Tom Hengl, Leandro Parente and Carmelo Bonannella

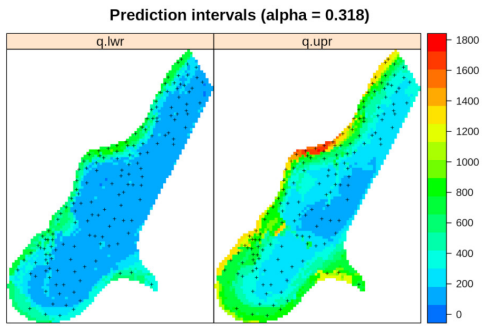
Spatial and spatiotemporal interpolation using Ensemble Machine Learning

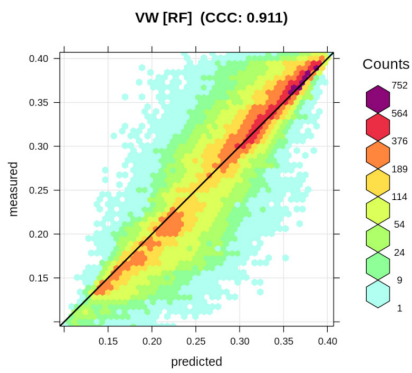
OpenGeoHub foundation, Wageningen, Netherlands

Contents

Introduction

Ensemble Machine Learning





[1] This Rmarkdown tutorial[2] provides practical instructions, illustrated with sample dataset, on how to use Ensemble Machine Learning to generate predictions (maps) from 2D, 3D, 2D+T (spatiotemporal) training (point) datasets. We show functionality to do automated benchmarking for spatial/spatiotemporal prediction problems, and for which we use primarily the mlr framework and spatial packages terra, rgdal and similar..

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

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

Ensembles are predictive models that combine predictions from two or more learners (Seni and Elder, 2010; Zhang and Ma, 2012). The specific benefits of using Ensemble learners are:

- **Performance**: they can help improve the average prediction performance over any individual contributing learner in the ensemble.
- **Robustness**: they can help reduce extrapolation / overshooting effects of individual learners.
- **Unbiasness**: they can help determine a model-free estimate of prediction errors.

Even the most flexible and best performing learners such as Random Forest or neural networks always carry a bias in the sense that the fitting produces recognizable patterns and these are limited by the properties of the algorithm. In the case of ensembles, the modeling algorithm becomes secondary, and even though the improvements in accuracy are often minor as compared to the best individual learner, there is a good chance that the final EML model will be less prone to overshooting and extrapolation problems.

There are in principle three ways to apply ensembles (Zhang and Ma, 2012):

- *bagging*: learn in parallel, then combine using some deterministic principle (e.g. weighted averaging),
- *boosting*: learn sequentially in an adaptive way, then combine using some deterministic principle,
- *stacking*: learn in parallel, then fit a meta-model to predict ensemble estimates,

The *"meta-model"* is an additional model that basically combines all individual or *"base learners"*. In this tutorial we focus only on the stacking approach to Ensemble ML.

There are several packages in R that implement Ensemble ML, for example:

- SuperLearner[3] package,
- caretEnsemble[4] package,
- h2o.stackedEnsemble[5] package,
- mlr[6] and mlr3[7] packages,

Ensemble ML is also available in Python through the scikit-learn[8] library.

In this tutorial we focus primarily on using the mlr package[9], i.e. a wrapper functions to mlr implemented in the landmap package.

[3] https://cran.r-project.org/web/packages/SuperLearner/vignettes/Guide-to-SuperLearner.html
[4] https://cran.r-project.org/web/packages/caretEnsemble/vignettes/caretEnsemble-intro.html
[5] http://docs.h2o.ai/h2o-tutorials/latest-stable/tutorials/ensembles-stacking/index.html
[6] https://mlr.mlr-org.com/reference/makeStackedLearner.html
[7] https://mlr3gallery.mlr-org.com/posts/2020-04-27-tuning-stacking/
[8] https://scikit-learn.org/stable/modules/ensemble.html
[9] https://mlr.mlr-org.com/

Using geographical distances to improve spatial interpolation

Machine Learning was for long time been considered suboptimal for spatial interpolation problems, in comparison to classifical geostatistical techniques such as kriging, because it basically ignores spatial dependence structure in the data. To incorporate spatial dependence structures in machine learning, one can now add the so-called "geographical features": buffer distance, oblique distances, and/or distances in the watershed, as features. This has shown to improve prediction performance and produce maps that visually appear as they have been produced by kriging (Hengl et al, 2018).

Use of geographical as features in machine learning for spatial predictions is explained in detail in:

- Behrens, T., Schmidt, K., Viscarra Rossel, R. A., Gries, P., Scholten, T., & MacMillan, R. A. (2018). Spatial modelling with Euclidean distance fields and machine learning[10]. European journal of soil science, 69(5), 757-770.
- Hengl, T., Nussbaum, M., Wright, M. N., Heuvelink, G. B., & Gräler, B. (2018). Random forest as a generic framework for predictive modeling of spatial and spatio-temporal variables[11]. PeerJ, 6, e5518. https://doi.org/10.7717/peerj.5518

- Møller, A. B., Beucher, A. M., Pouladi, N., and Greve, M. H. (2020). Oblique geographic coordinates as covariates for digital soil mapping[12]. SOIL, 6, 269–289, https://doi.org/10.5194/soil-6-269-2020
- Sekulić, A., Kilibarda, M., Heuvelink, G.B., Nikolić, M., Bajat, B. (2020). Random Forest Spatial Interpolation[13]. Remote Sens. 12, 1687. https://doi.org/10.3390/rs12101687

In the case the number of covariates / features becomes large, and assuming the covariates are diverse, and that the points are equally spread in an area of interest, there is probably no need for using geographical distances in model training because unique combinations of features become so large that they can be used to represent *geographical position* (Hengl et al, 2018).

Installing the landmap package

To install the most recent landmap package from Github use:

```
library(devtools)
install_github("envirometrix/landmap")
```

[10] https://doi.org/10.1111/ejss.12687
[11] https://doi.org/10.7717/peerj.5518
[12] https://doi.org/10.5194/soil-6-269-2020
[13] https://doi.org/10.3390/rs12101687

Important literature

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

- Becker, M. et al.: **"mlr3 book"**[14];
- Bivand, R., Pebesma, E. and Gómez-Rubio, V.: **"Applied Spatial Data Analysis with R"**[15];
- Irizarry, R.A.: **"Introduction to Data Science: Data Analysis and Prediction Algorithms with R"**[16];
- Kuhn, M.: **"The caret package"**[17];
- Molnar, C.: **"Interpretable Machine Learning: A Guide for Making Black Box Models Explainable"**[18];
- Lovelace, R., Nowosad, J. and Muenchow, J.: **"Geocomputation with R"**[19];

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[20], which matches in functionality what is available via the mlr package in R.

License

[21]

[14] https://mlr3book.mlr-org.com/
[15] https://asdar-book.org/
[16] https://rafalab.github.io/dsbook/
[17] https://topepo.github.io/caret/
[18] https://christophm.github.io/interpretable-ml-book/
[19] https://geocompr.robinlovelace.net/
[20] https://scikit-learn.org/stable/
[21] http://creativecommons.org/licenses/by-sa/4.0/
[22] http://creativecommons.org/licenses/by-sa/4.0/

Acknowledgements

 This tutorial is based on the **"R for Data Science"**[23] book by Hadley Wickham and contributors.

OpenLandMap[24] is a collaborative effort and many people have contributed data, software, fixes and improvements via pull request.

OpenGeoHub[25] is an independent not-for-profit research foundation promoting Open Source and Open Data solutions. These tools were developed primarily for the need of the Geo-harmonizer project and to enable generation of next-generation environmental layers for continental Europe (Witjes et al, 2022?; Bonannella et al, 2022?). **EnvirometriX Ltd.**[26] is the commercial branch of the group responsible for designing soil sampling designs for the **AgriCapture**[27] and similar soil monitoring projects.

[28]

OpenDataScience.eu[29] project is co-financed by the European Union (**CEF Telecom project 2018-EU-IA-0095**[30]).

[23] https://r4ds.had.co.nz/
[24] https://openlandmap.org
[25] https://opengeohub.org
[26] https://envirometrix.nl
[27] https://agricaptureco2.eu/
[28] https://opengeohub.org
[29] https://opendatascience.eu/
[30] https://ec.europa.eu/inea/en/connecting-europe-facility/cef-telecom/2018-eu-ia-0095

Chapter 1

Introduction to spatial and spatiotemporal data

You are reading the work-in-progress Spatial and spatiotemporal interpolation using Ensemble 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-prediction-eml/.

1.1 Spatial data and spatial interpolation

Spatial and/or geospatial data is any data that is spatially referenced (in the case of geographical data referenced to Earth surface) i.e. X and Y coordinates are known. With the implementation of the GPS and Earth Observation technology, almost everything is becoming spatial data and hence tools such as Geographical Information Systems (GIS) and spatial analysis tools to process, analyze and visualize geospatial data are becoming essential[1].

Spatial interpolation and/or **Spatial Prediction** is a process of estimating values of the target variable over the whole area of interest by using some input training point data, algorithm and values of the covariates at new locations (Mitas and Mitasova, 1999). Interpolation results in images or maps, which can then be used for decision making or similar. There is a difference between *interpolation* and *prediction*: *prediction* can imply both interpolation and extrapolation. We will more commonly use the term *spatial prediction* in this tutorial, even though the term *spatial interpolation* has been more widely accepted (Mitas and Mitasova, 1999). In geostatistics, e.g. in the case of **ordinary kriging**, interpolation corresponds to cases where the location being estimated is surrounded by the sampling locations and is within the spatial auto-correlation range (Diggle and Ribeiro Jr, 2007). Prediction outside of the practical range (i.e. where prediction error exceeds the global variance) is referred to as **spatial extrapolation**. In other words, extrapolation is prediction at locations where we do not have enough statistical evidence (based on the statistical model) to make significant predictions.

[1] https://towardsdatascience.com/the-impact-of-geospatial-features-on-machine-learning-3a71c99f080a

1.2 Spatial interpolation using Ensemble Machine Learning

Ensemble Machine Learning (Ensemble ML) is an approach to modeling where, instead of using a single best learner, we use multiple **strong learners** and then combine their predictive capabilities into a single union. This can both lead to higher accuracy and robustness (Seni and Elder, 2010), but also helps with deriving model-free estimate of prediction errors through non-parametric techniques such as bootstrapping (Zhang and Ma, 2012). This way we can help decrease some methodological disadvantages of individual learners as shown in the previous example with synthetic data.

Ensemble ML can be used to fit models and generate predictions using points data the same way ordinary kriging is used to generate interpolations. Ensemble ML for predictive mapping in 2D and 3D is discussed in detail in the first chapter of the tutorial. Spatiotemporal interpolation using EML (2D+T, 3D+T) is at the order of magnitude more computational (Gasch et al, 2015) but it follows the same logic.

Ensemble ML based on stacking is implemented in the mlr package (Bischl et al, 2016) and can be initiated via the `makeStackedLearner` function e.g.:

```
m = mlr::makeStackedLearner(base.learners = lrns,
              super.learner = "regr.ml", method = "stack.cv")
```

here the base learner predictions will be computed by 5-fold cross-validation (repeated re-fitting) and then used to determine the meta-learner. This algorithm is known as the *"Super Learner"*[2] algorithm (Polley and van der Laan, 2010).

In the case of spatial prediction, we also want to *block* training points based on spatial proximity to prevent from producing bias predictions. For this we should know the range of spatial dependence for the model residuals or similar i.e. something that can be derived by fitting a variogram, then limit the minimum spatial distance between training and validation points to avoid overfitting or similar (for more info refer to the Spatial Sampling tutorial[3]).

To automate fitting of an Ensemble Machine Learning models for the purpose of spatial interpolation / prediction, one can now use the landmap[4] package that combines:

- derivation of geographical distances,
- conversion of grids to principal components,
- automated filling of gaps in gridded data,
- automated fitting of variogram and determination of spatial auto-correlation structure,
- spatial overlay,

[2] https://machinelearningmastery.com/super-learner-ensemble-in-python/

[3] https://opengeohub.github.io/spatial-sampling-ml/

[4] https://github.com/Envirometrix/landmap

- model training using spatial Cross-Validation (Lovelace et al, 2019),
- model stacking i.e. fitting of the final EML,

The concept of automating spatial interpolation until the level that almost no human interaction is required is referred to as **"automated mapping"** or automated spatial interpolation (Pebesma et al, 2011).

1.3 Spatiotemporal data

Spatiotemporal data is practically any data that is referenced in both space and time. This implies that the following *coordinates* are known:

- geographic location (longitude and latitude or projected X, Y coordinates);
- spatial location accuracy or size of the block / volume in the case of bulking of samples;
- height above the ground surface (elevation);
- start and end time of measurement (year, month, day, hour, minute etc.);

Consider for example daily temperature measured at some meteorological station. This would have the following coordinates:

```
temp = 22
lat = 44.56123
lon = 19.27734
delta.xy = 30
begin.time = "2013-09-02 00:00:00 CEST"
end.time = "2013-09-03 00:00:00 CEST"
```

which means that the measurement is fully spatiotemporally referenced with both X, Y location defined, `delta.xy` location accuracy known, and begin and end time of measurement specified (in this case temporal support is 1 day).

Analysis of spatiotemporal data is somewhat different from pure spatial analysis. Time is of course NOT *just another* spatial dimension i.e. it has specific properties and different statistical assumptions and methods apply to spatiotemporal data. For an introduction to spatiotemporal data in R please refer to the **spacetime** package tutorial (Pebesma, 2012).

Conceptually speaking, spatiotemporal datasets and corresponding databases can be matched with the two major groups of features (Erwig et al, 1999): (1) **moving or dynamic objects** (discrete or vector geometries), and (2) dynamic **regions** (fields or continuous features). Distinct objects (entities) such as people, animals, vehicles and similar are best represented using vectors and **trajectories** (movement through time), and fields are commonly
represented using **gridded structures**. In the case of working with fields, we basically map either:

- dynamic changes in quantity or density of some material or chemical element,
- energy flux or any similar physical measurements,
- dynamic changes in probability of occurrence of some feature or object,

Spatiotemporal data can be best visualized 2D+T plots **space-time cubes**. One example of a spacetime cube is the following plot in Fig. 1.1 (Hengl et al, 2012, 2015).

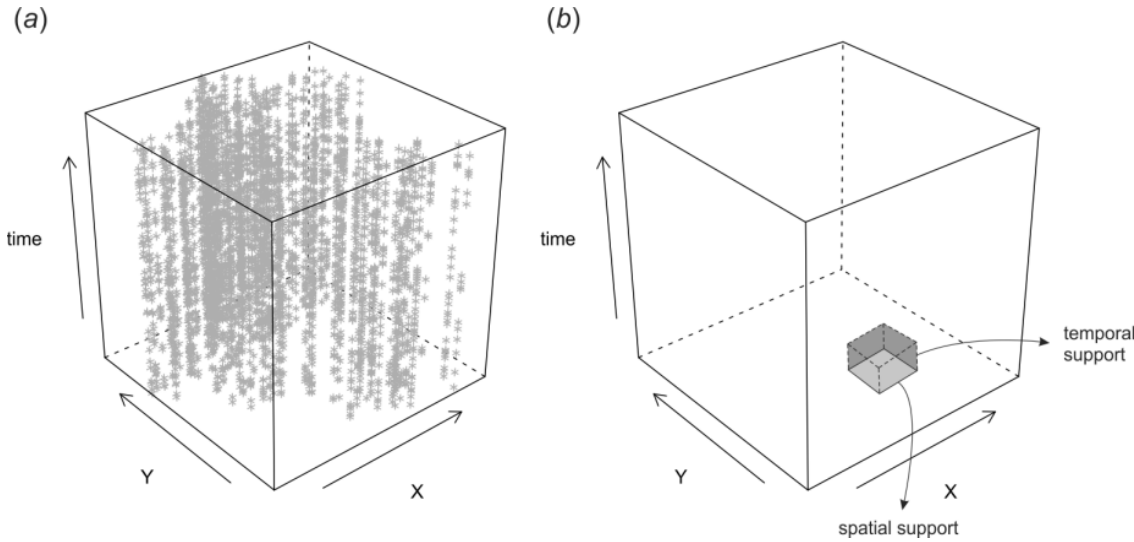


Fig. 1.1 Space-time cube visualized in R: (a) cloud plot showing location of meteorological stations in Croatia, (b) illustration of spatial and temporal support in the space-time cube.

The plot above shows distribution of meteorological stations over Croatia, and then repeated measurements through time. This dataset is further used in the use-case examples to produce spatiotemporal predictions of daily temperatures.

1.4 Time-series analysis

Field of statistics dealing with modeling changes of variables through time, including predicting values beyond the training data (forecasting) is **time-series analysis**. Some systematic guides on how to run time-series analysis in R can be found here[5].

How a variable varies through time (time-series curves) can often be drastically different from how it changes in space (spatial patterns). In general, one can say that, for many environmental variables, variation of values through time can be separated into **components** such as:

[5] http://r-statistics.co/Time-Series-Analysis-With-R.html

- Long-term component (**trend**) determined by long-term geological and extraterrestrial processes,
- Seasonal monthly and/or daily component (**seasonality**) determined by Earth rotation and incoming sun radiation,
- **Variation** component which can be due to chaotic behavior and/or local factors (hence *autocorrelated*), and
- **Pure noise** i.e. measurement errors and similar,

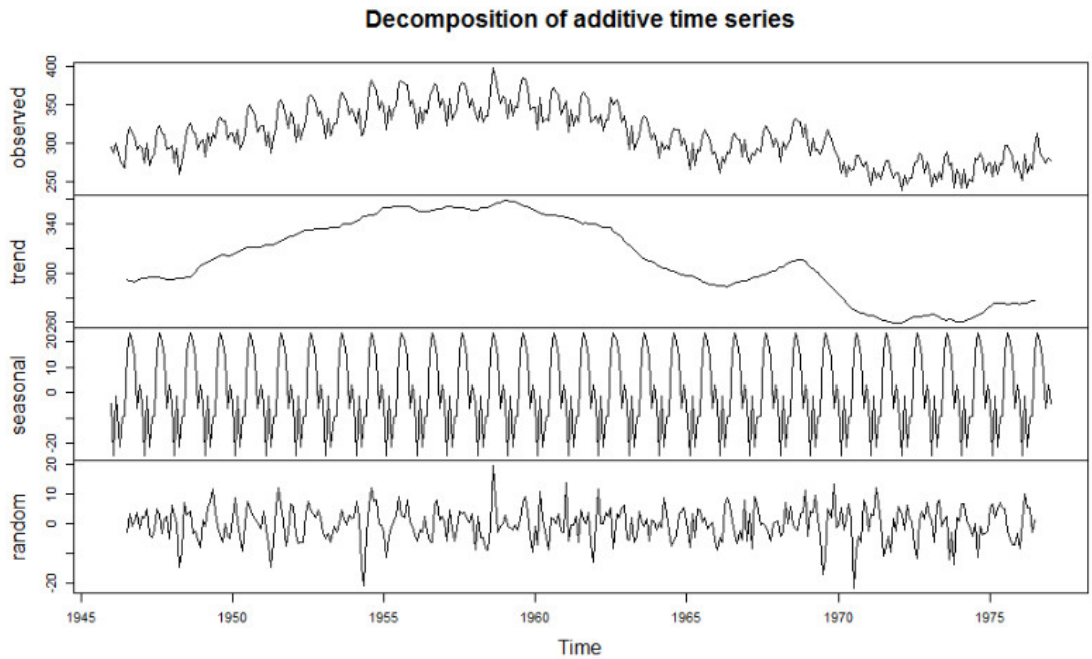


Fig. 1.2 Illustration of decomposition of time-series into: (1) trend, (2) seasonality, and (3) random.

Consider for example the case of the land surface temperature. The long-term component is determined by variations in Earth's orbit and/or Sun's energy output resulting in gradual drops and rises of global mean temperature (glacials and interglacials[6]). Fig. 1.3 shows example of a global temperature reconstruction from proxy data of Marcott et al (2013).

Seasonal i.e. monthly and daily components of variation of land surface temperature are also quite systematic. They are basically determined by Earth's rotation and angles of Sun in relation to Earth's surface. This is a relatively stable pattern that looks like sinusoidal curves or similar. The plot below shows variation of values of soil moisture and soil temperature at one meteo station in USA across multiple years (Gasch et al, 2015).

The data set in Fig. 1.4 is further discussed in the case studies to demonstrate 3D+T spatiotemporal modeling (Gasch et al, 2015). As we will see later, the seasonal daily and monthly part of variation is systematic and can be modeling using latitude, altitude and time/day of the year.

[6] https://en.wikipedia.org/wiki/Ice_age

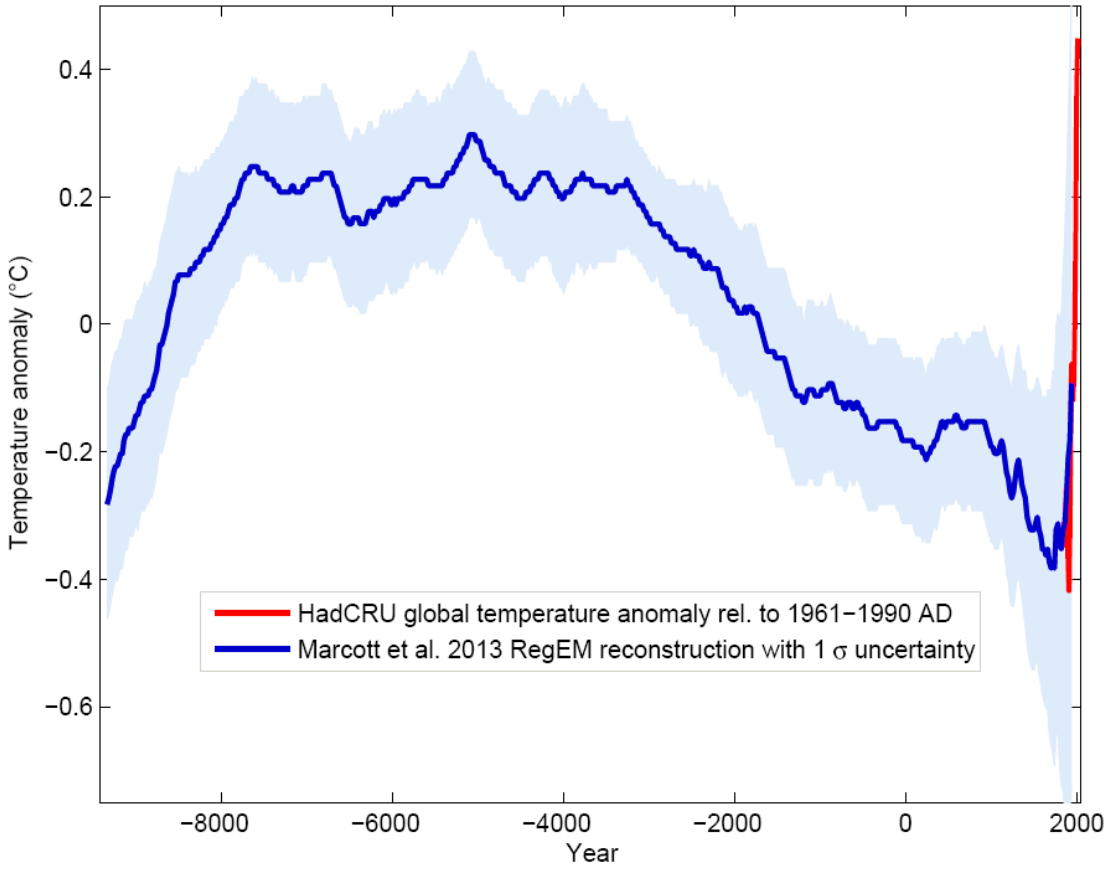


Fig. 1.3 Global temperature reconstruction. This shows how global temperature varies on a long-term term scale. Graph by: Klaus Bitterman.

1.5 Visualizing spatiotemporal data

Spatial data is usually visualized using static or interactive maps (see e.g. mapview[7] and/or tmap package[8]). Spatiotemporal data (2D+T) is more complex to visualize than 2D data, while 3D+T data can even require special software (Hengl et al, 2015) before users can make any seamless interpretation.

There are three possible groups of ways to visualize spatiotemporal data:

1. Using **static images** showing trend parameters together with time-series plots at selected representative point locations.

