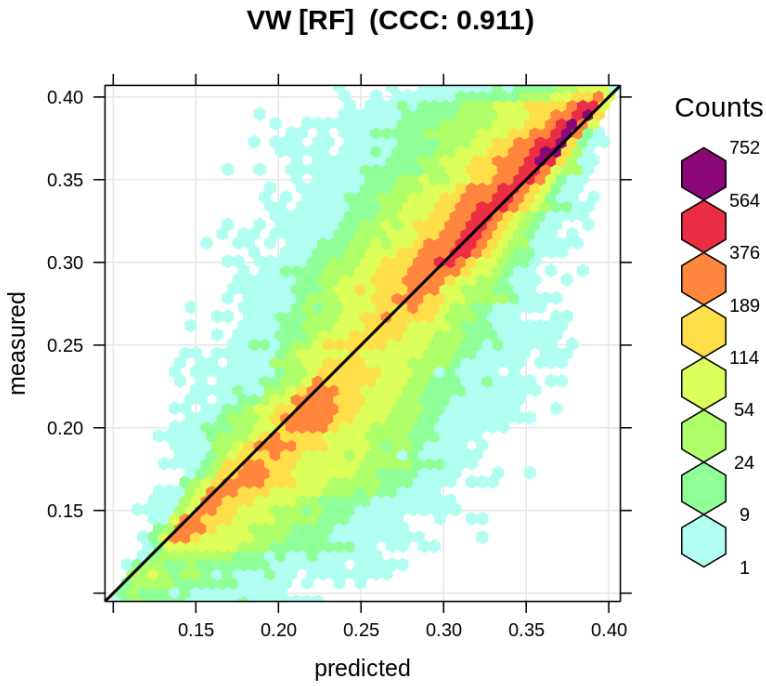


Fig. 3.1 Accuracy plot for soil moisture content fitted using RF.

```
library(mlr)
SL.lst = c("regr.ranger", "regr.gamboost", "regr.cvglmnet")
lrns.st <- lapply(SL.lst, mlr::makeLearner)
## subset to 5% to speed up computing
subs <- runif(nrow(cookfarm.rm))<.05
tsk.st <- mlr::makeRegrTask(data = cookfarm.rm[subs, all.vars(fm)], target = "VW",
                            blocking = as.factor(cookfarm.rm$SOURCEID)[subs])
tsk.st
#> Supervised task: cookfarm.rm[subs, all.vars(fm)]
#> Type: regr
#> Target: VW
#> Observations: 5321
#> Features:
#>    numerics     factors     ordered functionals
#>           9           0           0           0
#> Missings: FALSE
#> Has weights: FALSE
#> Has blocking: TRUE
#> Has coordinates: FALSE
```

The resulting model again used simple linear regression for stacking various learners:

```
#> Starting parallelization in mode=socket with cpus=32.
#> Exporting objects to slaves for mode socket: .mlr.slave.options
#> Mapping in parallel: mode = socket; level = mlr.resample; cpus = 32; elements = 10.
#> Exporting objects to slaves for mode socket: .mlr.slave.options
#> Mapping in parallel: mode = socket; level = mlr.resample; cpus = 32; elements = 10.
#> Loading required package: mboost
#> Loading required package: parallel
#> Loading required package: stabs
#>
#> Attaching package: 'stabs'
#> The following object is masked from 'package:mlr':
#>
#>     subsample
#> The following object is masked from 'package:spatstat.core':
#>
#>     parameters
#> This is mboost 2.9-2. See 'package?mboost' and 'news(package  = "mboost")'
#> for a complete list of changes.
#>
#> Attaching package: 'mboost'
#> The following object is masked from 'package:glmnet':
#>
#>     Cindex
#> The following object is masked from 'package:spatstat.core':
#>
#>     Poisson
#> The following objects are masked from 'package:raster':
#>
#>     cv, extract
#> Exporting objects to slaves for mode socket: .mlr.slave.options
#> Mapping in parallel: mode = socket; level = mlr.resample; cpus = 32; elements = 10.
#> Stopped parallelization. All cleaned up.
```