[7] https://r-spatial.github.io/mapview/

[8] https://cran.r-project.org/web/packages/tmap/vignettes/tmap-getstarted.html

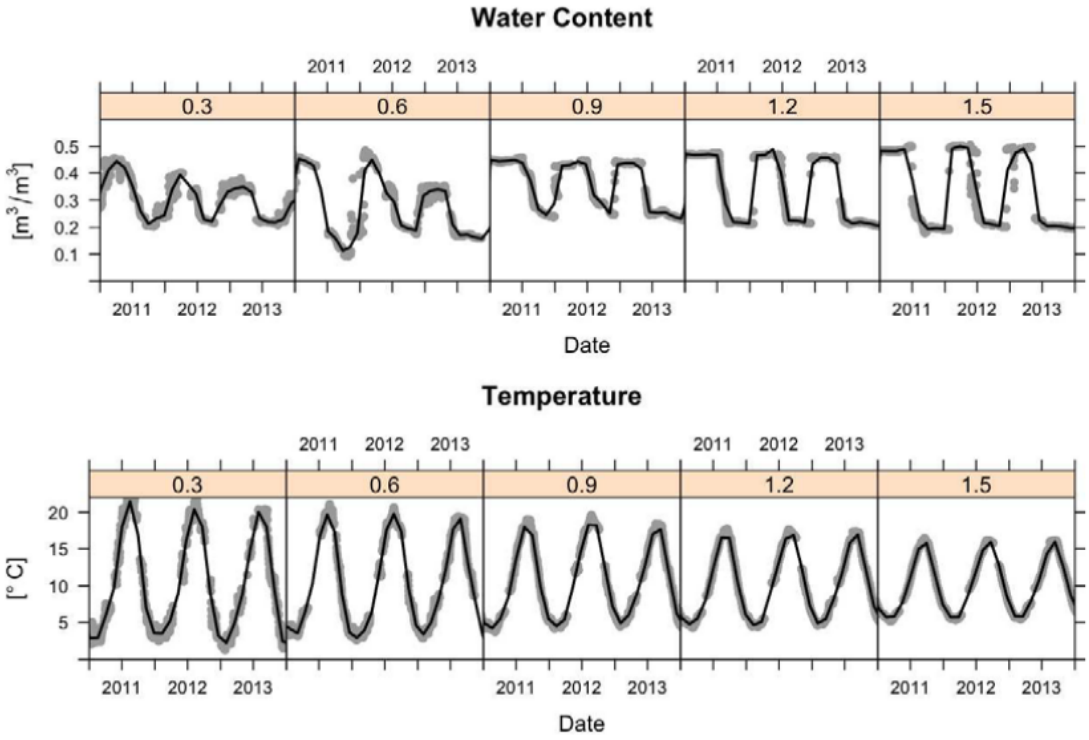


Fig. 1.4 Sensor values from five depths (0.3, 0.6, 0.9, 1.2, and 1.5 m) at one station at Cook Agronomy Farm from January 2011–January 2014. The black line indicates locally fitted splines.

2. Using **time-slices** or series of visualizations of the same spatial domain but changing in time (time-lapses).

3. Using **animations** or **interactive plots with time-sliders** allowing users to choose *speed* and *direction* of animation.

For an introduction to visualizing spatiotemporal and time-series data refer to Lamigueiro (2014). More complex visualization of spatiotemporal / dynamic geographic features is possible by using the https://geemap.org/ package (a Python package for interactive mapping with Google Earth Engine, ipyleaflet, and ipywidgets).

OpenLandMap.org also has multiple temporal datasets and users can interactive with the time-dimension by using time-slider implemented in OpenLayers and Geoserver[9] (Kilibarda and Protić, 2019).

https://miro.medium.com/max/700/0*fqWNU4XNbjHE3Uv_

[9] http://osgl.grf.bg.ac.rs/books/gvvk-en/

1.6 Spatiotemporal interpolation

Spatiotemporal interpolation and/or prediction implies that point samples are used to interpolate within the spacetime cube. This obviously assumes that enough point measurements are available, and which are spread in both space and time. We will show in this tutorial how Machine Learning can be used to interpolate values within the spacetime cube using real case-studies. Spatiotemporal interpolation using various kriging methods is implemented in the gstat package[10] (Bivand et al, 2013), but is not addressed in this tutorial.

For success of spatiotemporal interpolation (in terms of prediction accuracy), the key is to recognize systematic component of variation in spacetime, which is usually possible if we can find relationship between the target variable and some EO data that is available as a time-series and covers the same spacetime cube of interest. Once we establish a significant relation between **dynamic target** and **dynamic covariates**, we can use the fitted model to predict anywhere in spacetime cube.

For more in-depth discussion on spatiotemporal data in R please refer to Wikle et al (2019). For in-depth discussion on spatial and spatiotemporal blocking for purpose of modeling building and cross-validation refer to Roberts et al (2017).

1.7 Modeling seasonal components

Seasonality is the characteristic of the target variable to follow cyclical patterns such as in trigonometric functions. Such repeating patterns can happen at different **time-scales**:

- inter-annually,
- monthly or based on a season (spring, summer, autumn, winter),
- daily,
- hourly i.e. day-time and night-time patterns,

The monthly and daily seasonal component of variation is determined by Earth's rotation and Sun's angle. Kilibarda et al (2014) have shown that the seasonal component e.g. geometric Earth surface minimum and maximum daily temperature can be modeled, universally anywhere on globe, by using the following formula:

```
temp.from.geom <- function(fi, day, a=30.419375,
                           b=-15.539232, elev=0, t.grad=0.6) {
  f = ifelse(fi==0, 1e-10, fi)
  costeta = cos( (day-18 )*pi/182.5 + 2^(1-sign(fi) ) *pi)
  cosfi = cos(fi*pi/180 )
```

[10] https://cran.r-project.org/web/packages/gstat/vignettes/spatio-temporal-kriging.pdf

```
  A = cosfi
  B = (1-costeta ) * abs(sin(fi*pi/180 ) )
  x = a*A + b*B - t.grad * elev / 100
  return(x)
}
```

where `day` is the day of year, `fi` is the latitude, the number 18 represents the coldest day in the northern and warmest day in the southern hemisphere, `elev` is the elevation in meter, 0.6 is the vertical temperature gradient per 100-m, and `sign` denotes the *signum* function that extracts the sign of a real number.

This formula accounts for different seasons at southern and northern hemisphere and can be basically applied on gridded surfaces to compute expected temperature at a given day. A simple example of min daily temperature is:

```
temp.from.geom(fi=52, day=120)
#> [1] 8.73603
```

If we plot this function for five consecutive years, we get something similar to the spline-fitted functions in the previous plot:

```
days = seq(1:(5*365))
plot(temp.from.geom(fi=52, day=days))
```

1.8 Predictive mapping using spatial and spatiotemporal ML in R

Standard spatiotemporal ML for predictive mapping typically includes the following steps (Hengl et al, 2018; Hengl and MacMillan, 2019):

1. Prepare training (points) data and data cube with all covariates ideally as an analysis-ready datacube.
2. Overlay points and create a regression-matrix.
3. Fine-tune initial model, reduce complexity and produce production-ready prediction model.
4. Run mapping accuracy assessment and determine prediction uncertainty including the per pixel uncertainty.
5. Generate predictions and save as maps.
6. Visualize predictions using web-GIS solutions.

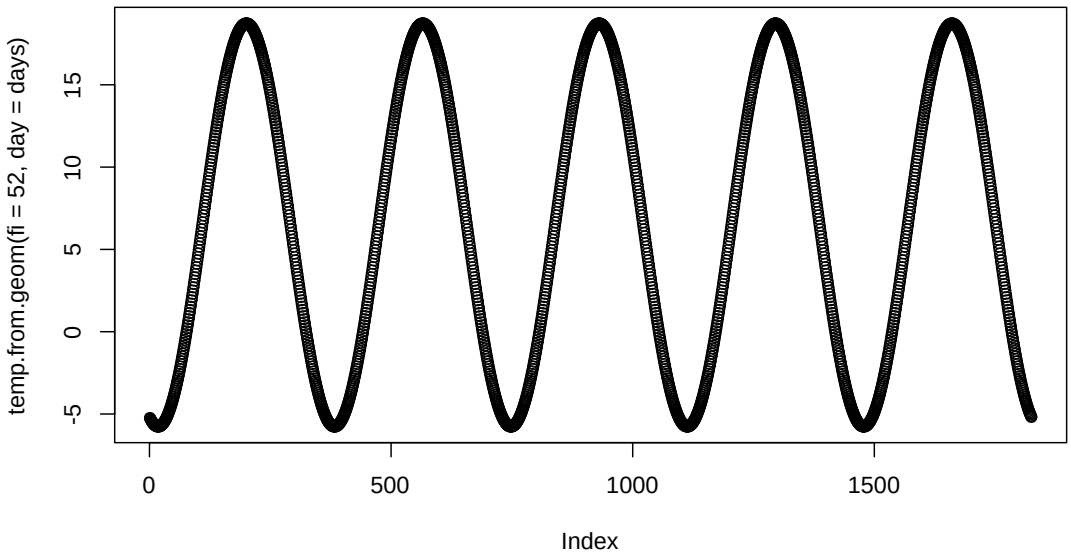


Fig. 1.5 Geometric temperature function plot for a given latitude.

1.9 Extrapolation and over-fitting problems of ML methods

Machine Learning has *defacto* become next-generation applied predictive modeling framework. ML techniques such as **Random Forest** (RF) have proven to over-perform vs more simple linear statistical methods, especially where the datasets are large, complex and target variable follows complex relationship with covariates (Hengl et al, 2018). Random Forest comes at a cost however. There are four main practical disadvantages of RF:

- Depending on data and assumptions about data, it can over-fit values without an analyst even noticing it.
- It predicts well only within the feature space with enough training data. **Extrapolation** i.e. prediction outside the training space can lead to poor performance (Meyer and Pebesma, 2021).
- It can be computationally expensive with computational load increasing exponentially with the number of covariates.
- It requires quality training data and is highly sensitive to blunders and typos in the data.

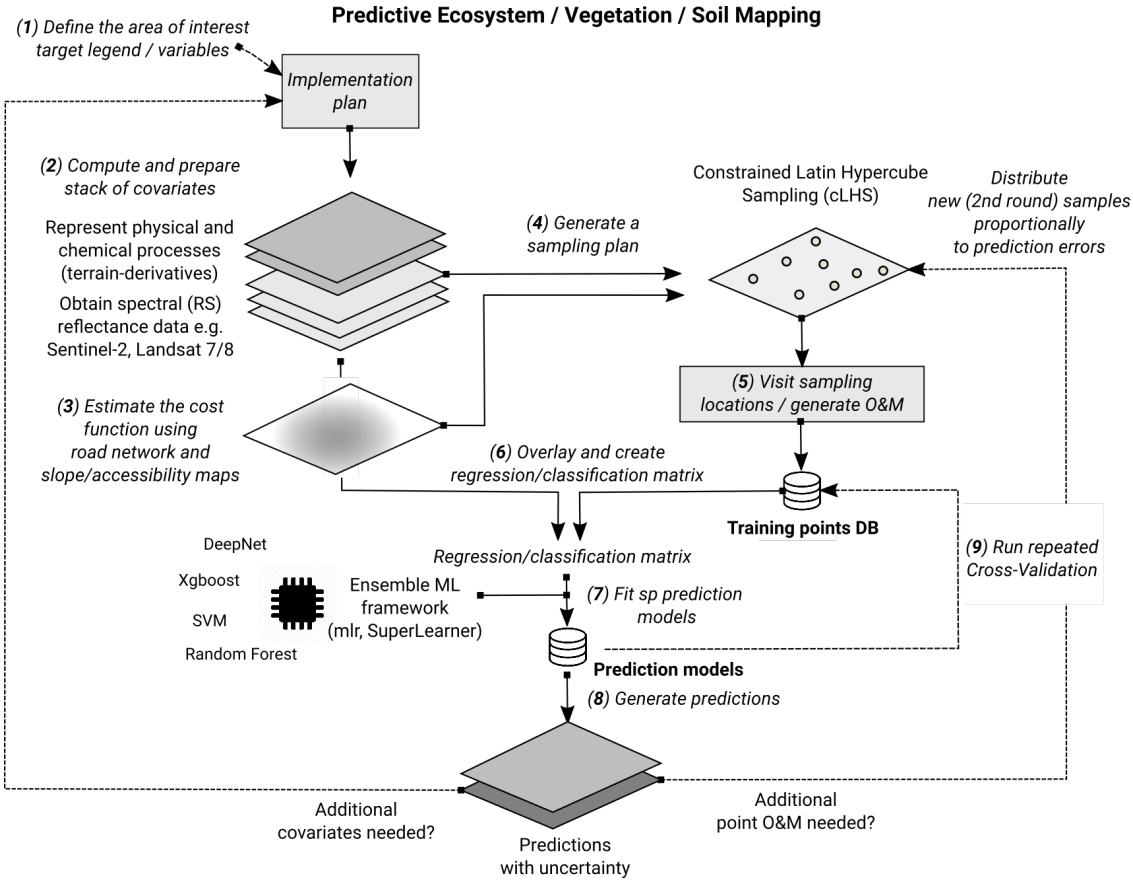


Fig. 1.6 General Machine Learning framework recommended for predictive mapping of vegetation / ecological / soil variables. Assuming full automation of modeling, [2nd-round samples](https://opengeohub.github.io/spatial-sampling-ml/) can be used to gradually improve mapping accuracy.

Read more about extrapolation problems of Random Forest in this post[11].

In the following section we will demonstrate that indeed RF can overfit data and can have serious problems with predicting in the extrapolation space. Consider for example the following small synthetic dataset assuming simple linear relationship (see original post by Dylan Beaudette[12]):

```
set.seed(200)
n = 100
x <- 1:n
y <- x + rnorm(n = 50, mean = 15, sd = 15)
```

If we fit a simple **Ordinary Least Square** model to this data we get:

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

[12] https://twitter.com/DylanBeaudette/status/1410666900581851138

```
m0 <- lm(y ~ x)
summary(m0)
#>
#> Call:
#> lm(formula = y ~ x)
#>
#> Residuals:
#>      Min       1Q   Median       3Q      Max
#> -27.6093  -6.4396   0.6437   6.7742  26.7192
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 14.84655    2.37953   6.239 1.12e-08 ***
#> x            0.97910    0.04091  23.934  < 2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 11.81 on 98 degrees of freedom
#> Multiple R-squared:  0.8539, Adjusted R-squared:  0.8524
#> F-statistic: 572.9 on 1 and 98 DF,  p-value: < 2.2e-16
```

we see that the model explains about 85% of variation in the data and that the RMSE estimated by the model (residual standard error) matches very well the noise component we have inserted on purpose using the `rnorm` function.

If we fit a Random Forest model to this data we get:

```
library(randomForest)
#> randomForest 4.6-14
#> Type rfNews() to see new features/changes/bug fixes.
#>
#> Attaching package: 'randomForest'
#> The following object is masked from 'package:ranger':
#>
#>     importance
rf = randomForest::randomForest(data.frame(x=x), y, nodesize = 5, keep.inbag = TRUE)
rf
#>
#> Call:
#>  randomForest(x = data.frame(x = x), y = y, nodesize = 5, keep.inbag = TRUE)
#>                Type of random forest: regression
#>                      Number of trees: 500
#> No. of variables tried at each split: 1
#>
#>           Mean of squared residuals: 202.6445
#>                     % Var explained: 78.34
```

Next, we can estimate the prediction errors using the method of Lu and Hardin (2021), which is available via the `forestError` package:

```
library(forestError)
rmse <- function(a, b) {  sqrt(mean((a - b)^2)) }
dat <- data.frame(x,y)
newdata <- data.frame(
  x = -100:200
)
newdata$y.lm <- predict(m0, newdata = newdata)
## prediction error from forestError:
quantiles = c((1-.682)/2, 1-(1-.682)/2)
pr.rf = forestError::quantForestError(rf, X.train=data.frame(x=x),
                                      X.test=data.frame(x = -100:200),
                                      Y.train=y, alpha = (1-(quantiles[2]-quantiles[1])))
newdata$y.rf <- predict(rf, newdata = newdata)
rmse.lm <- round(rmse(y, predict(m0)), 1)
rmse.rf <- round(rmse(y, predict(rf)), 1)
rmse.lm; rmse.rf
#> [1] 11.7
#> [1] 14.2
```

This shows that RF estimates higher RMSE than linear model. However, if we visualize the two models against each other we see that indeed RF algorithm seems to over-fit this specific data:

```
leg.txt <- sprintf("%s (%s)", c('lm', 'RF'), c(rmse.lm, rmse.rf))
par(mar = c(0, 0, 0, 0), fg = 'black', bg = 'white')
plot(y ~ x, xlim = c(-25, 125), ylim = c(-50, 150), type = 'n', axes = FALSE)
grid()
points(y ~ x, cex = 1, pch = 16, las = 1)
lines(y.lm ~ x, data = newdata, col = 2, lwd = 2)
lines(y.rf ~ x, data = newdata, col = 4, lwd = 2)
lines(newdata$x, pr.rf$estimates$lower_0.318, lty=2,col=4)
lines(newdata$x, pr.rf$estimates$upper_0.318, lty=2,col=4)
legend('bottom', legend = leg.txt, lwd = 2, lty = 1, col = c(2, 4, 3), horiz = TRUE, title = 'RMSE')
```

RF basically tries to fit relationship even to the **pure noise** component of variation (we know it is pure noise because we have generated it using the `rnorm` function). This is obvious over-fitting as we do not want to model something which is purely random.

Extrapolation would not maybe be so much of a problem in the example above if the prediction intervals from the `forestError` package expressed more realistically that the predictions deviate from the *linear structure* in the data. Assuming that, after the prediction, one would eventually collect ground-truth data for the RF model above, these would probably show that the prediction error / prediction intervals are completely off. Most traditional statisticians would consider these too-narrow and over-optimistic and the fitted line over-fit, and hence any further down the pipeline

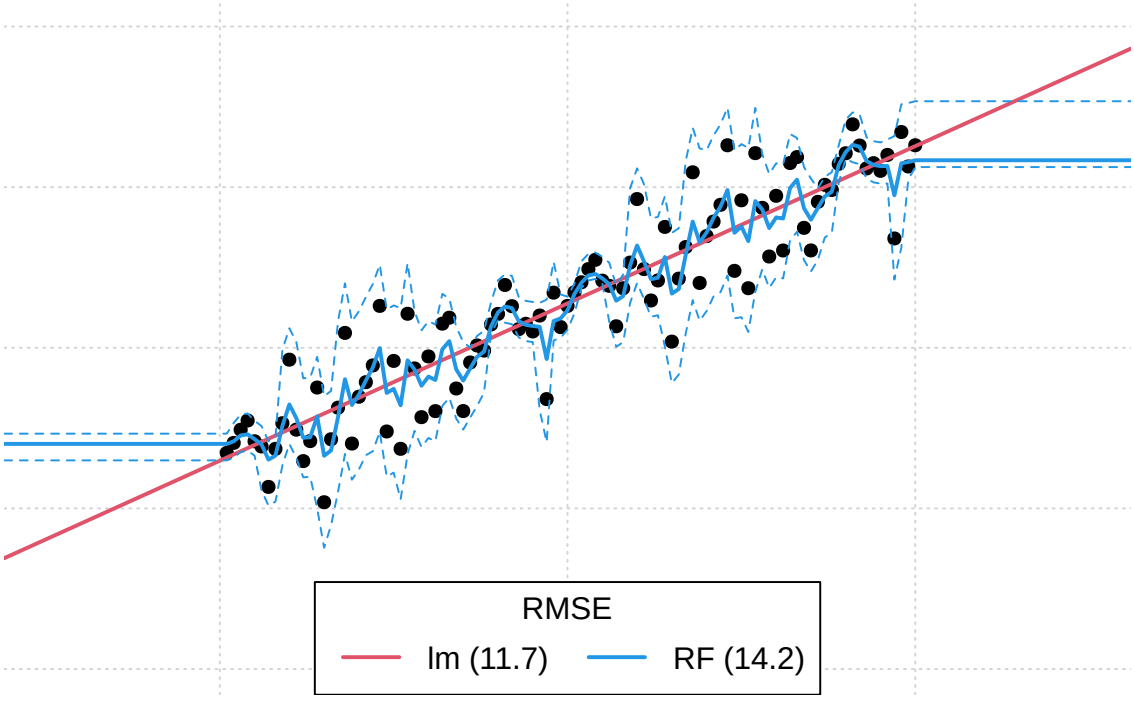


Fig. 1.7 Difference in model fits for sythetic data: lm vs RF. In this case we know that RF (blue line) is overfitting and under-estimating the prediction error in the extrapolation space. Dotted line shows respective 1 std. prediction interval for RF.

over-optimistic prediction uncertainty can result in decision makers being over-confident, leading to wrong decisions, and consequently making users losing any confidence in RF. For an in-depth discussion on extrapolation problems and **Area of Applicability** of Machine Learning models please refer to Meyer and Pebesma (2021).

A possible solution to the problem above is to, instead of using only one learners, we use multiple learners and then apply robust cross-validation that prevents the target model from over-fitting. This can be implemented efficiently, for example, by using the `mlr` package (Bischl et al, 2016). We can run an Ensemble Model by applying the following four steps. First, we define the task of interest and a combination of learners i.e. so-called **base learners**:

```
library(mlr)
library(kernlab)
#>
#> Attaching package: 'kernlab'
#> The following objects are masked from 'package:raster':
#>
#>     buffer, rotated
library(mboost)
#> Loading required package: parallel
#> Loading required package: stabs
```

```
#>
#> Attaching package: 'stabs'
#> The following object is masked from 'package:mlr':
#>
#>     subsample
#> 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 objects are masked from 'package:raster':
#>
#>     cv, extract
library(landmap)
SL.library = c("regr.ranger", "regr.glm", "regr.gamboost", "regr.ksvm")
lrns <- lapply(SL.library, mlr::makeLearner)
tsk <- mlr::makeRegrTask(data = dat, target = "y")
```

In this case we use basically four very different models: RF (`ranger`), linear model (`glm`), Gradient boosting (`gamboost`) and Support Vector Machine (`kvsm`). Second, we train the Ensemble model by using the stacking approach:

```
init.m <- mlr::makeStackedLearner(lrns, method = "stack.cv", super.learner = "regr.lm", resampling=mlr::makeResampl
eml = train(init.m, tsk)
summary(eml$learner.model$super.model$learner.model)
#>
#> Call:
#> stats::lm(formula = f, data = d)
#>
#> Residuals:
#>      Min       1Q   Median       3Q      Max
#> -28.0474  -7.0473  -0.4075   7.2528  28.6083
#>
#> Coefficients:
#>               Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   -0.86179    3.01456  -0.286  0.77560
#> regr.ranger   -0.27294    0.24292  -1.124  0.26402
#> regr.glm       4.75714    1.09051   4.362 3.27e-05 ***
#> regr.gamboost -3.55134    1.14764  -3.094  0.00259 **
#> regr.ksvm      0.08578    0.28482   0.301  0.76394
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 11.25 on 95 degrees of freedom
```

```
#> Multiple R-squared:  0.8714, Adjusted R-squared:  0.8659
#> F-statistic: 160.9 on 4 and 95 DF,  p-value: < 2.2e-16
```

The results show that `ranger` and `ksvm` basically under-perform and are in fact not significant for this specific data i.e. could be probably omitted from modeling.

Note that, for **stacking** of multiple learners we use a separate model (a meta-learner) which is in this case a simple linear model. We use a simple model because we assume that the non-linear relationships have already been modeled via complex models such as `ranger`, `gamboost` and/or `ksvm`.

Next, we need to estimate mapping accuracy and **prediction errors** for Ensemble predictions. This is not trivial as there are no simple derived formulas. We need to use a non-parametric approach basically and this can be very computational. A computationally interesting approach is to first estimate the (global) mapping accuracy, then adjust the prediction variance from multiple base learners:

```
newdata$y.eml = predict(eml, newdata = newdata)$data$response
m.train = eml$learner.model$super.model$learner.model$model
m.terms = eml$learner.model$super.model$learner.model$terms
eml.MSE0 = matrixStats::rowSds(as.matrix(m.train[,all.vars(m.terms)[-1]]), na.rm=TRUE)^2
eml.MSE = deviance(eml$learner.model$super.model$learner.model)/df.residual(eml$learner.model$super.model$learner.m
## correction factor / mass-preservation of MSE
eml.cf = eml.MSE/mean(eml.MSE0, na.rm = TRUE)
eml.cf
#> [1] 9.328646
```

This shows that variance of the learners is about 10 times smaller than the actual CV variance. Again, this proves that many learners try to fit data very closely so that variance of different base learners is often smoothed out.

Next, we can predict values and prediction errors at all new locations:

```
pred = mlr::getStackedBaseLearnerPredictions(eml, newdata=data.frame(x = -100:200))
rf.sd = sqrt(matrixStats::rowSds(as.matrix(as.data.frame(pred)), na.rm=TRUE)^2 * eml.cf)
rmse.eml <- round(sqrt(eml.MSE), 1)
```

and the plot the results of fitting linear model vs EML:

```
leg.txt <- sprintf("%s (%s)", c('lm', 'EML'), c(rmse.lm, rmse.eml))
par(mar = c(0, 0, 0, 0), fg = 'black', bg = 'white')
plot(y ~ x, xlim = c(-25, 125), ylim = c(-50, 150), type = 'n', axes = FALSE)
grid()
points(y ~ x, cex = 1, pch = 16, las = 1)
lines(y.lm ~ x, data = newdata, col = 2, lwd = 2)
lines(y.eml ~ x, data = newdata, col = 4, lwd = 2)
```

```
lines(newdata$x, newdata$y.eml+rmse.eml+rf.sd, lty=2, col=4)
lines(newdata$x, newdata$y.eml-(rmse.eml+rf.sd), lty=2, col=4)
legend('bottom', legend = leg.txt, lwd = 2, lty = 1, col = c(2, 4, 3), horiz = TRUE, title = 'RMSE')
```

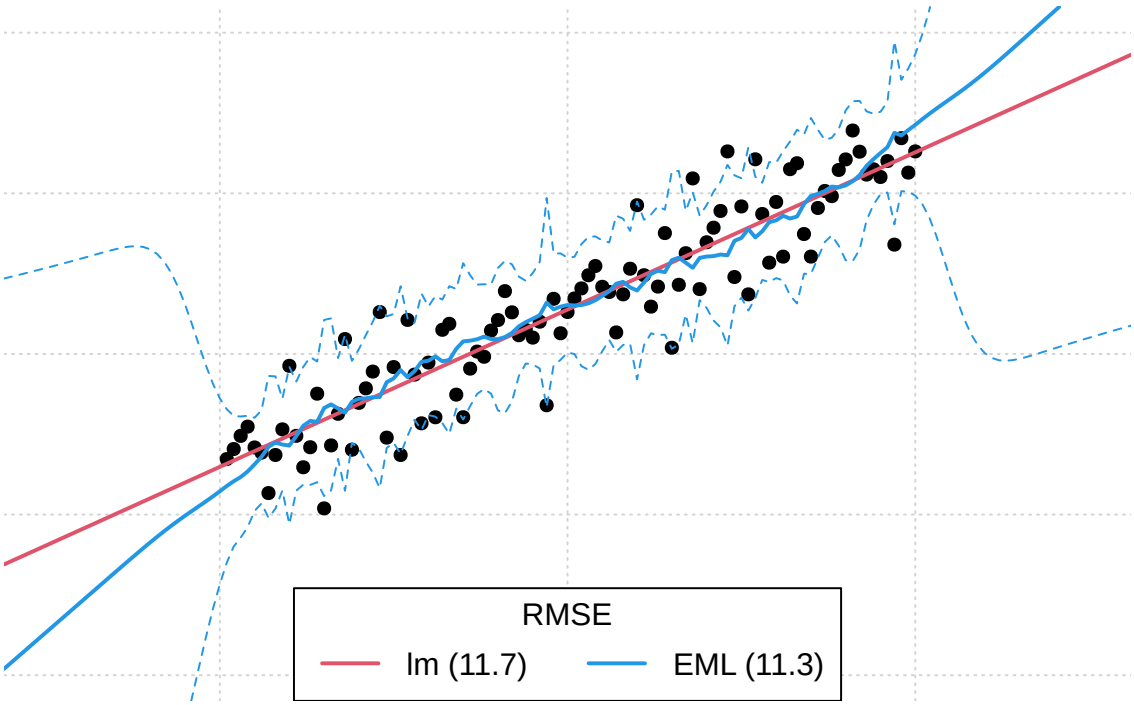


Fig. 1.8 Difference in model fits for sythetic data: lm vs Ensemble ML.

From the plot above, we see that the prediction error intervals in the extrapolation space are now wider (compare with Fig. 1.7), and this reflects much better what we would expect than if we have only used the `forestError` package.

In summary: it appears that combining linear and non-linear tree-based models in an Ensemble ML framework helps both: decrease over-fitting and produce more realistic predictions of uncertainty / prediction intervals. The Ensemble ML framework correctly identifies linear models as being more important than random forest or similar. Hopefully, this provides enough evidence to convince you that Ensemble ML is potentially interesting for use as a generic solution for spatial and spatiotemporal interpolation and extrapolation.

Chapter 2
Spatial interpolation using Ensemble ML

You are reading the work-in-progress Spatial and spatiotemporal interpolation using Ensemble 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-prediction-eml/.

2.1 Spatial interpolation using ML and buffer distances to points

A relatively simple approach to interpolate values from point data using e.g. Random Forest is to use **buffer distances** to all points as covariates. We can here use the meuse dataset for testing (Hengl et al, 2018):

```
library(rgdal)
library(ranger)
library(raster)
library(plotKML)
demo(meuse, echo=FALSE)
grid.dist0 <- landmap::buffer.dist(meuse["zinc"], meuse.grid[1],
                                   classes=as.factor(1:nrow(meuse)))
```

This creates 155 gridded maps i.e. one map per training point. These maps of distances can now be used to predict some target variable by running:

```
dn0 <- paste(names(grid.dist0), collapse="+")
fm0 <- as.formula(paste("zinc ~ ", dn0))
ov.zinc <- over(meuse["zinc"], grid.dist0)
rm.zinc <- cbind(meuse@data["zinc"], ov.zinc)
m.zinc <- ranger(fm0, rm.zinc, num.trees=150, seed=1)
m.zinc
#> Ranger result
```

```
#>
#> Call:
#>  ranger(fm0, rm.zinc, num.trees = 150, seed = 1)
#>
#> Type:                             Regression
#> Number of trees:                  150
#> Sample size:                      155
#> Number of independent variables:  155
#> Mtry:                             12
#> Target node size:                 5
#> Variable importance mode:         none
#> Splitrule:                        variance
#> OOB prediction error (MSE):       67501.48
#> R squared (OOB):                  0.4990359
```

Using this model we can generate and plot predictions using:

```
op <- par(oma=c(0,0,0,1), mar=c(0,0,4,3))
zinc.rfd <- predict(m.zinc, grid.dist0@data)$predictions
meuse.grid$zinc.rfd = zinc.rfd
plot(raster(meuse.grid["zinc.rfd"]), col=R_pal[["rainbow_75"]][4:20],
         main="Predictions RF on buffer distances", axes=FALSE, box=FALSE)
points(meuse, pch="+", cex=.8)
par(op)
```

The resulting predictions produce patterns very much similar to what we would produce if we have used ordinary kriging or similar. Note however that for RFsp model: (1) we did not have to fit any variogram, (2) the model is in essence *over- parameterized* with basically more covariates than training points.

2.2 Spatial interpolation using ML and geographical distances to neighbors

Deriving buffer distances for all points is obviously not suitable for very large point datasets. Sekulić et al (2020) describe an alternative, a more scalable method that uses closest neighbors (and their values) as covariates to predict target variable. This can be implemented using the `meteo` package:

```
library(meteo)
#> Warning: replacing previous import 'caret::MAE' by 'DescTools::MAE' when loading
#> 'meteo'
```

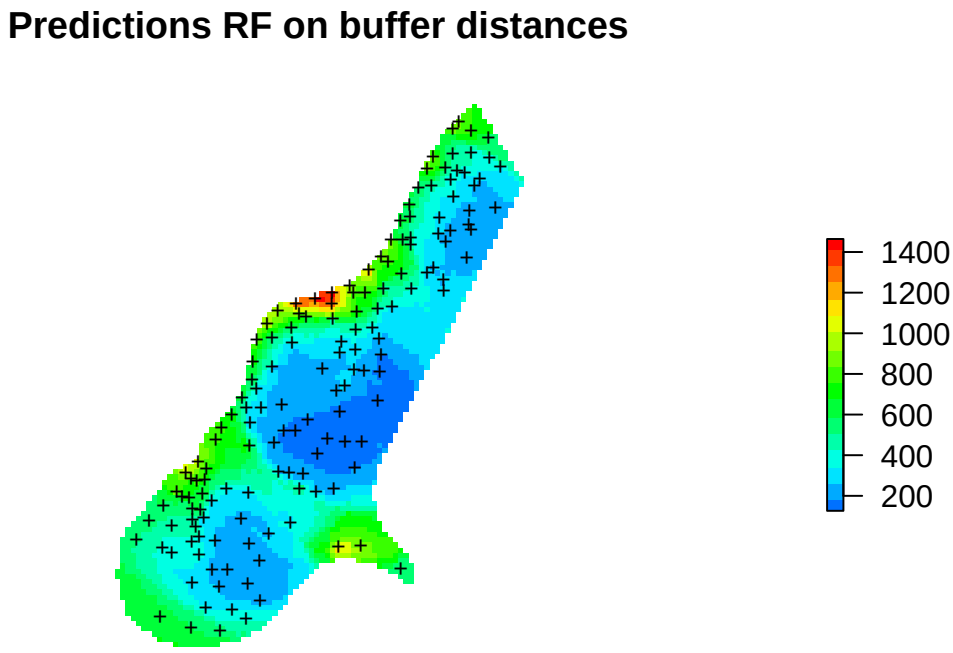


Fig. 2.1 Values of Zinc predicted using only RF on buffer distances.

```
#> Warning: replacing previous import 'caret::RMSE' by 'DescTools::RMSE' when
#> loading 'meteo'
nearest_obs <- meteo::near.obs(locations = meuse.grid,
                               locations.x.y = c("x","y"),
                               observations = meuse, observations.x.y=c("x","y"),
                               zcol = "zinc", n.obs = 10, rm.dupl = TRUE)
#> Warning in if (class(knn1$nn.idx) != "integer") {: the condition has length > 1
#> and only the first element will be used
str(nearest_obs)
#> 'data.frame':    3103 obs. of  20 variables:
#>  $ dist1 : num  168.2 112 139.9 172 56.4 ...
#>  $ dist2 : num  204 165 164 173 127 ...
#>  $ dist3 : num  239 183 210 230 139 ...
#>  $ dist4 : num  282 268 246 241 265 ...
#>  $ dist5 : num  370 331 330 335 297 ...
#>  $ dist6 : num  407 355 370 380 306 ...
#>  $ dist7 : num  429 406 391 388 390 ...
#>  $ dist8 : num  504 448 473 472 393 ...
#>  $ dist9 : num  523 476 484 496 429 ...
#>  $ dist10: num  524 480 488 497 431 ...
#>  $ obs1  : num  1022 1022 1022 640 1022 ...
#>  $ obs2  : num  640 640 640 1022 1141 ...
```

```
#>  $ obs3  : num  1141 1141 1141 257 640 ...
#>  $ obs4  : num  257 257 257 1141 257 ...
#>  $ obs5  : num  346 346 346 346 346 346 346 346 346 346 ...
#>  $ obs6  : num  406 406 406 269 406 406 406 269 257 406 ...
#>  $ obs7  : num  269 269 269 406 269 ...
#>  $ obs8  : num  1096 1096 1096 281 1096 ...
#>  $ obs9  : num  347 347 347 347 504 347 347 279 269 504 ...
#>  $ obs10 : num  281 504 281 279 347 504 279 347 347 347 ...
```

which produces 20 grids showing assigned values from 1st to 10th neighbor and distances. We can plot values based on the first neighbor, which corresponds to using e.g. Voronoi polygons[1]:

```
meuse.gridF = meuse.grid
meuse.gridF@data = nearest_obs
spplot(meuse.gridF[11])
```

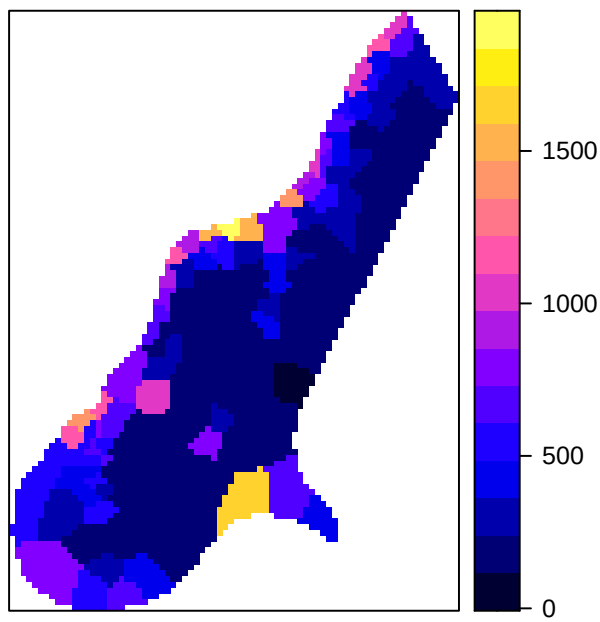


Fig. 2.2 Values of first neighbor for meuse dataset.

Next, we can estimate the same values for training points, but this time we remove any duplicates using `rm.dupl = TRUE`:

[1] https://r-spatial.github.io/sf/reference/geos_unary.html

```
## training points
nearest_obs.dev <- meteo::near.obs(locations = meuse,
                                   locations.x.y = c("x","y"),
                                   observations = meuse,
                                   observations.x.y=c("x","y"),
                                   zcol = "zinc", n.obs = 10, rm.dupl = TRUE)
#> Warning in if (class(knn1$nn.idx) != "integer") {: the condition has length > 1
#> and only the first element will be used
meuse@data <- cbind(meuse@data, nearest_obs.dev)
```

Finally, we can fit a model to predict values purely based on spatial autocorrelation between values (1st to 10th nearest neighbour):

```
fm.RFSI <- as.formula(paste("zinc ~ ", paste(paste0("dist", 1:10), collapse="+"), "+", paste(paste0("obs", 1:10), c
fm.RFSI
#> zinc ~ dist1 + dist2 + dist3 + dist4 + dist5 + dist6 + dist7 +
#>     dist8 + dist9 + dist10 + obs1 + obs2 + obs3 + obs4 + obs5 +
#>     obs6 + obs7 + obs8 + obs9 + obs10
rf_RFSI <- ranger(fm.RFSI, data=meuse@data, importance = "impurity", num.trees = 85, keep.inbag = TRUE)
rf_RFSI
#> Ranger result
#>
#> Call:
#>  ranger(fm.RFSI, data = meuse@data, importance = "impurity", num.trees = 85,      keep.inbag = TRUE)
#>
#> Type:                             Regression
#> Number of trees:                  85
#> Sample size:                      155
#> Number of independent variables:  20
#> Mtry:                             4
#> Target node size:                 5
#> Variable importance mode:         impurity
#> Splitrule:                        variance
#> OOB prediction error (MSE):       65997.22
#> R squared (OOB):                  0.5101999
```

To produce predictions we can run:

```
out = predict(rf_RFSI, meuse.gridF@data)
meuse.grid$zinc.rfsi = out$predictions
op <- par(oma=c(0,0,0,1), mar=c(0,0,4,3))
plot(raster(meuse.grid["zinc.rfsi"]), col=R_pal[["rainbow_75"]][4:20],
     main="Predictions RFSI", axes=FALSE, box=FALSE)
points(meuse, pch="+", cex=.8)
```

```
par(op)
#dev.off()
```

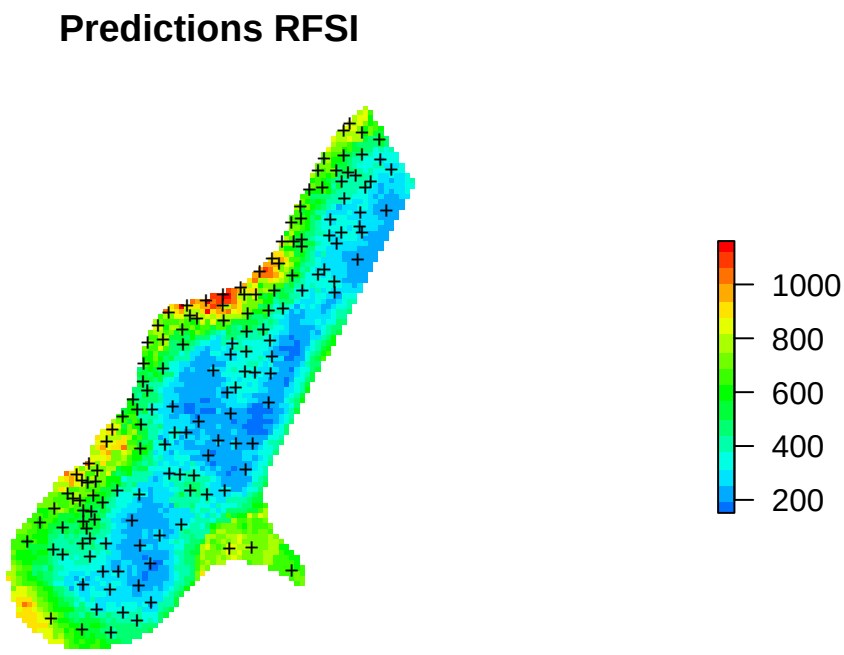


Fig. 2.3 Values of first neighbor for meuse dataset.

In summary, based on the Figs. 2.1 and 2.3, we can conclude that predictions produced using nearest neighbors (Fig. 2.3) show quite different patterns than predictions based on buffer distances (Fig. 2.1). The method by Sekulić et al (2020) (**Random Forest Spatial Interpolation**) RFSI is probably more interesting for general applications as it could be also added to spatiotemporal data problems. It also reflects closely idea of using spatial autocorrelation of values as used in kriging since both values of neighbors and distances to neighbors are used as covariates. On the other hand, RFSI seem to produce predictions that contain also short range variability (more noisy) and as such predictions might appear to look more like geostatistical simulations.

2.3 Interpolation of numeric values using spatial regression

We load the packages that will be used in this tutorial:

```
library(landmap)
library(rgdal)
library(geoR)
#> ------------------------------------------------------------
#>  Analysis of Geostatistical Data
#>  For an Introduction to geoR go to http://www.leg.ufpr.br/geoR
#>  geoR version 1.8-1 (built on 2020-02-08) is now loaded
#> ------------------------------------------------------------
library(plotKML)
library(raster)
library(glmnet)
library(xgboost)
library(kernlab)
#>
#> Attaching package: 'kernlab'
#> The following objects are masked from 'package:raster':
#>
#>     buffer, rotated
library(deepnet)
library(forestError)
library(mlr)
```

For testing we use meuse data set. We can fit a 2D model to interpolate zinc concentration based on sampling points, distance to the river and flooding frequency maps by using:

```
demo(meuse, echo=FALSE)
m <- train.spLearner(meuse["zinc"], covariates=meuse.grid[,c("dist","ffreq")],
                     lambda = 1, parallel=FALSE)
#> # weights:  103
#> initial  value 51302358.935975
#> final  value 19491244.345324
#> converged
#> # weights:  103
#> initial  value 45220763.375836
#> final  value 16169762.129496
#> converged
#> # weights:  103
#> initial  value 50723533.722266
#> final  value 18557431.400000
#> converged
#> # weights:  103
#> initial  value 52179393.837805
#> final  value 19673925.525180
#> converged
#> # weights:  103
#> initial  value 48966129.905012
```

```
#> final  value 19237393.850000
#> converged
#> # weights:  103
#> initial  value 50327217.343373
#> final  value 19238858.992857
#> converged
#> # weights:  103
#> initial  value 50106999.812372
#> final  value 19072846.949640
#> converged
#> # weights:  103
#> initial  value 48513819.381025
#> final  value 18339886.345324
#> converged
#> # weights:  103
#> initial  value 48861791.596477
#> final  value 18454493.171429
#> converged
#> # weights:  103
#> initial  value 48476787.002316
#> final  value 18430903.600000
#> converged
#> # weights:  103
#> initial  value 54933965.304156
#> final  value 20750447.509677
#> converged
```

This runs number of steps including derivation of geographical distances (Møller et al, 2020), derivation of principal components (to make sure all features are numeric and complete), fitting of variogram using the **geoR** package (Diggle and Ribeiro Jr, 2007), spatial overlay, training of individual learners and training of the super learner. In principle, the only parameter we need to set manually in the `train.spLearner` is the `lambda = 1` which is required to estimate variogram: in this case the target variable is log-normally distributed, and hence the geoR package needs the transformation parameter set at `lambda = 1`.

Note that the default meta-learner in `train.spLearner` is a linear model from five independently fitted learners `c("regr.ranger", "regr.xgboost", "regr.ksvm", "regr.nnet", "regr.cvglmnet")`. We can check the success of training based on the 5-fold spatial Cross-Validation using:

```
summary(m@spModel$learner.model$super.model$learner.model)
#>
#> Call:
#> stats::lm(formula = f, data = d)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
```

```
#> -470.94  -97.78  -15.73   64.59 1049.38
#>
#> Coefficients:
#>               Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   2146.8982   968.9798   2.216 0.028234 *
#> regr.ranger      0.6822     0.1912   3.569 0.000483 ***
#> regr.xgboost     0.5249     0.4071   1.289 0.199244
#> regr.nnet       -4.5017     2.0482  -2.198 0.029499 *
#> regr.ksvm        0.4241     0.2148   1.975 0.050145 .
#> regr.cvglmnet   -0.2757     0.1648  -1.673 0.096441 .
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 199.9 on 149 degrees of freedom
#> Multiple R-squared:  0.7132, Adjusted R-squared:  0.7036
#> F-statistic:  74.1 on 5 and 149 DF,  p-value: < 2.2e-16
```

Which shows that the model explains about 65% of variability in target variable and that `regr.ranger` learner (Wright and Ziegler, 2017) is the strongest learner. Average mapping error RMSE = 213, hence the models is somewhat more accurate than if we only used buffer distances.

To predict values at all grids we use:

```
meuse.y <- predict(m)
#> Predicting values using 'getStackedBaseLearnerPredictions'...TRUE
#> Deriving model errors using forestError package...TRUE
```

Note that, by default, we will predict two outputs:

- Mean prediction: i.e. the best unbiased prediction of response;
- Prediction errors: usually predicted as lower and upper 67% quantiles (1 std.) based on the forestError[2] (Lu and Hardin, 2021);

If not otherwise specified, derivation of the prediction error (**Root Mean Square Prediction Error**), bias and lower and upper prediction intervals is implemented by default via the forestError[3] algorithm. The method is explained in detail in Lu and Hardin (2021).

We could also produce the prediction intervals by using the **quantreg** Random Forest algorithm (Meinshausen, 2006) as implemented in the ranger package, or as a standard deviation of the bootstraped models, although using the method by Lu and Hardin (2021) is recommended.

To determine the prediction errors without drastically increasing computing time, we basically fit an independent random forest model using the five base-learners with setting `quantreg = TRUE`:

[2] https://cran.r-project.org/package=forestError

[3] https://cran.r-project.org/package=forestError

```
zinc ~ regr.ranger + regr.xgboost + regr.nnet + regr.ksvm + regr.cvglmnet
```

The prediction error methods are non-parameteric and users can choose any probability in the output via the `quantiles` argument. For example, the default `quantiles` are set to produce prediction intervals for the .682 range, which is the 1-standard-deviation range in the case of a Gaussian distribution. Deriving prediction errors, however, can be come computational for large number of features and trees in the random forest, so have in mind that EML comes with exponentially increased computing time.