Note that here we can use full-parallelisation to speed up computing by using the `parallelMap` package. This resulting EML model now shows a more realistic R-square / RMSE:

```
summary(eml.VW$learner.model$super.model$learner.model)
#>
#> Call:
#> stats::lm(formula = f, data = d)
#>
#> Residuals:
#>       Min        1Q    Median        3Q       Max
#> -0.188277 -0.044502  0.003824  0.044429  0.177609
#>
```

```
#> Coefficients:
#>                Estimate Std. Error t value Pr(>|t|)
#> (Intercept)    0.044985   0.009034   4.979 6.58e-07 ***
#> regr.ranger    1.002421   0.029440  34.050  < 2e-16 ***
#> regr.gamboost -0.664149   0.071813  -9.248  < 2e-16 ***
#> regr.cvglmnet  0.510689   0.052930   9.648  < 2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.06262 on 5317 degrees of freedom
#> Multiple R-squared:  0.3914, Adjusted R-squared:  0.3911
#> F-statistic:  1140 on 3 and 5317 DF,  p-value: < 2.2e-16
```

The accuracy plot also shows the CCC to be almost 40% smaller than if no blocking is used:

```
plot_hexbin(varn="VW_EML", breaks=c(vw.b[1], seq(vw.b[2], vw.b[3], length=25)),
      meas=eml.VW$learner.model$super.model$learner.model$model$VW,
      pred=eml.VW$learner.model$super.model$learner.model$fitted.values,
      main="VW [EML]")
```

The Ensemble ML is now a 3D+T model of `VW`, which means that we can use it to predict values of `VW` at any new `x,y,d,t` location. To make prediction for a specific spacetime *slice* we use:

```
cookfarm$weather$Precip_cum <- ave(cookfarm$weather$Precip_wrcc,
                                   rev(cumsum(rev(cookfarm$weather$Precip_wrcc)==0)), FUN=cumsum)
date = as.Date("2012-07-30")
cday = floor(unclass(date)/86400-.5)
cdayt = cos((cday-min(cookfarm.rm$cday))*pi/180)
depth = -0.3
new.st <- data.frame(cookfarm.grid)
new.st$Date = date
new.st$cdayt = cdayt
new.st$altitude = depth
new.st = plyr::join(new.st, cookfarm$weather, type="left")
#> Joining by: Date
## predict:
pr.df = predict(eml.VW, newdata = new.st[,all.vars(fm)[-1]])
#> Warning in bsplines(mf[[i]], knots = args$knots[[i]]$knots, boundary.knots =
#> args$knots[[i]]$boundary.knots, : Some 'x' values are beyond 'boundary.knots';
#> Linear extrapolation used.

#> Warning in bsplines(mf[[i]], knots = args$knots[[i]]$knots, boundary.knots =
#> args$knots[[i]]$boundary.knots, : Some 'x' values are beyond 'boundary.knots';
#> Linear extrapolation used.
```