We can plot the predictions and prediction errors next to each other by using:

```
par(mfrow=c(1,2), oma=c(0,0,0,1), mar=c(0,0,4,3))
plot(raster(meuse.y$pred["response"]), col=R_pal[["rainbow_75"]][4:20],
     main="Predictions spLearner", axes=FALSE, box=FALSE)
points(meuse, pch="+", cex=.8)
plot(raster(meuse.y$pred["model.error"]), col=rev(bpy.colors()),
     main="Prediction errors", axes=FALSE, box=FALSE)
points(meuse, pch="+", cex=.8)
```

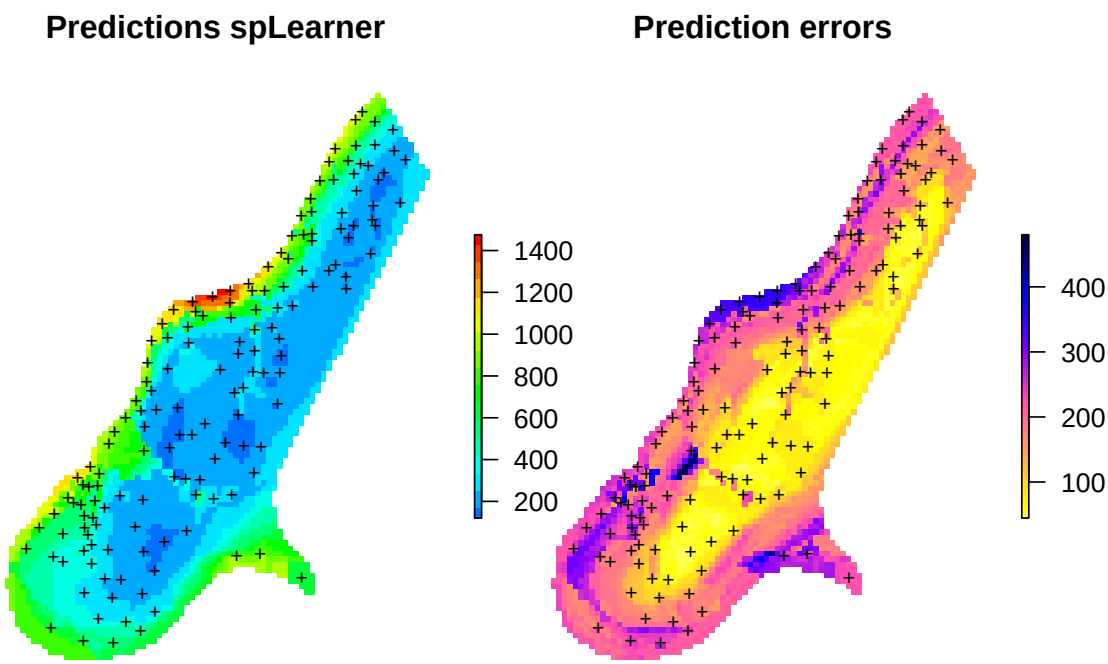


Fig. 2.4 Predicted zinc content based on meuse data set.

This shows that the prediction errors (right plot) are the highest:

- where the model is getting further away from the training points (spatial extrapolation),
- where individual points with high values can not be explained by covariates,
- where measured values of the response variable are in general high,

We can also plot the lower and upper prediction intervals for the .682 probability range using:

```
pts = list("sp.points", meuse, pch = "+", col="black")
spplot(meuse.y$pred[,c("q.lwr","q.upr")], col.regions=R_pal[["rainbow_75"]][4:20],
       sp.layout = list(pts),
       main="Prediction intervals (alpha = 0.318)")
```

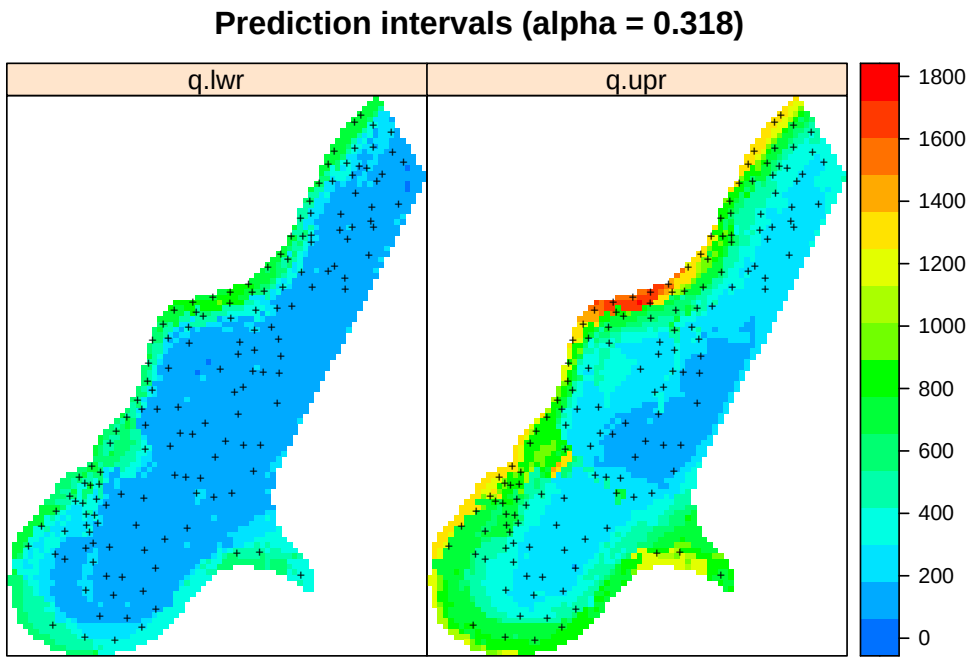


Fig. 2.5 Lower (q.lwr) and upper (q.upr) prediction intervals for zinc content based on meuse data set.

2.4 Model fine-tuning and feature selection

The function `tune.spLearner` can be used to further optimize spLearner object by:

- fine-tuning model parameters, especially the ranger `mtry` and XGBoost parameters,
- reduce number of features by running feature selection via the `mlr::makeFeatSelWrapper` function,

The package landmap currently requires that two base learners used include `regr.ranger` and `regr.xgboost`, and that there are at least 3 base learners in total. The model from above can be optimized using:

```
m0 <- tune.spLearner(m, xg.skip=TRUE, parallel=FALSE)
```

which reports RMSE for different `mtry` and reports which features have been left and which removed. Note that we turn off the fine-tuning of XGboost using `xg.skip = TRUE` as it takes at the order of magnitude more time. In summary, in this specific case, the fine-tuned model is not much more accurate, but it comes with the less features:

```
str(m0@spModel$features)
```

```
 chr [1:11] "PC2" "PC3" "PC4" "rX_0" "rY_0" "rY_0.2" "rX_0.5" "rY_1" "rY_1.4" "rY_2.9" "rY_3.1"
```

```
summary(m0@spModel$learner.model$super.model$learner.model)
```

```
Residuals:
    Min      1Q  Median      3Q     Max 
-404.09 -139.03  -42.05   64.69 1336.47 

Coefficients:
               Estimate Std. Error t value Pr(>|t|)   
(Intercept)  2091.87119  661.70995   3.161  0.00190 **
regr.ranger     0.14278    0.24177   0.591  0.55570   
regr.xgboost    0.92283    0.53131   1.737  0.08448 . 
regr.nnet      -4.34961    1.38703  -3.136  0.00206 **
regr.ksvm       0.66590    0.25027   2.661  0.00865 **
regr.cvglmnet  -0.08703    0.13808  -0.630  0.52944   
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 245.2 on 149 degrees of freedom
Multiple R-squared:  0.5683,    Adjusted R-squared:  0.5538 
F-statistic: 39.23 on 5 and 149 DF,  p-value: < 2.2e-16
```

Note that fine-tuning and feature selection can be quite computational and it is highly recommended to start with smaller subsets of data and then measure processing time. Note that the function `mlr::makeFeatSelWrapper` can result in errors if the covariates have a low variance or follow

a zero-inflated distribution. Reducing the number of features via feature selection and fine-tuning of the Random Forest `mtry` and XGboost parameters, however, can result in significantly higher prediction speed and can also help improve accuracy.

2.5 Estimation of prediction intervals

We can also print the lower and upper prediction interval[4] for every location using e.g.:

```
sp::over(meuse[1,], meuse.y$pred)
#>   response model.error model.bias    q.lwr    q.upr
#> 1 999.8759    214.1839   27.39722 749.5775 1289.188
```

where `q.lwr` is the lower and `q.upr` is the 68% probability upper quantile value. This shows that the 68% probability interval for the location `x=181072, y=333611` is about 734–1241 which means that the prediction error (±1 s.d.), at that location, is about 250. Compare with the actual value sampled at that location:

```
meuse@data[1,"zinc"]
#> [1] 1022
```

The average prediction error for the whole area is:

```
summary(meuse.y$pred$model.error)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>   44.63   77.04  167.69  159.58  215.06  480.11
```

which is somewhat lower than the RMSE derived by cross-validation, but this is also because most of the predicted values are in fact low (skewed distribution), and EML seems not have many problems predicting low values.

Note also, from the example above, if we refit a model using exactly the same settings we might get somewhat different maps and different values. This is to be expected as the number of training points and covariates is low, the stacking is done by using (random) 5-fold Cross-validation, and hence results will always be slightly different. The resulting models and maps, however, should not be significantly different as this would indicate that the Ensemble ML is *unstable*. In the case of larger datasets (1000 points), differences between predictions should become less and less visible.

[4] http://www.sthda.com/english/articles/40-regression-analysis/166-predict-in-r-model-predictions-and-confidence-intervals/

2.6 Predictions using log-transformed target variable

If the purpose of spatial prediction to make a more accurate predictions of low(er) values of the response, then we can train a model with the transformed variable:

```
meuse$log.zinc = log1p(meuse$zinc)
m2 <- train.spLearner(meuse["log.zinc"], covariates=meuse.grid[,c("dist","ffreq")], parallel=FALSE)
#> # weights:  103
#> initial  value 4998.882579
#> final  value 73.222851
#> converged
#> # weights:  103
#> initial  value 5683.969732
#> final  value 68.561330
#> converged
#> # weights:  103
#> initial  value 4834.044628
#> final  value 68.897172
#> converged
#> # weights:  103
#> initial  value 5706.570659
#> final  value 72.649748
#> converged
#> # weights:  103
#> initial  value 2349.857144
#> final  value 73.103072
#> converged
#> # weights:  103
#> initial  value 6184.275769
#> final  value 74.169414
#> converged
#> # weights:  103
#> initial  value 4451.463365
#> final  value 69.440176
#> converged
#> # weights:  103
#> initial  value 4852.414287
#> final  value 72.774801
#> converged
#> # weights:  103
#> initial  value 5902.359343
#> final  value 72.079999
#> converged
#> # weights:  103
#> initial  value 6689.386512
#> final  value 72.803948
```

```
#> converged
#> # weights:  103
#> initial  value 5607.506170
#> final  value 79.790191
#> converged
```

The summary model will usually have a somewhat higher R-square, but the best learners should stay about the same:

```
summary(m2@spModel$learner.model$super.model$learner.model)
#>
#> Call:
#> stats::lm(formula = f, data = d)
#>
#> Residuals:
#>      Min       1Q   Median       3Q      Max
#> -1.00361 -0.18873 -0.05022  0.14092  1.41474
#>
#> Coefficients:
#>               Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   25.75549   10.10511   2.549 0.011822 *
#> regr.ranger    0.74984    0.19969   3.755 0.000248 ***
#> regr.xgboost   0.31617    0.33255   0.951 0.343264
#> regr.nnet     -4.44329    1.70769  -2.602 0.010206 *
#> regr.ksvm      0.28476    0.19541   1.457 0.147146
#> regr.cvglmnet -0.07384    0.15905  -0.464 0.643137
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.3574 on 149 degrees of freedom
#> Multiple R-squared:  0.7615, Adjusted R-squared:  0.7535
#> F-statistic: 95.14 on 5 and 149 DF,  p-value: < 2.2e-16
```

We can next predict and then back-transform the values:

```
meuse.y2 <- predict(m2)
#> Predicting values using 'getStackedBaseLearnerPredictions'...TRUE
#> Deriving model errors using forestError package...TRUE
## back-transform:
meuse.y2$pred$response.t = expm1(meuse.y2$pred$response)
```

The predictions (Figs. 2.4 and 2.6) show similar patterns but the prediction error maps are quite different in this case. Nevertheless, the problem areas seem to match in both maps (see Figs. 2.4 and 2.6 right part). If we compare distributions of two predictions we can also see that the predictions do not differ much:

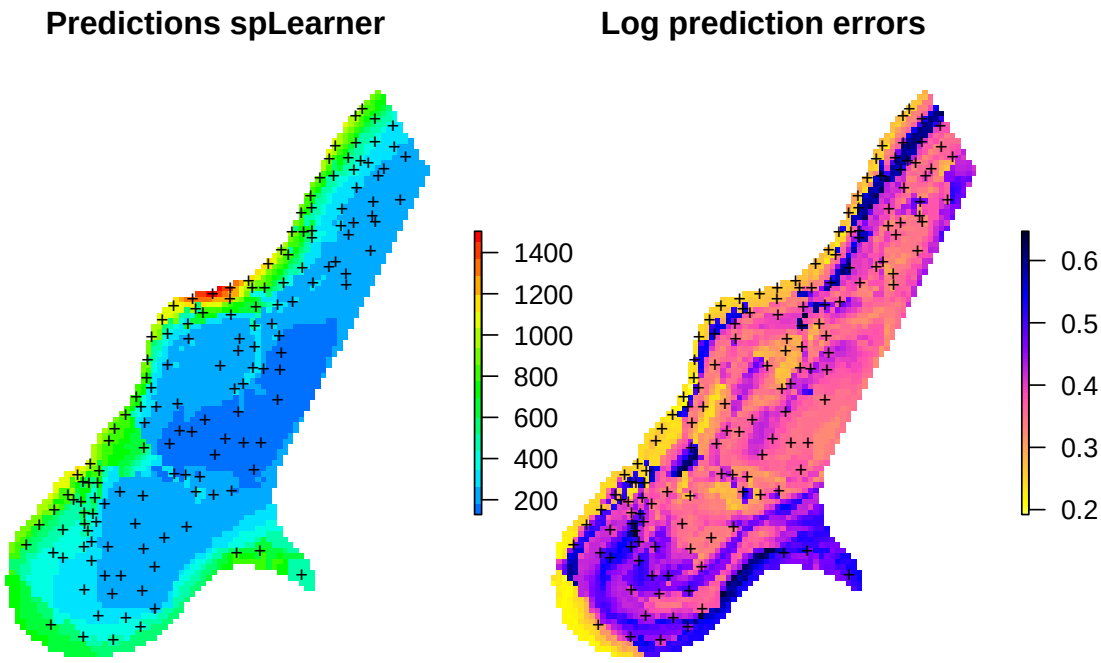


Fig. 2.6 Predicted zinc content based on meuse data set after log-transformation.

```
library(ggridges)
library(viridis)
#> Loading required package: viridisLite
library(ggplot2)
#>
#> Attaching package: 'ggplot2'
#> The following object is masked from 'package:kernlab':
#>
#>     alpha
zinc.df = data.frame(zinc=c(sp::over(meuse, meuse.y$pred["response"])[,1],
                            sp::over(meuse, meuse.y2$pred["response.t"])[,1],
                            meuse$zinc
))
zinc.df$type = as.vector(sapply(c("predicted", "log.predicted", "observed"), function(i){rep(i, nrow(meuse))}))
ggplot(zinc.df, aes(x = zinc, y = type, fill = ..x..)) +
  geom_density_ridges_gradient(scale = 0.95, rel_min_height = 0.01, gradient_lwd = 1.) +
  scale_x_continuous(expand = c(0.01, 0)) +
  ## scale_x_continuous(trans='log2') +
  scale_y_discrete(expand = c(0.01, 0.01)) +
  scale_fill_viridis(name = "Zinc", option = "C") +
  labs(title = "Distributions comparison") +
```

```
  theme_ridges(font_size = 13, grid = TRUE) + theme(axis.title.y = element_blank())
#> Picking joint bandwidth of 110
```

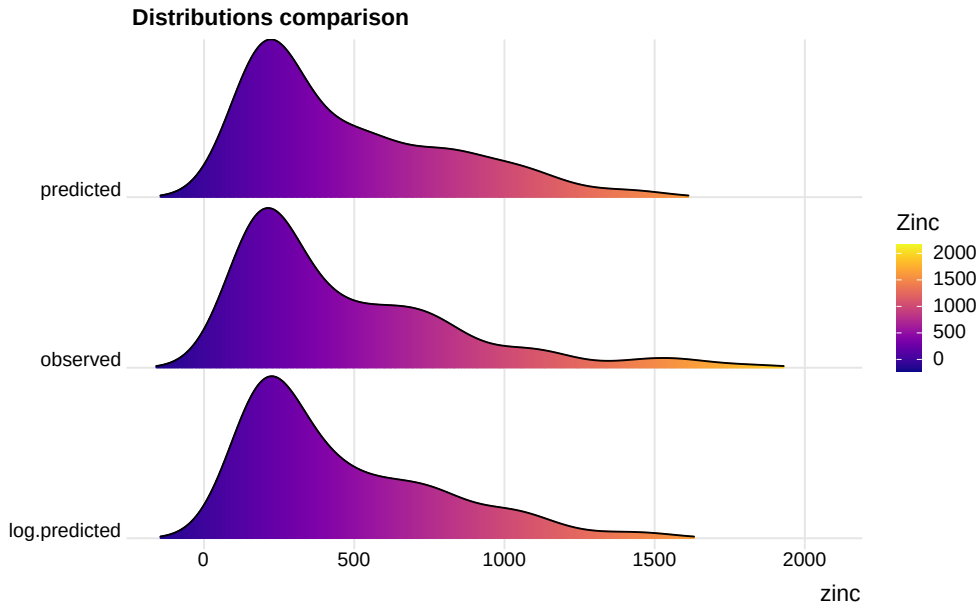


Fig. 2.7 Difference in distributions observed and predicted.

The observed very high values are somewhat smoothed out but the median value is about the same, hence we can conclude that the two EML models predict the target variable without a bias. To estimate the prediction intervals using the log-transformed variable we can use:

```
x = sp::over(meuse[1,], meuse.y2$pred)
expm1(x$q.lwr); expm1(x$q.upr)
#> [1] 837.8133
#> [1] 1638.359
```

Note that the log-transformation is not needed for a non-linear learner such ranger and/or Xgboost, but it is often a good idea if the focus of prediction is to get a better accuracy for lower values (Hengl et al, 2021). For example, if the objective of spatial interpolation is to map soil nutrient deficiencies, then log-transformation is a good idea as it will produce slightly better accuracy for lower values.

Another advantage of using log-transformation for log-normal variables is that the prediction intervals would most likely be symmetric, so that derivation of prediction error (± 1 s.d.) can be derived by:

```
pe = (q.upr - q.lwr)/2
```

2.7 Spatial prediction of soil types (factor-variable)

Ensemble Machine Learning can also be used to interpolate factor type variables e.g. soil types. This is an example with the Ebergotzen dataset available from the package plotKML (Hengl et al, 2015):

```
library(plotKML)
data(eberg_grid)
gridded(eberg_grid) <- ~x+y
proj4string(eberg_grid) <- CRS("+init=epsg:31467")
data(eberg)
coordinates(eberg) <- ~X+Y
proj4string(eberg) <- CRS("+init=epsg:31467")
summary(eberg$TAXGRSC)
#>     Auenboden     Braunerde          Gley         HMoor    Kolluvisol
#>            71           790            86             1           186
#>          Moor Parabraunerde  Pararendzina       Pelosol    Pseudogley
#>             1           704           215           252           487
#>        Ranker       Regosol      Rendzina          NA's
#>            20           376            23           458
```

In this case the target variable is TAXGRSC soil types based on the German soil classification system. This changes the modeling problem from regression to classification. We recommend using the following learners here:

```
sl.c <- c("classif.ranger", "classif.xgboost", "classif.nnTrain")
```

The model training and prediction however looks the same as for the regression:

```
X <- eberg_grid[c("PRMGEO6","DEMSRT6","TWISRT6","TIRAST6")]
if(!exists("mF")){
  mF <- train.spLearner(eberg["TAXGRSC"], covariates=X, parallel=FALSE)
}
#> Converting PRMGEO6 to indicators...
#> Converting covariates to principal components...
#> Deriving oblique coordinates...TRUE
#> Subsetting observations to 79% complete cases...TRUE
#> Skipping variogram modeling...TRUE
#> Estimating block size ID for spatial Cross Validation...TRUE
#> Using learners: classif.ranger, classif.xgboost, classif.nnTrain...TRUE
#> Fitting a spatial learner using 'mlr::makeClassifTask'...TRUE
```

To generate predictions we use:

```
if(!exists("TAXGRSC")){
  TAXGRSC <- predict(mF)
}
#> Predicting values using 'getStackedBaseLearnerPredictions'...TRUE
#> Deriving model errors using sd of sign. learners...TRUE
```

2.8 Classification accuracy

By default landmap package will predict both hard classes and probabilities per class. We can check the average accuracy of classification by using:

```
newdata = mF@vgmModel$observations@data
sel.e = complete.cases(newdata[,mF@spModel$features])
newdata = newdata[sel.e, mF@spModel$features]
pred = predict(mF@spModel, newdata=newdata)
pred$data$truth = mF@vgmModel$observations@data[sel.e, "TAXGRSC"]
print(calculateConfusionMatrix(pred))
#>                predicted
#> true            Auenboden Braunerde Gley Kolluvisol Parabraunerde Pararendzina
#>   Auenboden             34         6    0          0              5             0
#>   Braunerde              0       623    0          1             17             7
#>   Gley                   4         9   38          3              9             0
#>   Kolluvisol             0        13    1         99             17             1
#>   Parabraunerde          0        34    0          3            460             0
#>   Pararendzina           0        19    0          0              3           147
#>   Pelosol                0        11    0          1              1             2
#>   Pseudogley             0        54    2          4             24             3
#>   Ranker                 0        10    0          0              6             0
#>   Regosol                0        66    0          0             10             1
#>   Rendzina               0         2    0          0              0             0
#>   -err.-                 4       224    3         12             92            14
#>                predicted
#> true            Pelosol Pseudogley Ranker Regosol Rendzina -err.-
#>   Auenboden           0          3      0       0        0     14
#>   Braunerde           9          5      0       6        1     46
#>   Gley                0          5      0       0        0     30
#>   Kolluvisol          1          3      0       3        0     39
#>   Parabraunerde       2         10      0       4        0     53
#>   Pararendzina        6          1      0       0        0     29
#>   Pelosol           157          4      0       1        0     20
#>   Pseudogley          4        307      0      13        0    104
#>   Ranker              1          0      0       0        0     17
#>   Regosol             4         12      0     220        0     93
```

```
#>   Rendzina            0          0      0       0       20      2
#>   -err.-             27         43      0      27        1    447
```

which shows that about 25% of classes are miss-classified and the classification confusion is especially high for the `Braunerde` class. Note the result above is based only on the internal training. Normally one should repeat the process several times using 5-fold or similar (i.e. fit EML, predict errors using resampled values only, then repeat).

Predicted probabilities, however, are more interesting because they also show where EML possibly has problems and which are the transition zones between multiple classes:

```
plot(stack(TAXGRSC$pred[grep("prob.", names(TAXGRSC$pred))]),
     col=SAGA_pal[["SG_COLORS_YELLOW_RED"]], zlim=c(0,1))
```

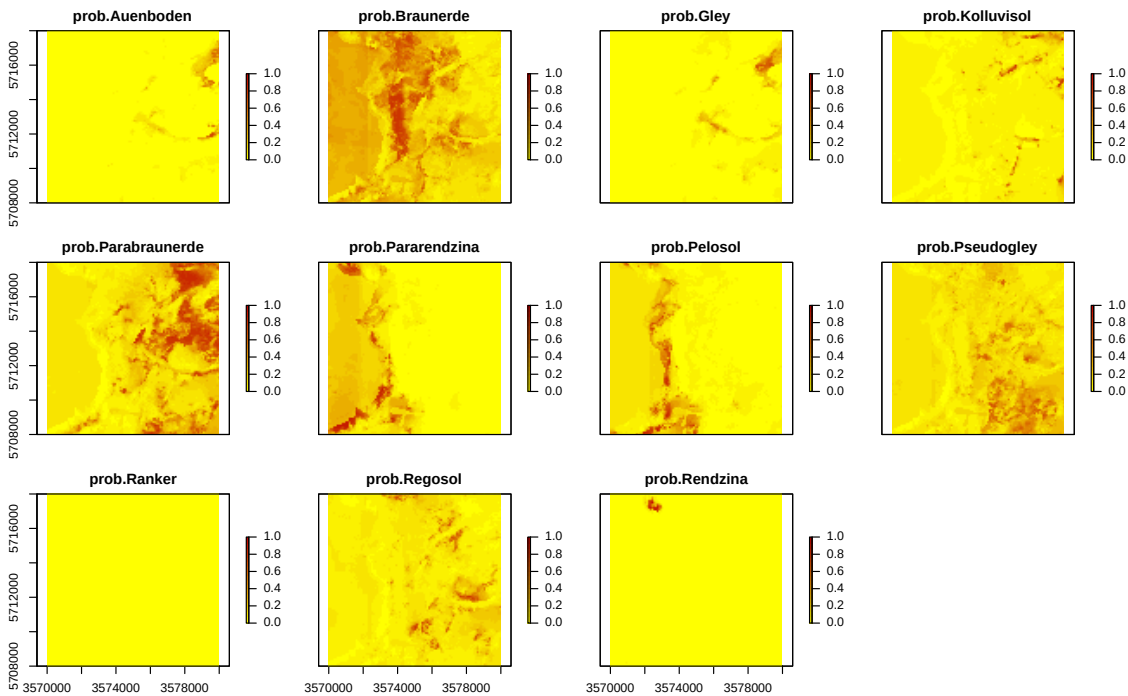


Fig. 2.8 Predicted soil types based on EML.

The maps show that also in this case geographical distances play a role, but overall, the features (DTM derivatives and parnt material) seem to be most important.

In addition to map of probabilities per class, we have also derived errors per probability, which in this case can be computed as the standard deviation between probabilities produced by individual learners (note: for classification problems techniques such as quantreg random forest currently do not exist):

```
plot(stack(TAXGRSC$pred[grep("error.", names(TAXGRSC$pred))]),
     col=SAGA_pal[["SG_COLORS_YELLOW_BLUE"]], zlim=c(0,0.45))
```

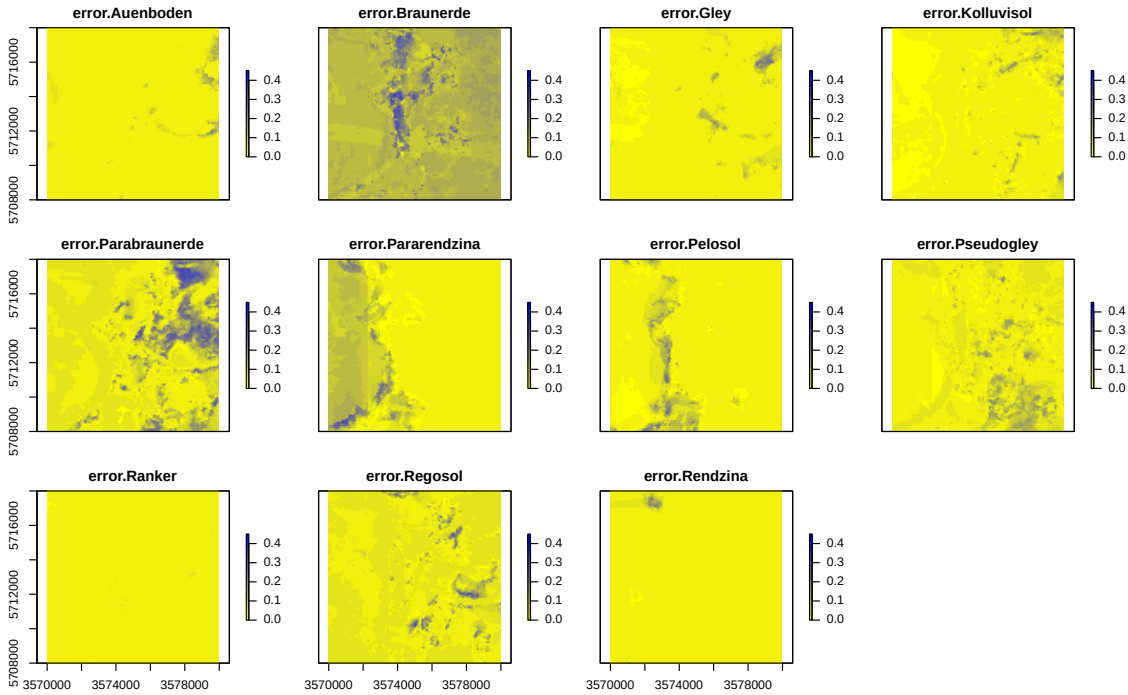


Fig. 2.9 Predicted errors per soil types based on s.d. between individual learners.

In probability space, instead of using RMSE or similar measures, it is often recommended to use the measures such as the log-loss[5] which correctly quantifies the difference between the observed and predicted probability. As a rule of thumb, log-loss values above 0.35 indicate poor accuracy of predictions, but the threshold number for critically low log-loss also depends on the number of classes. In the plot above we can note that, in general, the average error in maps is relatively low e.g. about 0.07:

```
summary(TAXGRSC$pred$error.Parabraunerde)
#>     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
#> 0.001558 0.029074 0.041220 0.066121 0.064179 0.339019
```

but there are still many pixels where confusion between classes and prediction errors are high. Recommended strategy to improve this map is to generate a sampling plan using the average prediction error[6] and/or Confusion Index map, then collect new observations & measurements and refit the prediction models.

[5] https://www.rdocumentation.org/packages/MLmetrics/versions/1.1.1/topics/LogLoss

[6] https://opengeohub.github.io/spatial-sampling-ml/

Chapter 3

Spatial interpolation in 3D using Ensemble ML

You are reading the work-in-progress Spatial and spatiotemporal interpolation using Ensemble 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-prediction-eml/.

3.1 Mapping concentrations of geochemical elements

Ensemble ML can also be used for mapping soil variables in 3D. Consider for example the Geochemical and minerological data set for USA48 (Smith et al, 2014). This is a public data set produced and maintained by the USA Geological Survey[1] and contains laboratory measurements of chemical elements and minerals in soil at 4,857 sites (for three depths 0 to 5 cm, A horizon and C horizon; Fig. 3.1).

We have previously imported and overlaid the sampling points vs large stack of covariate layers using e.g.:

```
## Training data ----
ngs.m <- read.csv("./training_data/ds_801_all.csv")
ngs.m$olc_id = olctools::encode_olc(ngs.m$latitude, ngs.m$longitude, 11)
## Spatial overlay ----
sel.pnts = !duplicated(ngs.m$olc_id)
summary(sel.pnts)
ngs.m.pnts = ngs.m[which(sel.pnts), c("olc_id", "longitude", "latitude")]
coordinates(ngs.m.pnts) <- ~ longitude + latitude
proj4string(ngs.m.pnts) <- "+init=epsg:4326"
ngs.xy = spTransform(ngs.m.pnts, crs(mask))
tif.lst = list.files("./1km", glob2rx("*.tif$"), full.names = TRUE)
## 262 layers
ov.tmp = parallel::mclapply(1:length(tif.lst), function(j){ terra::extract(terra::rast(tif.lst[j]), terra::vect(ngs
```

[1] https://mrdata.usgs.gov/ngdb/soil/

```
ov.tmp = dplyr::bind_cols(lapply(ov.tmp, function(i){i[,2]}))
names(ov.tmp) = tools::file_path_sans_ext(basename(tif.lst))
ov.tmp$olc_id = ngs.xy$olc_id
reg.matrix = plyr::join(ngs.m, ov.tmp)
saveRDS.gz(reg.matrix, "./input/ds801_geochem1km.rds")
```

We can load the regression matrix which contains all target variables and covariates:

```
ds801 = readRDS("./input/ds801_geochem1km.rds")
dim(ds801)
#> [1] 14275   407
```

which has a total of 4818 unique locations:

```
str(levels(as.factor(ds801$olc_id)))
#>  chr [1:4818] "75WXFVQG+WH6" "75WXJHVG+62X" "75WXJVM9+995" "75WXRG2G+5PV" ...
```

The individual records can be browsed directly via https://mrdata.usgs.gov/ngdb/soil/, for example a single record includes:

NGDB soil sample C184910 (agricultural/cultivated land)

About the sample | Elemental analysis (best measures) | Chemical analyses (all)

Field	Value
Lab ID	C184910
Field ID	SS4
Job ID	MRP03060
Sample submitter	Brown, Craig
Date collected	10/13/00
Date submitted	1/25/2001
State	CT
Country	United States
Latitude (WGS84)	41.89128367
Longitude (WGS84)	-72.52661409
Original datum	NAD27
Original latitude	41.891186
Original longitude	-72.527083
Location precision	1 second
Sampling depth	2"" - 4
Description of location	tobacco field
Sample source	agricultural/cultivated land
Collection method	single/grab
Primary media type	soil

NGDB soil sample C184910 (agricultural/cultivated land)

About the sample | Elemental analysis (best measures) | Chemical analyses (all)

Element	Measured value	Units of measure
Vanadium	53	parts per million by weight
Chromium	50	parts per million by weight
Nickel	14	parts per million by weight
Copper	108	parts per million by weight
Zinc	60	parts per million by weight
Gallium	15	parts per million by weight
Germanium	<2	parts per million by weight
Arsenic	23	parts per million by weight
Selenium	<1	parts per million by weight
Bromine	5	parts per million by weight
Rubidium	54	parts per million by weight
Strontium	110	parts per million by weight
Yttrium	29	parts per million by weight
Zirconium	360	parts per million by weight
Niobium	12	parts per million by weight
Molybdenum	2	parts per million by weight
Silver	<1	parts per million by weight
Cadmium	<1	parts per million by weight
Tin	<2	parts per million by weight
Antimony	<2	parts per million by weight

Fig. 3.1 Example of a geochemical sample with observations and measurements and coordiates of site.

Covariates prepared to help interpolation of the geochemicals include:

- Distance to cities (Nelson et al, 2019);
- MODIS LST[2] (monthly daytime and nighttime);
- MODIS EVI[3] (long-term monthly values);
- Lights at night images[4];
- Snow occurrence probability[5];
- Soil property and class maps for USA48 (Ramcharan et al, 2018);
- Terrain / hydrological indices;

We can focus on predict concentration of lead (Pb). Pb seems to change with depth and this is seems to be consistent for different land cover classes:

```
openair::scatterPlot(ds801[ds801$pb_ppm<140,], x = "hzn_depth", y = "pb_ppm", method = "hexbin", col = "increment",
#> Warning: removing 11 missing rows due to landcover1
```

Because we are aiming at producing predictions of geochemical elements for different depths in soil, we can also use depth of the soil sample as one of the covariates. This makes the prediction system 3D and the model can thus be used to predict at any new 3D location (X, Y, d). To fit a RF model for this data we can use:

```
ds801$log.pb = log1p(ds801$pb_ppm)
pr.vars = c(readRDS("./input/pb.pr.vars.rds"), "hzn_depth")
sel.pb = complete.cases(ds801[,c("log.pb", pr.vars)])
mrf = ranger::ranger(y=ds801$log.pb[sel.pb], x=ds801[sel.pb, pr.vars],
            num.trees = 85, importance = 'impurity')
mrf
#> Ranger result
#>
#> Call:
#>  ranger::ranger(y = ds801$log.pb[sel.pb], x = ds801[sel.pb, pr.vars],      num.trees = 85, importance = "impurit
#>
#> Type:                             Regression
#> Number of trees:                  85
#> Sample size:                      14264
#> Number of independent variables:  188
#> Mtry:                             13
```

[2] https://doi.org/10.5281/zenodo.1420114
[3] https://lpdaac.usgs.gov/products/mod13q1v006/
[4] https://eogdata.mines.edu/products/dmsp/
[5] https://climate.esa.int/en/odp/#/project/snow

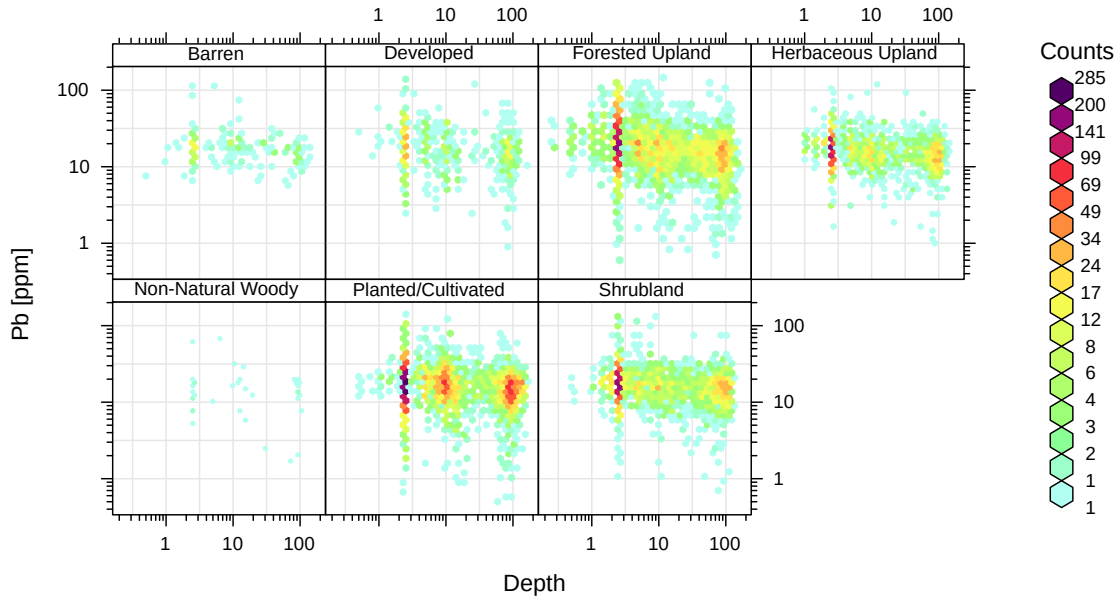


Fig. 3.2 Distribution of Pb as a function of land cover classes.

```
#> Target node size:                 5
#> Variable importance mode:         impurity
#> Splitrule:                        variance
#> OOB prediction error (MSE):       0.09636483
#> R squared (OOB):                  0.670695
```

which results in R-square of about 0.67. Because many training samples have exactly the same coordinates (same site, three depths), we assume that this model is over-fitting i.e. that the out-of-bag accuracy is probably over-optimistic[6]. Instead we can fit an Ensemble model where we block points within 30 by 30-km blocks:

```
if(!exists("eml.pb")){
  lrn.rf = mlr::makeLearner("regr.ranger", num.trees=85, importance="impurity",
                            num.threads = parallel::detectCores())
  lrns.pb <- list(lrn.rf, mlr::makeLearner("regr.xgboost"), mlr::makeLearner("regr.cvglmnet"))
  tsk0.pb <- mlr::makeRegrTask(data = ds801[sel.pb, c("log.pb", pr.vars)],
                               target = "log.pb", blocking = as.factor(ds801$ID[sel.pb]))
  init.pb <- mlr::makeStackedLearner(lrns.pb, method="stack.cv", super.learner="regr.lm",
                                     resampling=mlr::makeResampleDesc(method="CV", blocking.cv=TRUE))
```

[6] https://opengeohub.github.io/spatial-sampling-ml/

```
  parallelMap::parallelStartSocket(parallel::detectCores())
  eml.pb = train(init.pb, tsk0.pb)
  parallelMap::parallelStop()
}
#> [18:37:08] WARNING: amalgamation/../src/objective/regression_obj.cu:170: reg:linear is now deprecated in favor o
```

```
summary(eml.pb$learner.model$super.model$learner.model)
#>
#> Call:
#> stats::lm(formula = f, data = d)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -2.3979 -0.1990 -0.0117  0.1699  6.2948
#>
#> Coefficients:
#>               Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   -0.64745    0.04296 -15.069  < 2e-16 ***
#> regr.ranger    0.85552    0.02152  39.755  < 2e-16 ***
#> regr.xgboost   0.26550    0.05337   4.974 6.62e-07 ***
#> regr.cvglmnet  0.25631    0.02024  12.665  < 2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.4075 on 14260 degrees of freedom
#> Multiple R-squared:  0.4328, Adjusted R-squared:  0.4327
#> F-statistic:  3627 on 3 and 14260 DF,  p-value: < 2.2e-16
```

which shows somewhat lower R-square of 0.44, this time that whole sites have been taken out and hence this seems to be somewhat more realistic estimate of the mapping accuracy (Roberts et al, 2017). The accuracy plot shows that the model has some problems with predicting higher values, but overall matches the observed values:

```
t.pb = quantile(ds801$log.pb, c(0.001, 0.01, 0.999), na.rm=TRUE)
plot_hexbin(varn="log.pb", breaks=c(t.pb[1], seq(t.pb[2], t.pb[3], length=25)),
      meas=eml.pb$learner.model$super.model$learner.model$model$log.pb,
      pred=eml.pb$learner.model$super.model$learner.model$fitted.values,
      main="Pb [EML]")
```

Variables most important for explaining distribution of the target variable (based on the variable importance) seem to be soil depth (`hnz_depth`), soil type maps, annual day time temperature and travel time to cities:

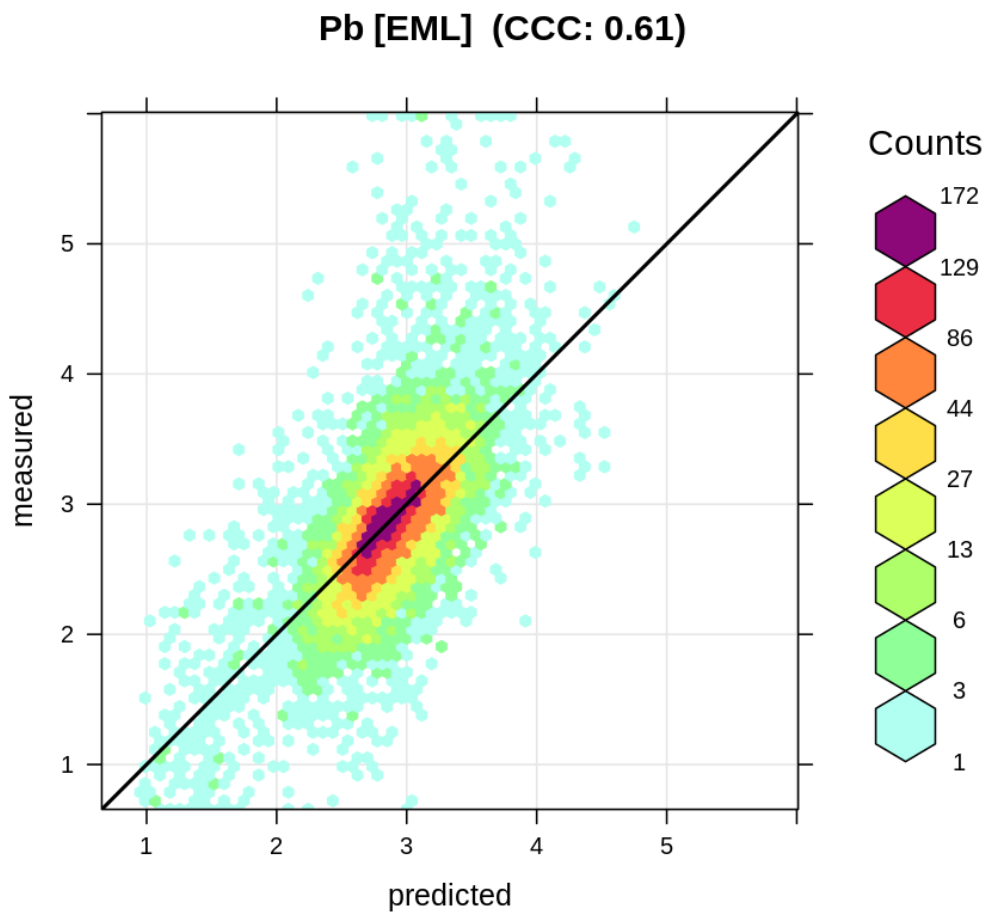


Fig. 3.3 Accuracy plot for Pb concentration in soil fitted using Ensemble ML.

```
library(ggplot2)
#>
#> Attaching package: 'ggplot2'
#> The following object is masked from 'package:latticeExtra':
#>
#>     layer
xl.pb <- as.data.frame(mlr::getFeatureImportance(eml.pb[["learner.model"]][["base.models"]][[1]])$res)
xl.pb$relative_importance = 100*xl.pb$importance/sum(xl.pb$importance)
xl.pb = xl.pb[order(xl.pb$relative_importance, decreasing = T),]
xl.pb$variable = paste0(c(1:nrow(xl.pb)), ". ", xl.pb$variable)
ggplot(data = xl.pb[1:20,], aes(x = reorder(variable, relative_importance), y = relative_importance)) +
  geom_bar(fill = "steelblue",
```

```
          stat = "identity") +
  coord_flip() +
  labs(title = "Variable importance",
       x = NULL,
       y = NULL) +
  theme_bw() + theme(text = element_text(size=15))
```

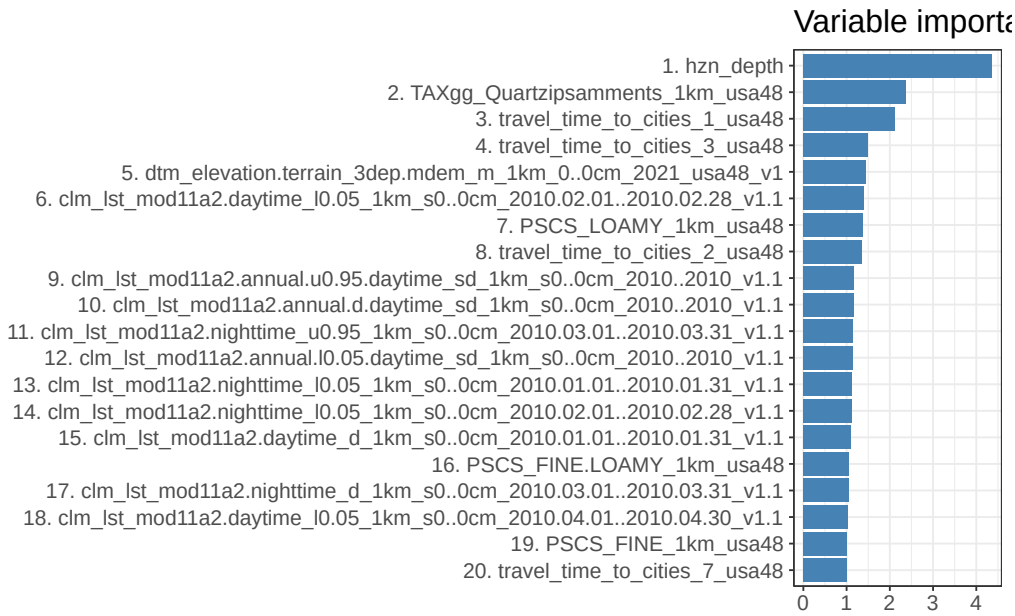


Fig. 3.4 Variable importance for 3D prediction model for Pb concentrations.