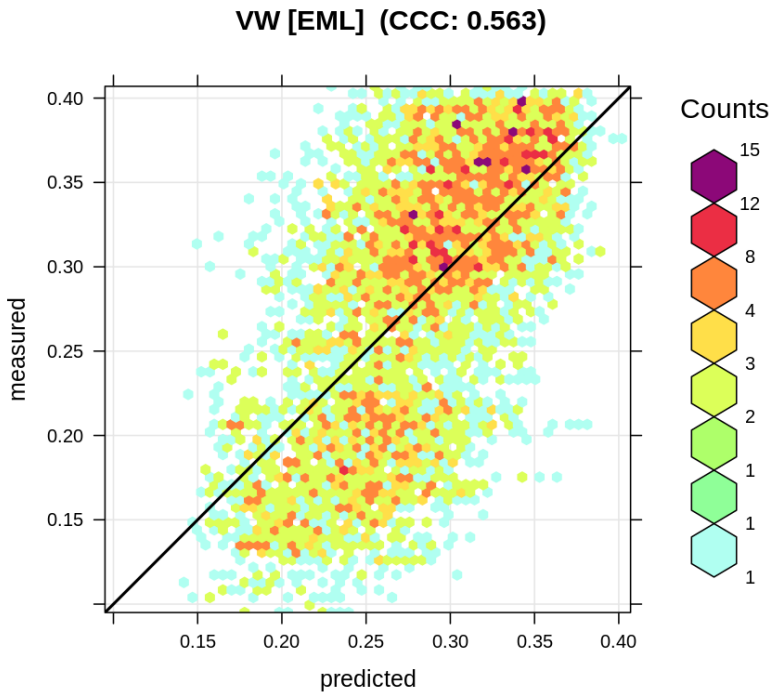


Fig. 3.2 Accuracy plot for soil moisture content fitted using Ensemble ML with blocking (taking complete stations out).

```
#> Warning in bsplines(mf[[i]], knots = args$knots[[i]]$knots, boundary.knots =
#> args$knots[[i]]$boundary.knots, : Some 'x' values are beyond 'boundary.knots';
#> Linear extrapolation used.

#> Warning in bsplines(mf[[i]], knots = args$knots[[i]]$knots, boundary.knots =
#> args$knots[[i]]$boundary.knots, : Some 'x' values are beyond 'boundary.knots';
#> Linear extrapolation used.
```

To plot prediction together with locations of training points we can use:

```
cookfarm.grid$pr.VW = pr.df$data$response
plot(raster::raster(cookfarm.grid["pr.VW"]), col=rev(bpy.colors()),
     main="Predicted VW for 2012-07-30 and depth -0.3 m", axes=FALSE, box=FALSE)
points(cookfarm$profiles[,c("Easting","Northing")], pch="+", cex=.8)
```

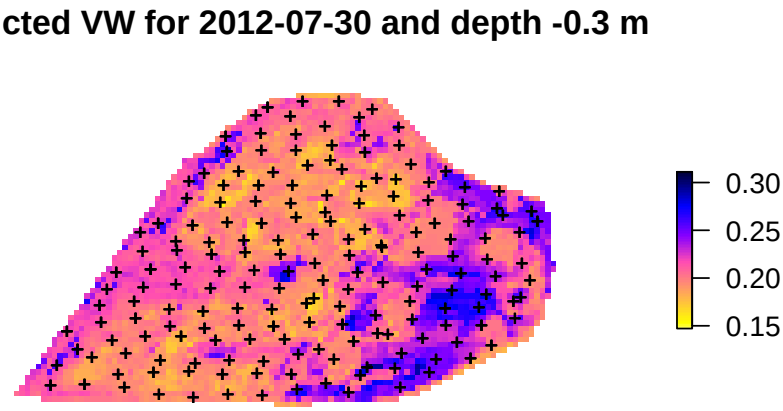


Fig. 3.3 Predicted soil water content based on spatiotemporal EML.

Because this is a spacetime dataset, we could also predict values of `VW` for longer periods (e.g. 100 days) then visualize changes using e.g. the `animation` package or similar.

In summary this study also demonstrate the importance of resampling point data using strict blocking of points that repeat in spacetime (measurement stations) is important to prevent from overfitting. The difference between models fitted using blocking per station and ignoring blocking can be drastic, hence how we define and use resampling is important (**?**).

Chapter 4
Generating 2nd, 3rd round sampling

You are reading the work-in-progress Spatial Sampling and Resampling for Machine Learning. This chapter is currently draft version, a peer-review publication is pending. You can find the polished first edition at https://opengeohub.github.io/spatial-sampling-ml/.

4.1 Uncertainty guided sampling

A sensible approach to improving quality of predictions is to: (a) estimate initial ML models, (b) produce a realistic estimate of prediction errors, and (c) revisit area and collect 2nd round samples that help at smallest possible costs significantly improve predictions. Logical focus of the 2nd round sampling could be thus on minimizing the overall prediction errors i.e. revising parts of the study area that shows the highest prediction errors / prediction problems (Fig. **??**). This is exactly procedure that **?** recommend in their paper and that they refer to as the *"Uncertainty guided sampling"*.

The 2nd round Uncertainty guided sampling can be implemented either by:

- Producing a strata based on the prediction error map (e.g. extrapolation areas / areas with highest uncertainty), then sampling only within that strata,
- Using the probability of exceeding threshold mapping accuracy to generate extra sampling points,

In both cases 2nd round points can be then added to the original training dataset and the model for the area can then be refitted (this procedure in data science is also referred to as *"re-analysis"*). Assuming that our initial model was unbiased, then even few tens of extra points could lead to significant reduction in RMSPE. In practice, we might have to also organize the 3rd round sampling until we finally reach some maximum possible accuracy (**?**).

To generate 2nd round sampling for the Edgeroi dataset we can first estimate probability of prediction errors exceeding some threshold probability e.g. RMSE=0.2. We first load point data and prediction error map produced in the previous example using Ensemble Machine Learning:

```
data(edgeroi)
edgeroi.sp <- edgeroi$sites
coordinates(edgeroi.sp) <- ~ LONGDA94 + LATGDA94
proj4string(edgeroi.sp) <- CRS("+proj=longlat +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +no_defs")
edgeroi.sp <- spTransform(edgeroi.sp, CRS("+init=epsg:28355"))
```

The probability of exceeding some threshold, assuming normal distribution of prediction erros can be derived as:

```
t.RMSE = 0.2
edgeroi.error = readGDAL("./output/edgeroi_soc_rmspe.tif")
#> ./output/edgeroi_soc_rmspe.tif has GDAL driver GTiff
#> and has 321 rows and 476 columns
edgeroi.error$inv.prob = 1/(1-2*(pnorm(edgeroi.error$band1/t.RMSE)-0.5))
```

next, we can generate a sampling plan using the **spatstat** package and the inverse probability of the pixels exceeding threshold uncertainty (`f` argument):

```
dens.var <- spatstat.geom::as.im(sp::as.image.SpatialGridDataFrame(edgeroi.error["inv.prob"]))
pnts.new <- spatstat.core::rpoint(50, f=dens.var)
```

So the new sampling plan, assuming adding only 50 new points would thus look like this:

```
plot(log1p(raster(edgeroi.error[c("inv.prob")])))
points(edgeroi.sp, pch="+", cex=.8, col="blue")
points(as(pnts.new, "SpatialPoints"), pch=19, cex=0.8, col="black")
```

This puts much higher weight at locations where the prediction errors are higher, however, technically speaking we finish still sampling points across the whole study area and, of course, including randomization.

By adding 2nd, 3rd round sampling we can imagine that the mapping accuracy / RMSE would gradually decrease following a decay function (e.g. `RMSE = b1 * x ^ -b1`) until some maximum possible accuracy is achieved. This way soil surveyor can optimize the predictions and reduce costs by minimizing number of additional samples i.e. it could be considered *the shortest path* to increasing the mapping accuracy without a need to start sampling from scratch.

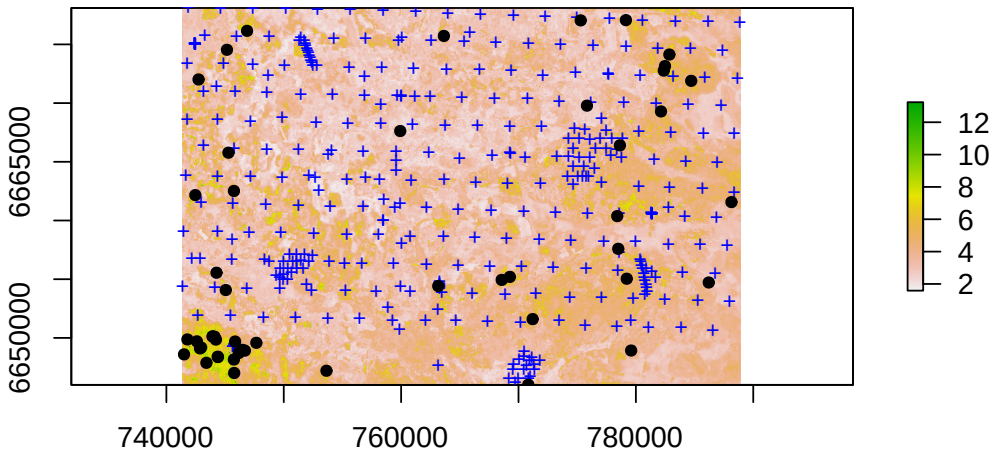


Fig. 4.1 Uncertainty guided 2nd round sampling: locations of initial (+) and 2nd round points (dots).