If we plot the travel time to cities vs Pb concentrations, we can clearly see that Pb is negatively correlated with travel time to cities (following a log-log linear relationship):

```
openair::scatterPlot(ds801[ds801$pb_ppm<140,], x = "travel_time_to_cities_1_usa48", y = "pb_ppm", method = "hexbin"
#> Warning: removing 11 missing rows due to hzn_depth
```

3.2 Predictions in 3D

To produce predictions we can focus on area around Chicago conglomeration. We can load the covariate layers by using:

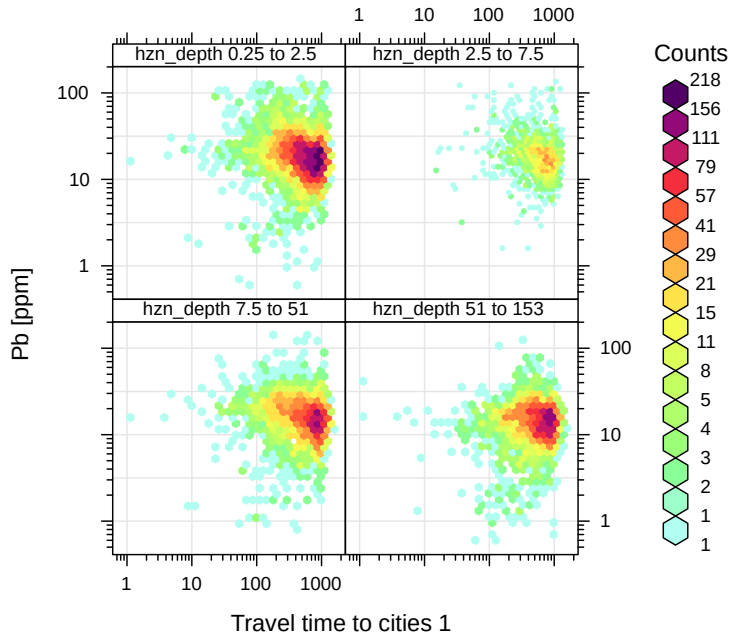


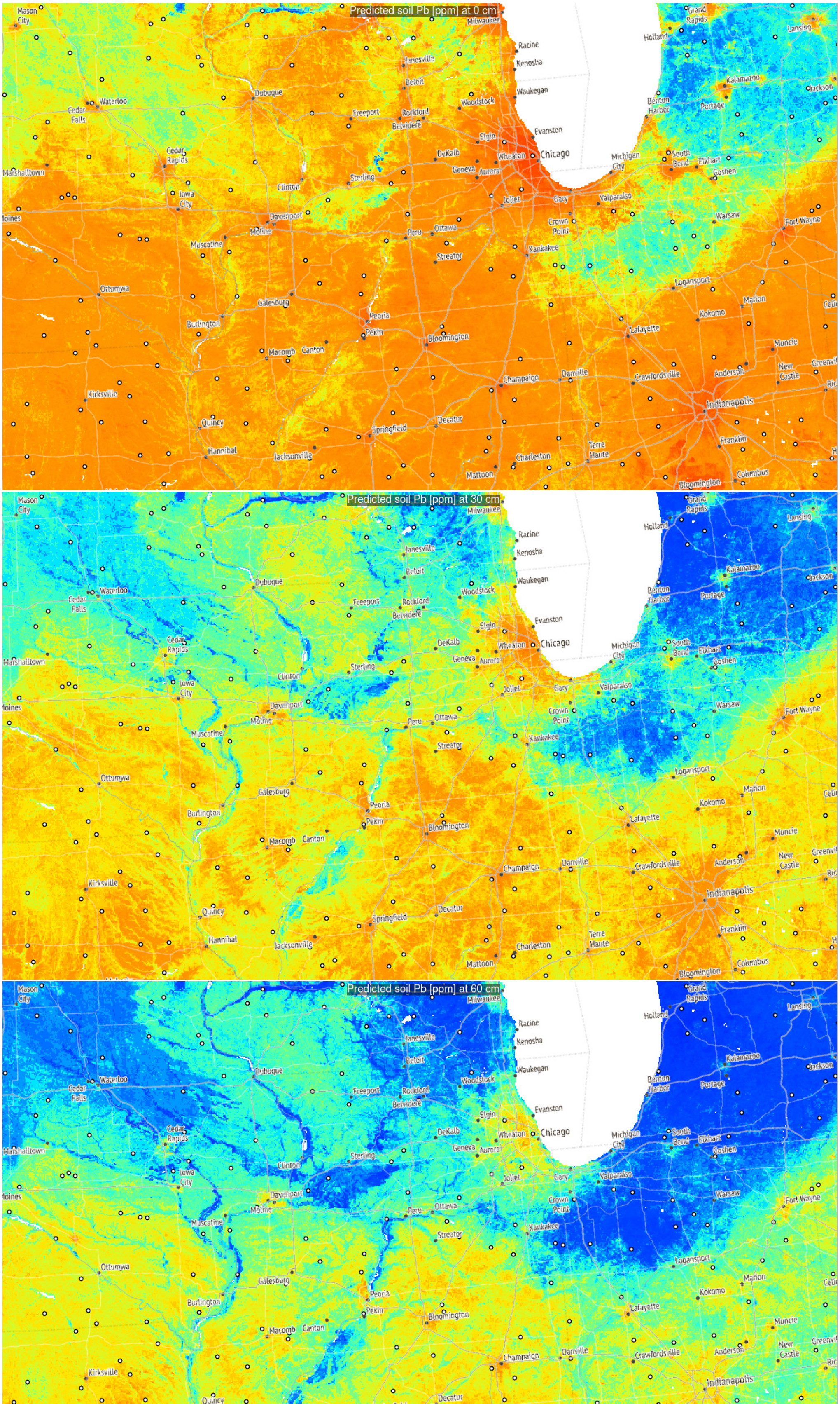
Fig. 3.5 Distribution of Pb as a function of travel time to cities for different depths.

```
g1km = readRDS("./input/chicago_grid1km.rds")
```

This contains all layers we used for training. We can generate predictions by adding a depth column, then write predictions to GeoTIFFs:

```
for(k in c(5, 30, 60)){
  out.tif = paste0("./output/pb_ppm_", k, "cm_1km.tif")
  if(!file.exists(out.tif)){
    g1km$hzn_depth = k
    sel.na = complete.cases(g1km)
    newdata = g1km[sel.na, eml.pb$features]
    pred = predict(eml.pb, newdata=newdata)
    g1km.sp = SpatialPixelsDataFrame(as.matrix(g1km[sel.na,c("x","y")]),
                data=pred$data, proj4string=CRS("EPSG:5070"))
    g1km.sp$pred = expm1(g1km.sp$response)
    rgdal::writeGDAL(g1km.sp["pred"], out.tif, type="Int16", mvFlag=-32768, options=c("COMPRESS=DEFLATE"))
    #gc()
  }
}
```

This finally gives the following pattern:


Predicted soil Pb [ppm] at 0 cm
Predicted soil Pb [ppm] at 30 cm
Predicted soil Pb [ppm] at 60 cm
Chicago
Milwaukee
Indianapolis
Dubuque
Davenport
Peoria
Bloomington
Champaign
Springfield
Kankakee
Lafayette
Kalamazoo
Waterloo
Rockford
Joliet
Gary
Galesburg
Quincy
Decatur
Danville

Based on these results, it can be said that in general:

1. Distribution of Pb across USA seem to be controlled by a mixture of factors including climatic factors, soil-terrain units and anthropogenic factors (travel distance to cities);

2. Overall big urban areas show significantly higher concentrations for some heavy metals and this relationship if log-log linear;

3. Soil depth for some geochemical elements comes as the overall most important covariate hence mapping soil variables in 3D is fully justified;

3.3 Advantages and limitations of running 3D predictive mapping

In summary 3D soil mapping is relatively straight forward to implement especially for mapping soil variables from soil profiles (multiple samples per soil layer). Before modeling the target variable with depth, it is a good idea to plot relationship between target variable and depth under different settings (as in Fig. 3.2). 3D soil mapping based on Machine Learning is now increasingly common (Hengl and MacMillan, 2019; Sothe et al, 2022).

A limitation for 3D predictive mapping is the size of data i.e. data volumes increasing proportionally to number of slices we need to predict (in this case three). Also, we show that points with exactly the same coordinates might result in e.g. Random Forest overfitting emphasizing some covariate layers that are possibly less important, hence it is again important to use the blocking parameter that separates training and validation points.

Soil depth is for many soil variables most important explanatory variables, but in the cases it does not correlate with the target variable, there is probably also no need for 3D soil mapping. In the case depth is not significantly correlated, one could simply first aggregate all values to fixed block depth and convert modeling from 3D to 2D.

Chapter 4

Spatiotemporal interpolation using Ensemble ML

You are reading the work-in-progress Spatial and spatiotemporal interpolation using Ensemble 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-prediction-eml/.

4.1 Spatiotemporal interpolation of daily temperatures

In previous examples we have demonstrated effects of over-fitting and how Ensemble ML helps decrease overfitting and extrapolation problems using synthetic data. We can now look at some real-life cases for example the daily temperatures measured for several years for Croatia described in Hengl et al (2012). This data sets consist of two parts: (1) measurements of daily temperatures at meteo stations, (2) list of gridded covariates (Fig. 4.1).

We can load the point data by using:

```
library(rgdal)
hrmeteo = readRDS("input/hrtemp2006_meteo.rds")
str(hrmeteo)
#> List of 2
#>  $ meteo   :'data.frame':    44895 obs. of  5 variables:
#>   ..$ IDSTA : chr [1:44895] "GL001" "GL001" "GL001" "GL001" ...
#>   ..$ DATE  : chr [1:44895] "2006-1-1" "2006-1-2" "2006-1-3" "2006-1-4" ...
#>   ..$ MDTEMP: num [1:44895] 1.6 0.7 1.5 0.3 -0.1 1 0.3 -1.9 -5.4 -3.6 ...
#>   ..$ cday  : num [1:44895] 13148 13149 13150 13151 13152 ...
#>   .. ..- attr(*, "tzone")= chr ""
#>   ..$ x     : num [1:44895] NA NA NA NA NA NA NA NA NA NA ...
#>  $ stations:'data.frame':    152 obs. of  3 variables:
#>   ..$ IDSTA: chr [1:152] "GL001" "GL002" "GL003" "GL004" ...
#>   ..$ X    : num [1:152] 670760 643073 673778 752344 767729 ...
#>   ..$ Y    : num [1:152] 5083464 5086417 5052001 4726567 4717878 ...
```

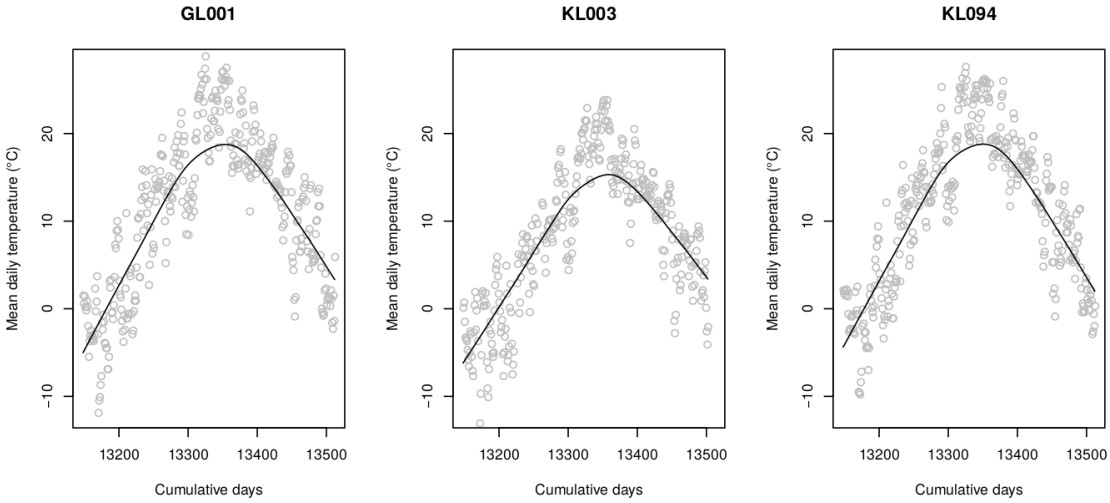


Fig. 4.1 Temporal dynamics of mean-daily temperatures at sample meteorological stations. This shows seasonality effects (smoothed line) and daily oscillations.

```
idsta.pnts = hrmeteo$stations
coordinates(idsta.pnts) = ~ X + Y
```

This is a typical format for spatiotemporal meteorological data with locations of stations in one table, and measurements of daily temperatures (`MDTEMP`) in other. The column `cday` here is the cumulative day since 1970, which allows us to present time on a linear scale i.e. by using a numeric value instead of dates.

The gridded data includes: (a) static covariates (relief data), and (b) dynamic time-series data (MODIS LST). To load the static covariates we use:

```
hrgrid1km = readRDS("input/hrgrid1km.rds")
#plot(hrgrid1km[1])
proj4string(idsta.pnts) = proj4string(hrgrid1km)
str(hrgrid1km@data)
#> 'data.frame':    238630 obs. of  4 variables:
#>  $ HRdem : int  1599 1426 1440 1764 1917 1912 1707 1550 1518 1516 ...
#>  $ HRdsea: num  93 89.6 89.8 93.6 95 ...
#>  $ Lat   : num  46.5 46.5 46.5 46.5 46.5 ...
#>  $ Lon   : num  13.2 13.2 13.2 13.2 13.2 ...
```

The dynamic time-series data is stored in a local folder (`input/LST2006HR`) as individual files that we can list by using:

```
LST.listday <- dir("input/LST2006HR", pattern=glob2rx("LST2006_**_**.LST_Day_1km.tif"), full.names = TRUE)
LST.listnight <- dir("input/LST2006HR", pattern=glob2rx("LST2006_**_**.LST_Night_1km.tif"), full.names = TRUE)
str(LST.listday)
#>  chr [1:46] "input/LST2006HR/LST2006_01_01.LST_Day_1km.tif" ...
```

Here we see there are 46 images for year 2006 with daytime and 46 images for night time estimates of LST. We do not want to load all these rasters to R because we might experience RAM problems. We can first overlay points and see which variables can help with mapping daily temperatures.

For the static covariates we only have to run the overlay once:

```
idsta.ov <- sp::over(idsta.pnts, hrgrid1km)
idsta.ov$IDSTA = idsta.pnts$IDSTA
str(idsta.ov)
#> 'data.frame':    152 obs. of  5 variables:
#>  $ HRdem : int  161 134 202 31 205 563 80 96 116 228 ...
#>  $ HRdsea: num  198.5 181.7 192.9 0 1.5 ...
#>  $ Lat   : num  45.9 45.9 45.6 42.7 42.6 ...
#>  $ Lon   : num  17.2 16.8 17.2 18.1 18.3 ...
#>  $ IDSTA : chr  "GL001" "GL002" "GL003" "GL004" ...
```

For the spatiotemporal data (MODIS LST time-series) we need to run overlay as in a spacetime cube. This means that we need to match points using `x,y,t` coordinates with grids covering the same `x,y,t` fields. To speed up spacetime overlay we use our custom function `extract_st`, which basically builds on top of the `terra` package. First, we need to define begin, end times for each GeoTIFF:

```
library(terra)
source("mlst_functions.R")
hrmeteo$meteo$x = plyr::join(hrmeteo$meteo, hrmeteo$stations, by="IDSTA")$X
hrmeteo$meteo$y = plyr::join(hrmeteo$meteo, hrmeteo$stations, by="IDSTA")$Y
## generate row ID:
hrmeteo$meteo$row.id = 1:nrow(hrmeteo$meteo)
hrmeteo$meteo$Date = as.Date(hrmeteo$meteo$DATE, format = "%Y-%m-%d")
## strip dates from filename:
begin.tif1.lst = as.Date(paste0("2006-", substr(basename(LST.listday), 9, 10),
                                "-", substr(basename(LST.listday), 12, 13)))-4
end.tif1.lst = as.Date(paste0("2006-", substr(basename(LST.listday), 9, 10),
                              "-", substr(basename(LST.listday), 12, 13)))+4
```

now that we know spacetime coordinates for both points and grids, we can run overlay in parallel to speed up computing:

```
mc.cores = parallel::detectCores()
ov.pnts <- parallel::mclapply(1:length(LST.listday), function(i){
  extract_st(tif=LST.listday[i], hrmeteo$meteo, date="Date",
             crs = proj4string(hrgrid1km),
             date.tif.begin=begin.tif1.lst[i],
             date.tif.end=end.tif1.lst[i],
             coords=c("x","y"), variable.name="LST.day") },
  mc.cores=mc.cores)
ov.pnts = ov.pnts[!sapply(ov.pnts, is.null)]
ov.tifs1 = plyr::join_all(ov.pnts, by="row.id", type="full")
str(ov.tifs1)
#> 'data.frame':    44895 obs. of  2 variables:
#>  $ LST.day: num  0 0 0 0 0 0 0 0 0 0 ...
#>  $ row.id : int  1 2 3 4 5 366 367 368 369 370 ...
ov.tifs1$LST.day = ifelse(ov.tifs1$LST.day == 0, NA, ov.tifs1$LST.day)
```

In this case we also exclude the values of `LST.day` are equal to 0 as these are basically missing values in the GeoTIFFs. We repeat the same overlay operation for the night light images:

```
begin.tif2.lst = as.Date(paste0("2006-", substr(basename(LST.listnight), 9, 10),
                                "-", substr(basename(LST.listnight), 12, 13)))-4
end.tif2.lst = as.Date(paste0("2006-", substr(basename(LST.listnight), 9, 10),
                              "-", substr(basename(LST.listnight), 12, 13)))+4
ov.pnts <- parallel::mclapply(1:length(LST.listnight), function(i){
  extract_st(tif=LST.listnight[i], hrmeteo$meteo, date="Date",
             crs = proj4string(hrgrid1km),
             date.tif.begin=begin.tif2.lst[i],
             date.tif.end=end.tif2.lst[i],
             coords=c("x","y"), variable.name="LST.night") },
  mc.cores=mc.cores)
ov.pnts = ov.pnts[!sapply(ov.pnts, is.null)]
ov.tifs2 = plyr::join_all(ov.pnts, by="row.id", type="full")
str(ov.tifs2)
#> 'data.frame':    44895 obs. of  2 variables:
#>  $ LST.night: num  13344 13344 13344 13344 13344 ...
#>  $ row.id   : int  1 2 3 4 5 366 367 368 369 370 ...
ov.tifs2$LST.night = ifelse(ov.tifs2$LST.night == 0, NA, ov.tifs2$LST.night)
```

The result of spacetime overlay is a simple long table matching exactly the meteo-data table. We next bind results of overlay using static and dynamic covariates:

```
hrmeteo.rm = plyr::join_all(list(hrmeteo$meteo, ov.tifs1, ov.tifs2))
#> Joining by: row.id
#> Joining by: row.id
```

```
hrmeteo.rm = plyr::join(hrmeteo.rm, idsta.ov)
#> Joining by: IDSTA
```

we also add the geometric component of temperature based on the sphere formulas:

```
hrmeteo.rm$temp.mean <- temp.from.geom(fi=hrmeteo.rm$Lat,
                    as.numeric(strftime(hrmeteo.rm$Date, format = "%j")),
                    a=37.03043, b=-15.43029, elev=hrmeteo.rm$HRdem, t.grad=0.6)
```

We have now produced a **spatiotemporal regression matrix** that can be used to fit a prediction model for daily temperature. The model is of form:

```
fm.tmp <- MDTEMP ~ temp.mean + LST.day + LST.night + HRdsea
```

We next fit an Ensemble ML using the same process described in the previous sections:

```
library(mlr)
lrn.rf = mlr::makeLearner("regr.ranger", num.trees=150, importance="impurity",
                          num.threads = parallel::detectCores())
lrns.st <- list(lrn.rf, mlr::makeLearner("regr.nnet"), mlr::makeLearner("regr.gamboost"))
sel = complete.cases(hrmeteo.rm[,all.vars(fm.tmp)])
hrmeteo.rm = hrmeteo.rm[sel,]
#summary(sel)
subs <- runif(nrow(hrmeteo.rm))<.2
tsk0.st <- mlr::makeRegrTask(data = hrmeteo.rm[subs,all.vars(fm.tmp)],
                             target = "MDTEMP", blocking = as.factor(hrmeteo.rm$IDSTA[subs]))
tsk0.st
#> Supervised task: hrmeteo.rm[subs, all.vars(fm.tmp)]
#> Type: regr
#> Target: MDTEMP
#> Observations: 7573
#> Features:
#>    numerics     factors     ordered functionals
#>           4           0           0           0
#> Missings: FALSE
#> Has weights: FALSE
#> Has blocking: TRUE
#> Has coordinates: FALSE
```

Train model using a subset of points:

```
init.TMP <- mlr::makeStackedLearner(lrns.st, method="stack.cv", super.learner="regr.lm",
                                    resampling=mlr::makeResampleDesc(method="CV", blocking.cv=TRUE))
parallelMap::parallelStartSocket(parallel::detectCores())
eml.TMP = train(init.TMP, tsk0.st)
#> # weights:  19
#> initial  value 1752597.611462
#> final  value 503633.983959
#> converged
parallelMap::parallelStop()
```

This shows that daily temperatures can be predicted with relatively high R-square, although the residual values are still significant (ranging from -1.8 to 1.8 degrees):

```
summary(eml.TMP$learner.model$super.model$learner.model)
#>
#> Call:
#> stats::lm(formula = f, data = d)
#>
#> Residuals:
#>      Min       1Q   Median       3Q      Max
#> -16.2439  -1.8106   0.0204   1.8004  14.9611
#>
#> Coefficients:
#>               Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   35.06219    8.10904   4.324 1.55e-05 ***
#> regr.ranger    0.70575    0.02770  25.474  < 2e-16 ***
#> regr.nnet     -2.72276    0.62920  -4.327 1.53e-05 ***
#> regr.gamboost  0.29540    0.02799  10.553  < 2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 2.888 on 7569 degrees of freedom
#> Multiple R-squared:  0.8747, Adjusted R-squared:  0.8746
#> F-statistic: 1.761e+04 on 3 and 7569 DF,  p-value: < 2.2e-16
```

The variable importance analysis shows that the most important variable for predicting daily temperatures is, in fact, the night-time MODIS LST:

```
library(ggplot2)
xl <- as.data.frame(mlr::getFeatureImportance(eml.TMP[["learner.model"]][["base.models"]][[1]])$res)
xl$relative_importance = 100*xl$importance/sum(xl$importance)
xl = xl[order(xl$relative_importance, decreasing = TRUE),]
xl$variable = paste0(c(1:length(xl$variable)), ". ", xl$variable)
ggplot(data = xl[1:4,], aes(x = reorder(variable, relative_importance), y = relative_importance)) +
```

```
  geom_bar(fill = "steelblue",
           stat = "identity") +
  coord_flip() +
  labs(title = "Variable importance",
       x = NULL,
       y = NULL) +
  theme_bw() + theme(text = element_text(size=15))
```

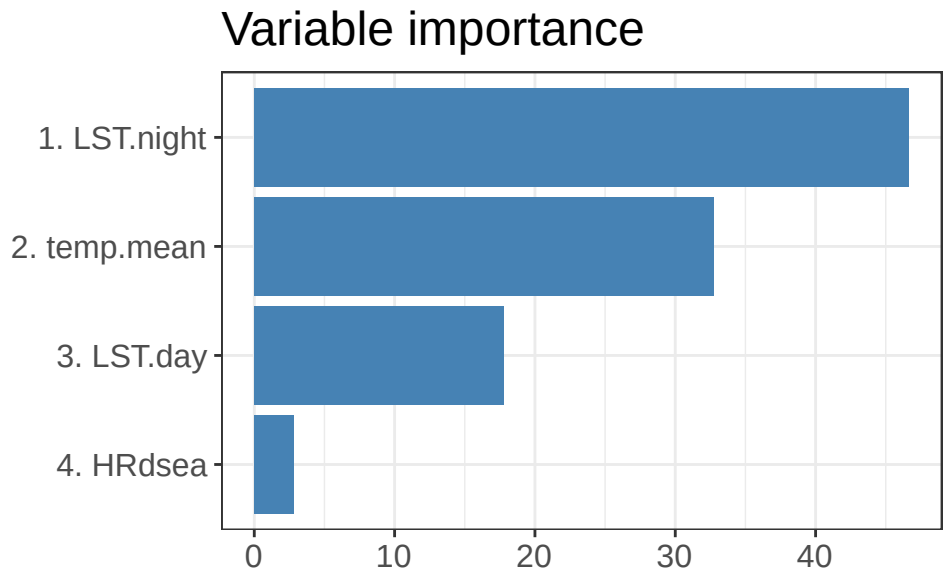


Fig. 4.2 Variable importance for modeling spacetime daily temperatures.

We can use the **fitted spacetime EML model** to generate predictions e.g. for four consecutive days in August. First, we import MODIS LST for month of interest:

```
hrpred1km = hrgrid1km
sel.tifs1 = LST.listday[grep("_08_", LST.listday)]
sel.tifs2 = LST.listnight[grep("_08_", LST.listnight)]
## read to R in parallel
x1 = as.data.frame( parallel::mclapply(sel.tifs1,
      function(i){x <- readGDAL(i)$band1; x <- ifelse(x<1, NA, x); return(x)},
                                       mc.cores = mc.cores))
x2 = as.data.frame( parallel::mclapply(sel.tifs2,
      function(i){x <- readGDAL(i)$band1; x <- ifelse(x<1, NA, x); return(x)},
                                       mc.cores = mc.cores))
names(x1)  <- basename(sel.tifs1); names(x2) <- basename(sel.tifs2)
```

Second, we interpolate values between 8–day periods and fill gaps in EO data using simple linear interpolation (MODIS images are available only every 8 days):

```
dates.lst = as.Date("2006-08-13")+1:8
in.dates = c("2006-08-05", "2006-08-13", "2006-08-21", "2006-08-29")
in.days = as.numeric(strftime(as.Date(c(in.dates)), format = "%j"))
## interpolate values for missing dates in spacetime
library(parallel)
cl <- makeCluster(detectCores())
clusterExport(cl, c("in.days", "dates.lst"))
t1s = parallel::parApply(cl, x1, 1,
      function(y) { try( approx(in.days, as.vector(y), xout=as.numeric(strftime(dates.lst, format = "%j")))$y ) })
t2s = parallel::parApply(cl, x2, 1,
      function(y) { try( approx(in.days, as.vector(y), xout=as.numeric(strftime(dates.lst, format = "%j")))$y ) })
stopCluster(cl)
## remove missing pixels
x.t1s = parallel::mclapply(t1s, length, mc.cores = mc.cores)
t1s[which(!x.t1s==8)] <- list(rep(NA, 8))
t1s = do.call(rbind.data.frame, t1s)
names(t1s) = paste0("LST.day_", dates.lst)
x.t2s = parallel::mclapply(t2s, length, mc.cores = mc.cores)
t2s[which(!x.t2s==8)] <- list(rep(NA, 8))
t2s = do.call(rbind.data.frame, t2s)
names(t2s) = paste0("LST.night_", dates.lst)
```

Now we can make predictions for the target days in August 2006 by using (we run this operation in a loop to avoid RAM overload):

```
for(j in paste(dates.lst)){
  out.tif = paste0("output/MDTEMP_", j, ".tif")
  if(!file.exists(out.tif)){
    hrpred1km@data[,"LST.day"] = t1s[,paste0("LST.day_", j)]
    hrpred1km@data[,"LST.night"] = t2s[,paste0("LST.night_", j)]
    hrpred1km$temp.mean = temp.from.geom(fi=hrpred1km$Lat,
                    as.numeric(strftime(as.Date(j), format = "%j")),
                    a=37.03043, b=-15.43029, elev=hrpred1km$HRdem, t.grad=0.6)
    sel.pix = complete.cases(hrpred1km@data[,eml.TMP$features])
    out = predict(eml.TMP, newdata=hrpred1km@data[sel.pix,eml.TMP$features])
    hrpred1km@data[,paste0("MDTEMP_", j)] = NA
    hrpred1km@data[sel.pix, make.names(paste0("MDTEMP_", j))] = out$data$response * 10
    writeGDAL(hrpred1km[make.names(paste0("MDTEMP_", j))], out.tif, mvFlag = -32768,
              type = "Int16", options = c("COMPRESS=DEFLATE"))
  } else {
    hrpred1km@data[,make.names(paste0("MDTEMP_", j))] = readGDAL(out.tif)$band1
  }
}
```

```
#> output/MDTEMP_2006-08-14.tif has GDAL driver GTiff
#> and has 487 rows and 490 columns
#> output/MDTEMP_2006-08-15.tif has GDAL driver GTiff
#> and has 487 rows and 490 columns
#> output/MDTEMP_2006-08-16.tif has GDAL driver GTiff
#> and has 487 rows and 490 columns
#> output/MDTEMP_2006-08-17.tif has GDAL driver GTiff
#> and has 487 rows and 490 columns
#> output/MDTEMP_2006-08-18.tif has GDAL driver GTiff
#> and has 487 rows and 490 columns
#> output/MDTEMP_2006-08-19.tif has GDAL driver GTiff
#> and has 487 rows and 490 columns
#> output/MDTEMP_2006-08-20.tif has GDAL driver GTiff
#> and has 487 rows and 490 columns
#> output/MDTEMP_2006-08-21.tif has GDAL driver GTiff
#> and has 487 rows and 490 columns
```

To plot these predictions we can either put predictions in the `spacetime` package class (see gstat tutorial[1]), or simply plot them using `sp` package:

```
st.pts = list("sp.points", idsta.pnts, pch = "+", col="black")
spplot(hrpred1km[make.names(paste0("MDTEMP_", dates.lst[c(1,4,8)]))],
       col.regions=plotKML::R_pal[["rainbow_75"]][4:20],
       at = seq(143, 239, length.out=17),
       sp.layout = list(st.pts),
       main="Prediction daily temperature")
#dev.off()
```

In summary, this example shows how to fit spatiotemporal EML with using also seasonality component together with the EO data. It can hence be considered a *complete framework* for spatiotemporal interpolation as both static, dynamic covariates and latitude / elevation are used for model training.

4.2 Spatiotemporal distribution of Fagus sylvatica

In the next example we show how to fit a spatiotemporal model using biological data: occurrences of *Fagus sylvatica*[2] over Europe. This is the domain of **Species Distribution modeling**, except in this case we model distribution of target species also in spacetime. The training (point) data has been compiled for the purpose of the OpenDataScience.eu project, then cleaned and overlaid

[1] https://cran.r-project.org/web/packages/gstat/vignettes/spatio-temporal-kriging.pdf

[2] https://www.gbif.org/species/2882316

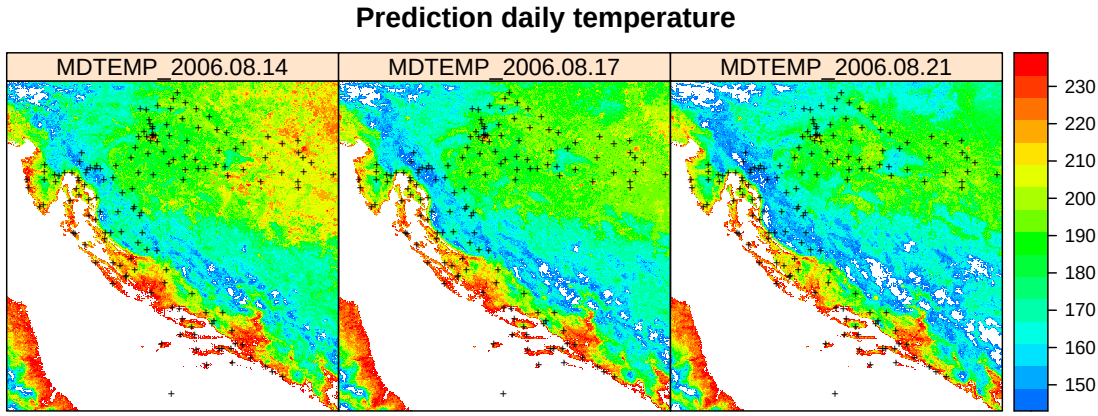


Fig. 4.3 Predictions spacetime daily temperature for August 2006.

vs time-series of Landsat GLAD[3] images and climatic variables (Witjes et al, 2022?; Bonannella et al, 2022?). For more details about the data refer also to the eumap repository[4].

We can load a snapshot of data by using:

```
library(data.table)
library(mlr)
library(sp)
fs.rm = readRDS('input/fagus_sylvatica_st.rds')
occ.pnts = fs.rm[,c("Atlas_class","easting","northing")]
coordinates(occ.pnts) = ~ easting + northing
proj4string(occ.pnts) = "+init=epsg:3035"
occ.pnts = spTransform(occ.pnts, CRS("+init=epsg:4326"))
```

This is a subset of a larger dataset[5] that has been used to produce predictions of distribution of key forest tree species for Europe (you can browse the data via https://maps.opendatascience.eu). The first columns of this dataset show:

[3] https://gitlab.com/geoharmonizer_inea/eumap
[4] https://gitlab.com/geoharmonizer_inea/eumap
[5] https://doi.org/10.5281/zenodo.5818021

```
head(fs.rm[,1:10])
#>          id year postprocess Tile_ID easting northing Atlas_class lc1
#> 9   1499539 2015   spacetime    9689 3948500  2431500           1
#> 38   660325 2008   spacetime    8728 3318500  2283500           1
#> 56   660325 2002   spacetime    8728 3318500  2283500           1
#> 68  2044229 2006   spacetime   11017 4294500  2655500           1
#> 104  586994 2016   spacetime    8724 3204500  2309500           1
#> 111  622055 2016   spacetime    8349 3231500  2226500           1
#>     clm_air.temp_era5.copernicus_av_1km_200..200cm_02.01..02.28_avg_5yrs_eumap_epsg3035_v0.1.tif
#> 9                                                                                           -35
#> 38                                                                                            0
#> 56                                                                                           19
#> 68                                                                                          -35
#> 104                                                                                          31
#> 111                                                                                          -6
#>     clm_air.temp_era5.copernicus_av_1km_200..200cm_02.01..02.28_mean_eumap_epsg3035_v0.1.tif
#> 9                                                                                      -36
#> 38                                                                                      31
#> 56                                                                                      22
#> 68                                                                                     -40
#> 104                                                                                     46
#> 111                                                                                      9
```

The header columns are:

- `id`: is the unique ID of each point;
- `year`: is the year of obsevation;
- `postprocess`: column can have value yearly or spacetime to identify if the temporal reference of an observation comes from the original dataset or is the result of post-processing (yearly for originals, spacetime for post-processed points);
- `Tile_ID`: is as extracted from the 30-km tiling system;
- `easting`: is the easting coordinate of the observation point;
- `northing`: is the northing coordinate of the observation point;
- `Atlas_class`: contains name of the tree species or NULL if it's an absence point coming from LUCAS;
- `lc1`: contains original LUCAS land cover classes or NULL if it's a presence point;

Other columns are the EO and ecological covariates that we use for modeling distribution of *Fagus sylvatica*[6]. We can plot distribution of points over EU using:

[6] https://www.gbif.org/species/2882316

```
library(rnaturalearth)
library(raster)
europe <- rnaturalearth::ne_countries(scale=10, continent = 'europe')
europe <- raster::crop(europe, extent(-24.8,35.2,31,68.5))
op = par(mar=c(0,0,0,0))
plot(europe, col="lightgrey", border="darkgrey", axes=FALSE)
points(occ.pnts[occ.pnts$Atlas_class==1,], pch="+", cex=.8)
par(op)
#dev.off()
```

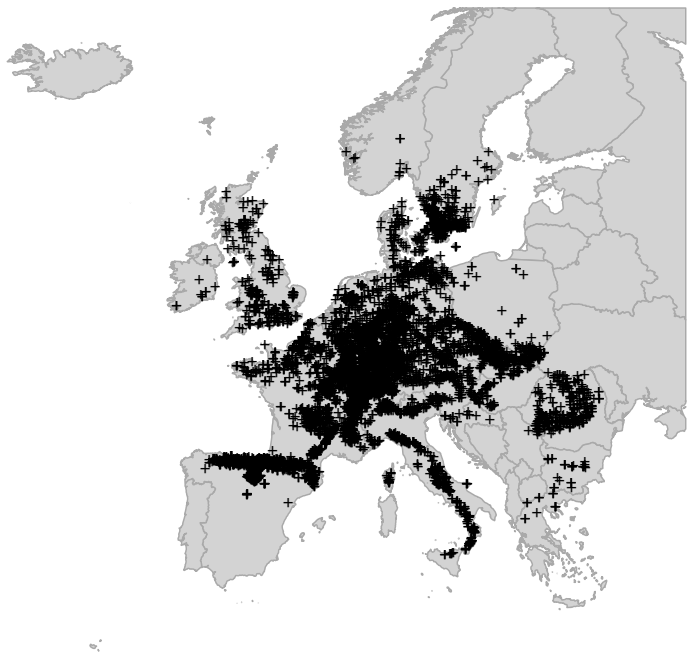


Fig. 4.4 Distribution of occurrence locations for Fagus Sylvatica. Each training points is also referrenced in time and hence can be used to run spacetime overlay.

As in previous examples, we first define the target model formula. We remove from model all other columns that are not used for prediction:

```
covs = grep("id|year|postprocess|Tile_ID|easting|northing|Atlas_class|lc1",
          colnames(fs.rm), value = TRUE, invert = TRUE)
fm.fs = stats::as.formula(paste("Atlas_class ~ ", paste(covs, collapse="+")))
fs.rm$Atlas_class = factor(fs.rm$Atlas_class)
all.vars(fm.fs)[1:5]
#> [1] "Atlas_class"
```

```
#> [2] "clm_air.temp_era5.copernicus_av_1km_200..200cm_02.01..02.28_avg_5yrs_eumap_epsg3035_v0.1.tif"
#> [3] "clm_air.temp_era5.copernicus_av_1km_200..200cm_02.01..02.28_mean_eumap_epsg3035_v0.1.tif"
#> [4] "clm_air.temp_era5.copernicus_av_1km_200..200cm_02.01..02.28_std_eumap_epsg3035_v0.1.tif"
#> [5] "clm_air.temp_era5.copernicus_av_1km_200..200cm_03.01..03.31_avg_5yrs_eumap_epsg3035_v0.1.tif"
```

To speed-up fitting of the models we have prepared a function that wraps all modeling steps:

```
source("mlst_functions.R")
fs.rm0 = fs.rm[runif(nrow(fs.rm))<.2,]
eml.fs = train_sp_eml(data = fs.rm0, formula = fm.fs, blocking = as.factor(fs.rm$Tile_ID))
```

This fits an ensemble model of binary classification models (`classif.ranger`, `classif.xgboost`, `classif.glmnet`) and which basically can be used to predict probability of any training point being classified `0` (not-occurring) or `1` (occurring).

The intermediate models (fine-tuned RF and XGboost) are written to local folder `output` as RDS files. For meta-learner we use a GLM model with binomial link function:

```
#eml.fs = readRDS("output/EML_model.rds")
summary(eml.fs$learner.model$super.model$learner.model)
#>
#> Call:
#> stats::glm(formula = f, family = "binomial", data = getTaskData(.task,
#>     .subset), weights = .weights, model = FALSE)
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -3.4025  -0.0917   0.0666   0.0701   3.2589
#>
#> Coefficients:
#>                 Estimate Std. Error z value Pr(>|z|)
#> (Intercept)       5.6680     0.4800  11.807  < 2e-16 ***
#> classif.ranger   -8.4832     0.6452 -13.147  < 2e-16 ***
#> classif.xgboost   1.2604     1.2307   1.024    0.306
#> classif.glmnet   -3.4816     0.5708  -6.099 1.07e-09 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#>     Null deviance: 8432.56  on 7472  degrees of freedom
#> Residual deviance:  700.97  on 7469  degrees of freedom
#> AIC: 708.97
#>
#> Number of Fisher Scoring iterations: 8
```

The variable importance analysis (RF component) shows that the most important covariates for mapping distribution of Fagus sylvatica are landsat images (which is expected):

```
library(ggplot2)
xl <- as.data.frame(mlr::getFeatureImportance(eml.fs[["learner.model"]][["base.models"]][[1]])$res)
xl$relative_importance = 100*xl$importance/sum(xl$importance)
xl = xl[order(xl$relative_importance, decreasing = T),]
xl$variable = paste0(c(1:182), ". ", xl$variable)
ggplot(data = xl[1:20,], aes(x = reorder(variable, relative_importance), y = relative_importance)) +
  geom_bar(fill = "steelblue",
           stat = "identity") +
  coord_flip() +
  labs(title = "Variable importance",
       x = NULL,
       y = NULL) +
  theme_bw() + theme(text = element_text(size=15))
```

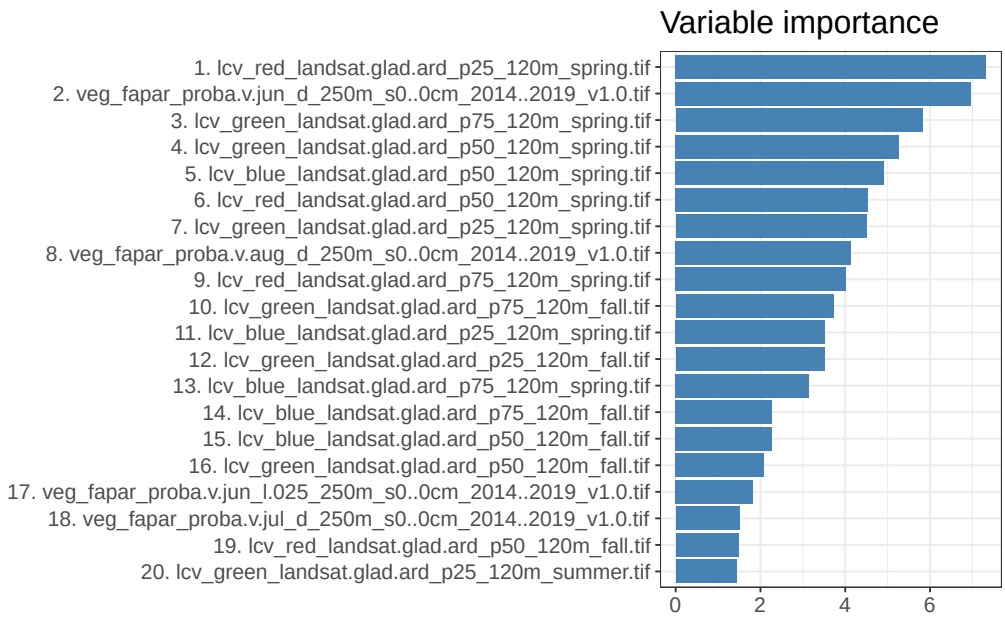


Fig. 4.5 Variable importance for spatiotemporal model used to predict distribution of Fagus sylvatica.

To produce spacetime predictions for some tiles (120-m spatial resolution) we can run:

```
m1 = predict_tiles(input = "9690.2015",  model = eml.fs)
#> [1] "9690 - reading the data"
#> Warning in dir.create(tmp_folder, recursive = TRUE): 'output//9690' already
```

```
#> exists
#> [1] "9690 - running predictions"
#> Warning in if (class(probability_map) == "try-error") {: the condition has
#> length > 1 and only the first element will be used
#> [1] "9690 - writing files"
m2 = predict_tiles(input = "9690.2017",  model = eml.fs)
#> [1] "9690 - reading the data"
#> Warning in dir.create(tmp_folder, recursive = TRUE): 'output//9690' already
#> exists
#> [1] "9690 - running predictions"
#> Warning in if (class(probability_map) == "try-error") {: the condition has
#> length > 1 and only the first element will be used
#> [1] "9690 - writing files"
m3 = predict_tiles(input = "9690.2019",  model = eml.fs)
#> [1] "9690 - reading the data"
#> Warning in dir.create(tmp_folder, recursive = TRUE): 'output//9690' already
#> exists
#> [1] "9690 - running predictions"
#> Warning in if (class(probability_map) == "try-error") {: the condition has
#> length > 1 and only the first element will be used
#> [1] "9690 - writing files"
m1$Prob.2015 = m1$Prob
m1$Prob.2017 = m2$Prob
m1$Prob.2019 = m3$Prob
```

We can compare predictions of the probability of occurrence of the target species for two years next to each other by using:

```
pts = list("sp.points", spTransform(occ.pnts[occ.pnts$Atlas_class==1,], CRS("+init=epsg:3035")),
           pch = "+", col="black")
spplot(m1[,c("Prob.2015","Prob.2017","Prob.2019")], col.regions=rev(bpy.colors()),
  sp.layout = list(pts),
  main="Predictions Fagus Sylvatica")
#dev.off()
```

In this case there seems to be no drastic changes in the distribution of the target species through time, which is also expected because forest species distribution change on a scale of 50 to 100 year and not on a scale of few years. Some changes in distribution of the species, however, can be detected nevertheless. These can be due to abrupt events such as pest-pandemics, fires, floods or landslides or clear cutting of forests of course.

We can also plot the images using the `plotKML` package so that we can open and visualize predictions also in Google Earth or similar:

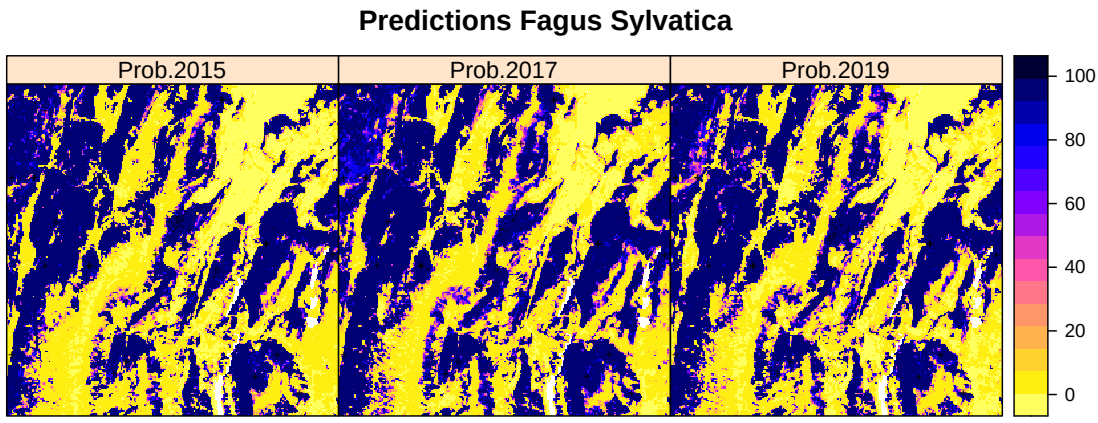


Fig. 4.6 Predicted probability of occurrence for Fagus Sylvatica for three periods.

```
library(plotKML)
for(j in c(2015,2017,2019)){
  kml(m1, colour=m1@data[,paste0("Prob.", j)], file.name=paste0("prob.", j, ".kml"),
      raster_name = paste0("prob.", j, ".png"),
      colour_scale = SAGA_pal[["SG_COLORS_YELLOW_BLUE"]],
      z.lim=c(0,100),
      TimeSpan.begin = as.Date(paste0(j, "-01-01")), TimeSpan.end = as.Date(paste0(j, "-12-31")))
}
```

In this case we attach to each prediction also the `TimeSpan.begin` and `TimeSpan.end` which means that Google Earth will recognize the temporal reference of the predictions. Opening predictions in Google Earth allows us to do some interpretation of produced maps and also analyze how much are the changes in vegetation cover connected with relief, proximity to urban areas, possible fire / flood events and similar.

4.3 Summary notes

In this tutorial we have reviewed some aspects of spatial and spatiotemporal data and demonstrated how to use ML, specifically Ensemble ML, to train spatiotemporal models and produce

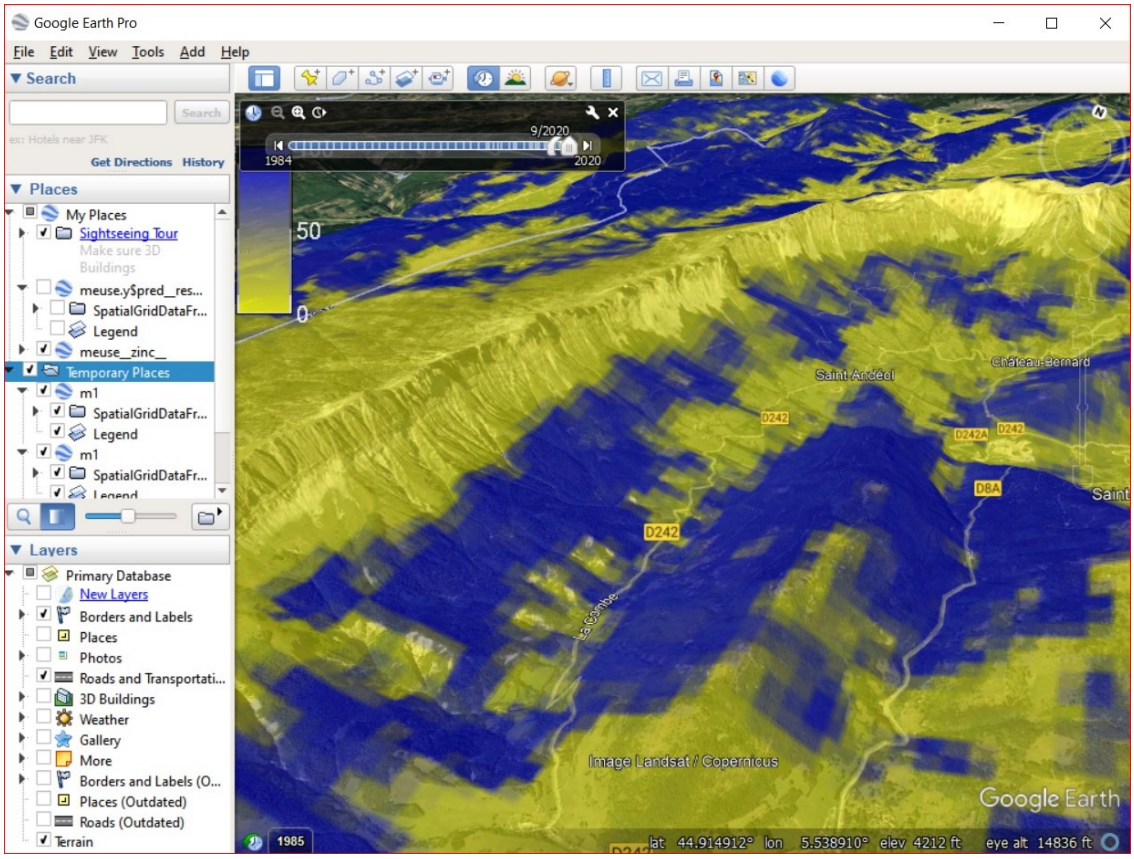


Fig. 4.7 Spacetime predictions of distribution of Fagus Sylvatica visualized as time-series data in Google Earth.

time-series of predictions. We have also shown, using some synthetic and real-life datasets, how incorrectly setting up training and cross-validation can lead to over-fitting problems. This was done to help users realize that Machine Learning, however trivial it might seem, is not a click-a-button process and solid knowledge and understanding of advanced statistics (regression, hypothesis testing, sampling and resampling, probability theory) is still required.

For spatiotemporal models, we recommend combining covariates that can represent both long-term or accumulated effects of climate, together with covariates that can represent daily to monthly oscillation of variables such as soil moisture, temperatures and similar. During the design of the modeling system, we highly recommend **trying to understand ecology and processes behind variable of interest** first, then designing the modeling system to best reflect expert knowledge. The example with RF over-fitting data in Gasch et al (2015) shows how in-depth understanding of the problem can help design modeling framework and prevent from over-fitting problems and similar. Witjes et al (2022?) and Bonannella et al (2022?) describe a more comprehensive framework for spatiotemporal ML which can even be run on large datasets.