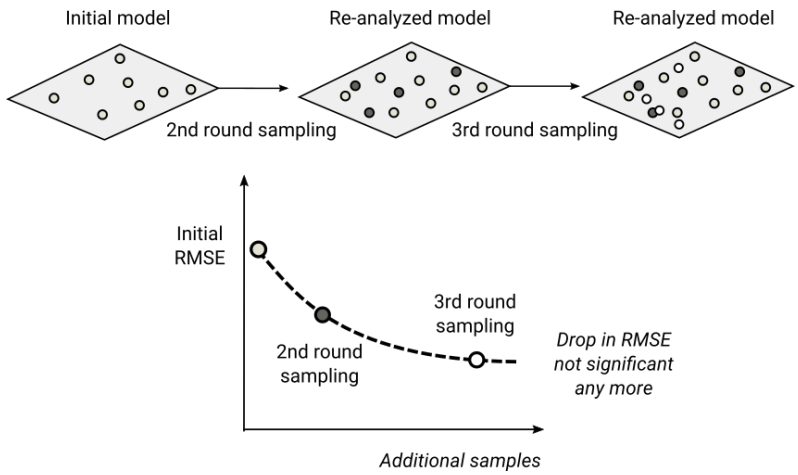


Fig. 4.2 General scheme for re-sampling and re-analysis using uncertainty guided sampling principles.

Chapter 5
Summary notes

You are reading the work-in-progress Spatial Sampling and Resampling for Machine Learning. This chapter is currently draft version, a peer-review publication is pending. You can find the polished first edition at https://opengeohub.github.io/spatial-sampling-ml/.

5.1 Which sampling algorithm to choose?

In this tutorial we have demonstrated some main steps required to analyze existing sample designs (point patterns) and compare them with sampling algorithms such as the SRC, LHS and FSCS. Some general conclusions are:

- Understanding limitations of spatial samples used for ML is important. Diversity of tools now exist that allow for sampling diagnostics, especially to determine spatial clustering, potential extrapolation areas, to test Complete Spatial Randomness etc;
- Ensemble Machine Learning is at the order of magnitude more computational, but using combination of simple and complex base learners and spatial blocking seem to help produce models with less artifacts in extrapolation space and which report a more realistic mapping accuracy than if spatial clustering is ignored;
- The **forestError**[1] package seems to provide a complete framework for uncertainty assessment and can be used to derive the prediction errors (RMSPE) *per-pixel* i.e. for each new prediction location; the average prediction error of the whole area is the mean prediction error that one can report to the users as the best unbiased estimate of the mean uncertainty;

Figure below shows differences between the above mentioned sampling algorithms in both geographical and feature spaces. In this case: actual sampling is significantly missing the whole cluster in feature space, while FSCS seems to show the highest spread in the feature space and by many authors is recognized as the most advantageous sampling design for predictive mapping

[1] https://rdrr.io/cran/forestError/

(**?**). Such sampling diagnostics / comparisons geographical vs feature space help us detect any possible problems before we start running ML.

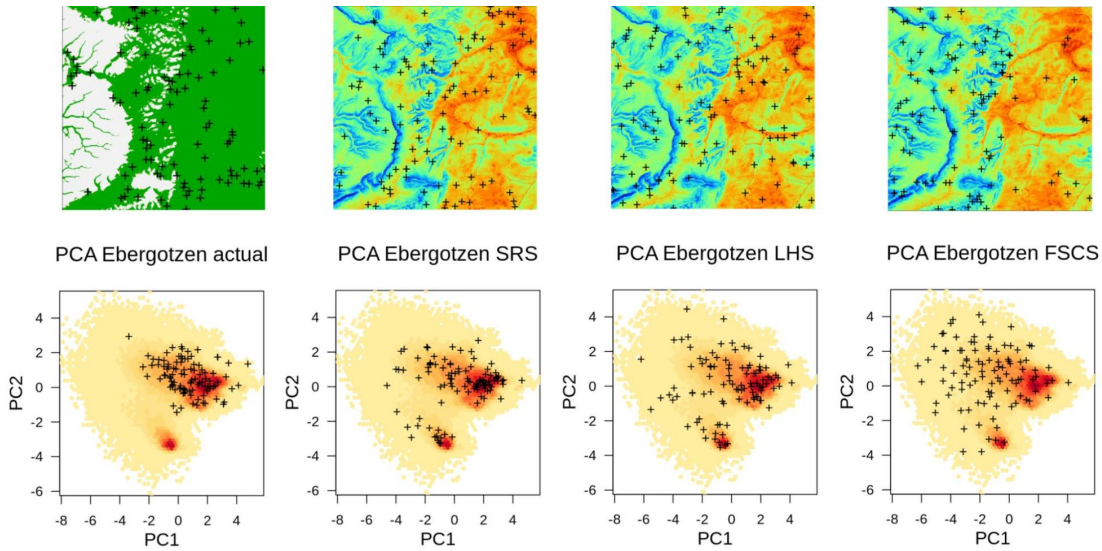


Fig. 5.1 Summary comparison of sampling designs: convenience sampling (actual), Simple Random Sample (SRS), Latin Hypercube Sampling (LHS), and Feature Space Coverage Sampling (FSCS). Points shown in geographical (above) and feature space (below; with first 2 principal components as x, y coordinates).

5.2 Sampling in a new area

Recommended steps to prepare a sampling plan include:

1. Prepare all covariate layers (rasters) that you plan to use to fit predictive mapping models; import them to R;
2. Convert covariate layers to Principal Components using the **landmap::spc**[2] function;
3. Cluster the feature space using the **h2o.kmeans**[3] function; for smaller number of samples use number of clusters equal to number of sampling locations;
4. Generate a sampling design and export the points to **GPX format**[4] so they can be imported to a hand-held GPS or similar. For fieldwork we recommend using the **ViewRanger app**[5] which has useful functionality for field work including planning the optimal routes.

[2] https://rdrr.io/cran/landmap/man/spc.html

[3] https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/k-means.html

[4] https://nl.wikipedia.org/wiki/GPS_Exchange_Format

[5] https://play.google.com/store/apps/details?id=com.augmentra.viewranger.android&hl=en&gl=US

If you are collecting more than a few hundred points, then FSCS could become cumbersome and we hence recommend using LHS sampling. This sampling algorithm spreads points symmetrically in the feature space and ensures that the extrapolation (in feature space) is minimized.

5.3 ML on clustered point samples

Assuming that there is significant spatial and/or feature space clustering in training points, it appears that various blocking / Cross-Validation strategies, especially based on Ensemble ML help produce more balanced estimate of regression parameters and of the mapping accuracy. Incorporation of spatial proximity i.e. autocorrelation has roots in the **Generalized Least Squares methods** (**?**) and in the classical data science papers e.g. by **?**. Ensemble ML with spatial blocking comes, however, at the costs of the order of magnitude higher computing costs.

In theory, even the most clustered point datasets can be used to fit predictive mapping models, however, it is then important to use modeling methods that account for clustering and prevent doing over-fitting and/or producing not realistic measures of mapping accuracy. Eventually, very biased point samples totally missing 50% of the feature / geographical space would be of limited use for producing predictions, but could still be used to get some initial estimates.

? shows that, assuming that training points are based on the probability sampling i.e. SRS, there is no need for spatial blocking i.e. regardless of the spatial dependence structure in the target variable, any subset of SRS would give an unbiased estimator of the mean population parameters. Many spatial statisticians hence argue that mapping accuracy can be determined only by collecting data using probability sampling (**?**). Indeed, we also recommend to users of these tutorials to try your best to generate sampling designs using LHS, FSCS or at least SRS, as this ensures unbiased derivation of population parameters. Here the book by **?** seems to be a valuable resource as it also provides practical instructions[6] for a diversity of data types.

If you have a dataset that you have used to test sampling and resampling, please share your experiences by posting an issue[7] and/or providing a screenshot of your results.

[6] https://github.com/DickBrus/SpatialSamplingwithR

[7] https://github.com/OpenGeoHub/spatial-sampling-ml/

Chapter 6
References