Where time-series EO data exists, these can be also incorporated into the mapping algorithm as shown with three case studies. For spacetime overlays we recommend using Cloud-Optimized

GeoTIFFs and the `terra` package (Hijmans, 2019) which helps speed-up overlays. Other options for efficient overlay are the `stars` and `gdalcubes` package (Appel and Pebesma, 2019).

Spatiotemporal datasets can be at the order of magnitude larger, hence it is important, when implementing analysis of spatiotemporal data, to consider **computing optimization**, which typically implies:

- Running operations in parallel;
- Separating fine-tuning and parameter optimization (best to run on subset and save computing time) from predictions,
- Using tiling systems to run overlay, predictions and visualizations,

Finally, we recommend following these generic steps to fit spatiotemporal models:

0. Define target of interest and design the modeling framework by understanding ecology and processes behind variable of interest.

1. Prepare training (points) data and a data cube with all covariates referenced in spacetime.

2. Overlay points in spacetime, create a spatiotemporal regression-matrix.

3. Add seasonal components, fine-tune initial model, reduce complexity as much as possible, and produce production-ready spatiotemporal prediction model (usually using Ensemble Machine Learning).

4. Run mapping accuracy assessment and determine prediction uncertainty including the per pixel uncertainty.

5. Generate predictions in spacetime — create time-series of predictions.

6. (optional) Run change-detection / trend analysis and try to detect main drivers of positive / negative trends (Witjes et al, 2022?).

7. Deploy predictions as Cloud-Optimized GeoTIFF and produce the final report with mapping accuracy, variable importance.

Ensemble ML framework we used here clearly offers many benefits, but it also comes at a cost of at the order of magnitude higher computational load. Also interpretation of such models can be a cumbersome as there a multiple learners plus a meta-learner, so it often becomes difficult to track-back individual relationship between variables. To help increase confidence in the produced models, we recommend studying the Interpretable Machine Learning[7] methods (Molnar, 2020), running additional model diagnostics, and intensively plotting data in space and spacetime and feature space.

Note that the `mlr` package is discontinued, so some of the examples above might become unstable with time. You are advised instead to use the new mlr3 package[8].

[7] https://christophm.github.io/interpretable-ml-book/

[8] https://mlr3.mlr-org.com/

Chapter 5

Multi-scale spatial prediction models

You are reading the work-in-progress Spatial and spatiotemporal interpolation using Ensemble 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-prediction-eml/.

5.1 Rationale for multiscale models

In the previous examples we have shown how to fit spatial and spatiotemporal models to generate predictions using multiple covariate layers. In practice spatial layers used for predictive mapping could come and different spatial scales i.e. they could be represent different part of spatial variation. There are at least two scales of spatial variation (Hengl et al, 2021):

- **Coarse scale** e.g. representing effects of planetary climate;
- **Fine scale** e.g. representing meso-relief and local conditions;

In fact, we can imagine that spatial variation can probably be decomposed into different scale components, as illustrated in plot below.

The idea of modeling soil spatial variation at different scales can be traced back to the work of McBratney (1998). That also suggests that we could produce predictions models of different components of variation, then sum the components to produce ensemble prediction. The rationale for this, in the case of large datasets, is that we can (a) significantly reduce size of the data, (b) separate and better focus modeling based on the component of variation.

5.2 Fitting and predicting with multiscale models

In the next example we use EML to make spatial predictions using data-set with two sets of covariates basically at different resolutions 250-m and 100-m. For this we use the Edgeroi data-set (Malone et al, 2009) used commonly in the soil science to demonstrate 3D soil mapping of

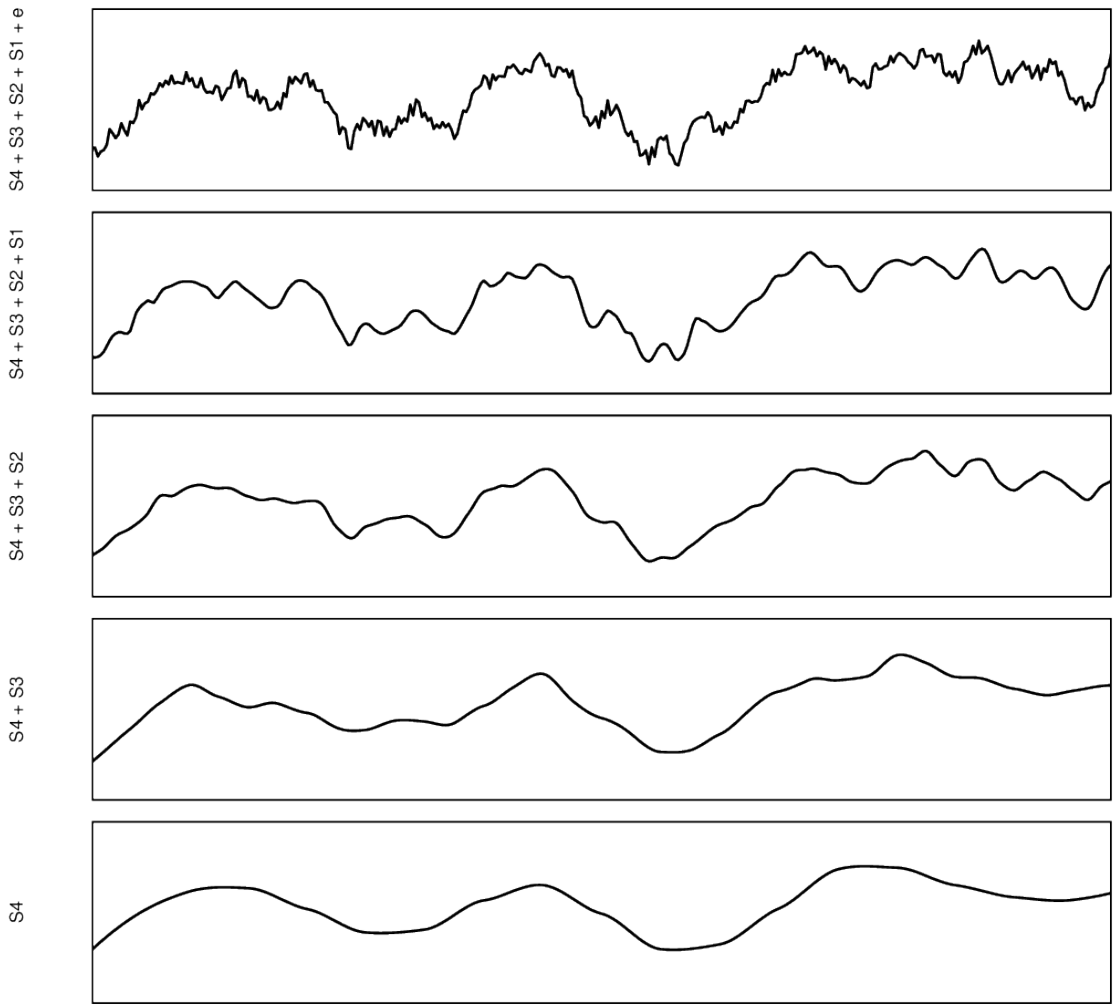


Fig. 5.1 Decomposition of a signal of spatial variation into four components plus noise. Based on McBratney (1998).

soil organic carbon (g/kg) based on samples taken from diagnostic soil horizons (multiple depth intervals):

```
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"))
out.file = paste(getwd(), "output/edgeroi/edgeroi_training_points.gpkg", sep="/")
#if(!file.exists("out.file")){
```

```
#  writeOGR(edgeroi.sp, out.file, layer="edgeroi_training_points", driver = "GPKG")
#}
```

We can fit two independent EML's using the two sets of covariates and then produce final predictions by combining them. We will refer to the two models as coarse and fine-scale models. The fine-scale models will often be much larger datasets and require serious computing capacity.

5.3 Coarse-scale model

First we use the 250-m resolution covariates:

```
load("input/edgeroi.grids.rda")
gridded(edgeroi.grids) <- ~x+y
proj4string(edgeroi.grids) <- CRS("+init=epsg:28355")
ov2 <- over(edgeroi.sp, edgeroi.grids)
ov2$SOURCEID <- edgeroi.sp$SOURCEID
ov2$x = edgeroi.sp@coords[,1]
ov2$y = edgeroi.sp@coords[,2]
```

This is a 3D soil data set, so we also use the horizon DEPTH to explain distribution of SOC in soil:

```
source("PSM_functions.R")
h2 <- hor2xyd(edgeroi$horizons)
## regression matrix:
rm2 <- plyr::join_all(dfs = list(edgeroi$sites, h2, ov2))
#> Joining by: SOURCEID
#> Joining by: SOURCEID
formulaStringP2 <- ORCDRC ~ DEMSRT5+TWISRT5+EV1MOD5+EV2MOD5+EV3MOD5+DEPTH
rmP2 <- rm2[complete.cases(rm2[,all.vars(formulaStringP2)]),]
str(rmP2[,all.vars(formulaStringP2)])
#> 'data.frame':    4972 obs. of  7 variables:
#>  $ ORCDRC : num  8.5 7.3 5 4.7 4.7 ...
#>  $ DEMSRT5: num  198 198 198 198 198 198 185 185 185 185 ...
#>  $ TWISRT5: num  19.5 19.5 19.5 19.5 19.5 19.5 19.2 19.2 19.2 19.2 ...
#>  $ EV1MOD5: num  1.14 1.14 1.14 1.14 1.14 1.14 -4.7 -4.7 -4.7 -4.7 ...
#>  $ EV2MOD5: num  1.62 1.62 1.62 1.62 1.62 1.62 3.46 3.46 3.46 3.46 ...
#>  $ EV3MOD5: num  -5.74 -5.74 -5.74 -5.74 -5.74 -5.74 0.01 0.01 0.01 0.01 ...
#>  $ DEPTH  : num  11.5 17.5 26 55 80 ...
```

We can now fit an EML directly by using the derived regression matrix:

```
if(!exists("m.oc")){
  m.oc = train.spLearner.matrix(rmP2, formulaStringP2, edgeroi.grids,
                        parallel=FALSE, cov.model="nugget", cell.size=1000)
}
#> as.geodata: 4655 replicated data locations found.
#>  Consider using jitterDupCoords() for jittering replicated locations.
#> WARNING: there are data at coincident or very closed locations, some of the geoR's functions may not work.
#>  Use function dup.coords() to locate duplicated coordinates.
#>  Consider using jitterDupCoords() for jittering replicated locations
#> # weights:  25
#> initial  value 253895.808438
#> iter  10 value 131424.395513
#> iter  20 value 92375.545449
#> iter  30 value 88023.497878
#> iter  40 value 78161.622563
#> iter  50 value 71869.588437
#> iter  60 value 69482.655270
#> iter  70 value 68642.175713
#> iter  80 value 68405.024865
#> iter  90 value 68402.034341
#> final  value 68402.000242
#> converged
#> # weights:  25
#> initial  value 254790.970425
#> final  value 136782.219163
#> converged
#> # weights:  25
#> initial  value 285010.126650
#> final  value 135478.529982
#> converged
#> # weights:  25
#> initial  value 253832.562811
#> final  value 137484.280876
#> converged
#> # weights:  25
#> initial  value 254385.881547
#> iter  10 value 98551.221418
#> iter  20 value 94579.214923
#> iter  30 value 93473.664614
#> iter  40 value 93169.177514
#> iter  50 value 93139.660457
#> iter  60 value 93111.617240
#> iter  70 value 93111.256287
#> iter  80 value 93109.156591
#> iter  90 value 93099.925818
#> iter 100 value 93021.880998
#> final  value 93021.880998
```

```
#> stopped after 100 iterations
#> # weights:  25
#> initial  value 233465.782262
#> final  value 134647.206038
#> converged
#> # weights:  25
#> initial  value 246624.689888
#> final  value 138702.343415
#> converged
#> # weights:  25
#> initial  value 241227.341802
#> final  value 138599.168021
#> converged
#> # weights:  25
#> initial  value 245735.599010
#> final  value 131152.446689
#> converged
#> # weights:  25
#> initial  value 258267.657849
#> iter  10 value 97368.003255
#> iter  20 value 91058.331259
#> iter  30 value 88735.472097
#> iter  40 value 78495.790097
#> iter  50 value 72384.348608
#> iter  60 value 69221.579541
#> iter  70 value 68248.107158
#> iter  80 value 68073.306877
#> iter  80 value 68073.306456
#> iter  90 value 68072.812887
#> iter  90 value 68072.812254
#> final  value 68072.784171
#> converged
#> # weights:  25
#> initial  value 268786.613045
#> iter  10 value 128937.282143
#> iter  20 value 105536.706972
#> iter  30 value 100970.717402
#> iter  40 value 89676.958144
#> iter  50 value 80499.715395
#> iter  60 value 76269.748745
#> iter  70 value 74645.990665
#> iter  80 value 74429.114194
#> iter  90 value 74041.488710
#> iter 100 value 73877.378810
#> final  value 73877.378810
#> stopped after 100 iterations
```

The **geoR** package here reports problems as the data set is 3D and hence there are spatial duplicates. We can ignore this problem and use the pre-defined cell size of 1-km for spatial blocking, although in theory one can also fit 3D variograms and then determine blocking parameter using training data.

The results show that the EML model is significant:

```
summary(m.oc@spModel$learner.model$super.model$learner.model)
#>
#> Call:
#> stats::lm(formula = f, data = d)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -17.668  -0.984  -0.066   0.711  64.291
#>
#> Coefficients:
#>               Estimate Std. Error t value Pr(>|t|)
#> (Intercept)    0.11730    0.18365   0.639  0.52302
#> regr.ranger    1.12706    0.02474  45.553  < 2e-16 ***
#> regr.xgboost  -0.37833    0.07448  -5.080 3.92e-07 ***
#> regr.nnet     -0.04227    0.02399  -1.762  0.07808 .
#> regr.ksvm      0.08299    0.02900   2.861  0.00423 **
#> regr.cvglmnet -0.05140    0.03879  -1.325  0.18525
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 3.06 on 4966 degrees of freedom
#> Multiple R-squared:  0.6938, Adjusted R-squared:  0.6935
#> F-statistic:  2250 on 5 and 4966 DF,  p-value: < 2.2e-16
```

We can now predict values at e.g. 5-cm depth by adding a dummy spatial layer with all fixed values:

```
out.tif = "output/edgeroi/pred_oc_250m.tif"
edgeroi.grids$DEPTH <- 5
if(!exists("edgeroi.oc")){
  edgeroi.oc = predict(m.oc, edgeroi.grids[,m.oc@spModel$features])
}
#> Predicting values using 'getStackedBaseLearnerPredictions'...TRUE
#> Deriving model errors using forestError package...TRUE
if(!file.exists(out.tif)){
  writeGDAL(edgeroi.oc$pred["response"], out.tif,
            options = c("COMPRESS=DEFLATE"))
  writeGDAL(edgeroi.oc$pred["model.error"], "output/edgeroi/pred_oc_250m_pe.tif",
            options = c("COMPRESS=DEFLATE"))
}
```

which shows the following:

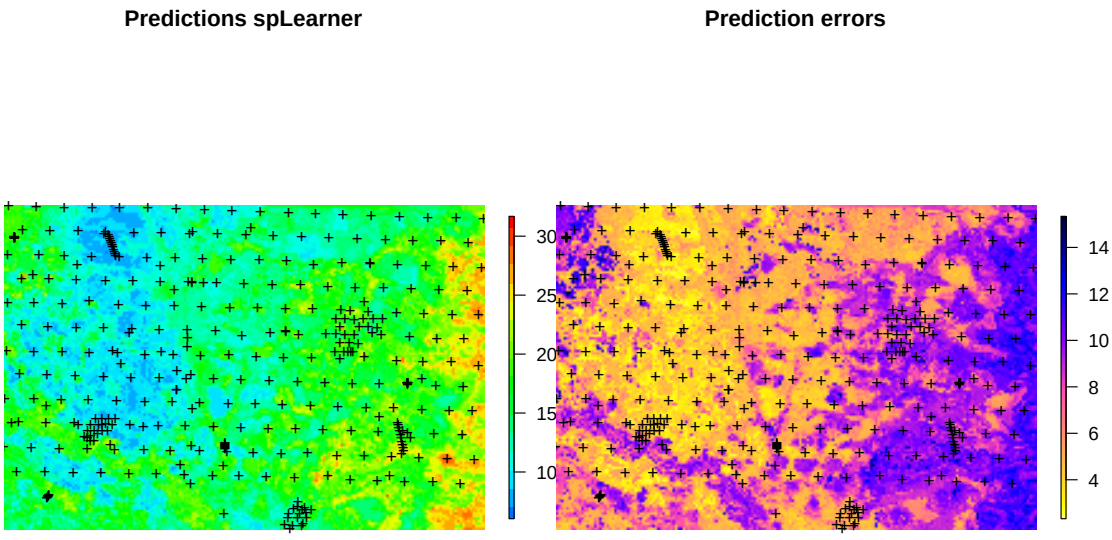


Fig. 5.2 Predicted SOC content using 250-m covariates.

The average prediction error in the map is somewhat higher than the average error from the model fitting:

```
summary(edgeroi.oc$pred$model.error)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>   2.323   4.710   6.340   6.929   9.257  15.338
```

This is because we are predicting the top-soil SOC, which is exponentially higher at the soil surface and hence average model errors for top soil should be slightly larger than the mean error for the whole soil.

5.4 Fine-scale model

We can now fit the fine-scale model independently from the coarse-scale model using the 100-m resolution covariates. In this case the 100-m covariates are based on Landsat 8 and gamma radiometrics images (see `?edgeroi` for more details):

```
edgeroi.grids100 = readRDS("input/edgeroi.grids.100m.rds")
#gridded(edgeroi.grids100) <- ~x+y
#proj4string(edgeroi.grids100) <- CRS("+init=epsg:28355")
ovF <- over(edgeroi.sp, edgeroi.grids100)
ovF$SOURCEID <- edgeroi.sp$SOURCEID
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
formulaStringPF <- ORCDRC ~ MVBSRT6+TI1LAN6+TI2LAN6+PCKGAD6+RUTGAD6+PCTGAD6+DEPTH
rmPF <- rmF[complete.cases(rmF[,all.vars(formulaStringPF)]),]
str(rmPF[,all.vars(formulaStringPF)])
#> 'data.frame':    5001 obs. of  8 variables:
#>  $ ORCDRC : num  8.5 7.3 5 4.7 4.7 ...
#>  $ MVBSRT6: num  5.97 5.97 5.97 5.97 5.97 5.97 6.7 6.7 6.7 6.7 ...
#>  $ TI1LAN6: num  31.8 31.8 31.8 31.8 31.8 31.8 14.3 14.3 14.3 14.3 ...
#>  $ TI2LAN6: num  32.9 32.9 32.9 32.9 32.9 32.9 22.1 22.1 22.1 22.1 ...
#>  $ PCKGAD6: num  1.39 1.39 1.39 1.39 1.39 1.39 1.06 1.06 1.06 1.06 ...
#>  $ RUTGAD6: num  0.14 0.14 0.14 0.14 0.14 0.14 0.16 0.16 0.16 0.16 ...
#>  $ PCTGAD6: num  7.82 7.82 7.82 7.82 7.82 7.82 6.48 6.48 6.48 6.48 ...
#>  $ DEPTH  : num  11.5 17.5 26 55 80 ...
```

We fit the 2nd fine-scale model:

```
if(!exists("m.ocF")){
  m.ocF = train.spLearner.matrix(rmPF, formulaStringPF, edgeroi.grids100,
                        parallel=FALSE, cov.model="nugget", cell.size=1000)
}
#> as.geodata: 4655 replicated data locations found.
#>  Consider using jitterDupCoords() for jittering replicated locations.
#> WARNING: there are data at coincident or very closed locations, some of the geoR's functions may not work.
#>  Use function dup.coords() to locate duplicated coordinates.
#>  Consider using jitterDupCoords() for jittering replicated locations
#> # weights:  28
#> initial  value 259952.010839
#> iter  10 value 101252.397395
#> iter  20 value 92849.746212
#> iter  30 value 84138.890538
#> iter  40 value 81171.674547
#> iter  50 value 80244.316068
#> iter  60 value 79942.793909
#> iter  70 value 79368.296567
#> iter  80 value 78323.201957
#> iter  90 value 77181.404075
#> iter 100 value 76708.190347
```

```
#> final  value 76708.190347
#> stopped after 100 iterations
#> # weights:  28
#> initial  value 226902.901029
#> iter  10 value 134520.116202
#> iter  20 value 106665.533239
#> iter  30 value 100456.523529
#> iter  40 value 94907.032527
#> iter  50 value 94598.860459
#> iter  60 value 94311.934401
#> iter  70 value 93110.357174
#> iter  80 value 92843.643684
#> iter  90 value 92584.240506
#> iter 100 value 92181.626754
#> final  value 92181.626754
#> stopped after 100 iterations
#> # weights:  28
#> initial  value 234725.323531
#> iter  10 value 99738.688037
#> iter  20 value 95202.777671
#> iter  30 value 93332.714310
#> iter  40 value 84502.258499
#> iter  50 value 81245.631274
#> iter  60 value 80530.199169
#> iter  70 value 79322.812977
#> iter  80 value 78753.418715
#> iter  90 value 78202.739233
#> iter 100 value 76867.515069
#> final  value 76867.515069
#> stopped after 100 iterations
#> # weights:  28
#> initial  value 264624.170952
#> iter  10 value 101566.765280
#> iter  20 value 93105.271953
#> iter  30 value 79221.563953
#> iter  40 value 75437.096559
#> iter  50 value 74819.981899
#> iter  60 value 74258.787761
#> iter  70 value 72481.383976
#> iter  80 value 71415.613349
#> iter  90 value 69310.427661
#> iter 100 value 66134.495814
#> final  value 66134.495814
#> stopped after 100 iterations
#> # weights:  28
#> initial  value 269334.603789
#> iter  10 value 114589.803267
```

```
#> iter  20 value 98585.812766
#> iter  30 value 95611.345448
#> iter  40 value 94119.323815
#> iter  50 value 92069.210575
#> iter  60 value 90209.532038
#> iter  70 value 86238.674926
#> iter  80 value 82014.171622
#> iter  90 value 78106.825699
#> iter 100 value 76099.552544
#> final  value 76099.552544
#> stopped after 100 iterations
#> # weights:  28
#> initial  value 259884.878897
#> iter  10 value 103726.379375
#> iter  20 value 94570.752266
#> iter  30 value 76708.677450
#> iter  40 value 73787.991288
#> iter  50 value 73126.652279
#> iter  60 value 72979.797218
#> iter  70 value 72965.952547
#> iter  80 value 72468.280591
#> iter  90 value 72359.464672
#> iter 100 value 71873.895760
#> final  value 71873.895760
#> stopped after 100 iterations
#> # weights:  28
#> initial  value 285193.065615
#> iter  10 value 102340.204856
#> iter  20 value 89532.696472
#> iter  30 value 83391.890007
#> iter  40 value 79203.819730
#> iter  50 value 75520.520094
#> iter  60 value 71641.894828
#> iter  70 value 66583.248718
#> iter  80 value 64316.930097
#> iter  90 value 64067.020769
#> iter 100 value 63723.399424
#> final  value 63723.399424
#> stopped after 100 iterations
#> # weights:  28
#> initial  value 209074.598360
#> iter  10 value 92158.672417
#> iter  20 value 88477.518932
#> iter  30 value 83611.323463
#> iter  40 value 80398.938707
#> iter  50 value 75639.793693
#> iter  60 value 72497.710902
```

```
#> iter  70 value 68574.884188
#> iter  80 value 66515.331380
#> iter  90 value 64170.821850
#> iter 100 value 63336.087094
#> final  value 63336.087094
#> stopped after 100 iterations
#> # weights:  28
#> initial  value 264913.914581
#> iter  10 value 90799.203779
#> iter  20 value 83326.307020
#> iter  30 value 71495.281328
#> iter  40 value 66078.704046
#> iter  50 value 64850.594409
#> iter  60 value 64005.390526
#> iter  70 value 63563.194716
#> iter  80 value 62770.259605
#> iter  90 value 62084.800044
#> iter 100 value 60518.371937
#> final  value 60518.371937
#> stopped after 100 iterations
#> # weights:  28
#> initial  value 254133.236100
#> iter  10 value 117010.387846
#> iter  20 value 96496.685965
#> iter  30 value 81754.875719
#> iter  40 value 80702.591180
#> iter  50 value 79637.660106
#> iter  60 value 77195.047203
#> iter  70 value 74660.187056
#> iter  80 value 70931.471910
#> iter  90 value 69014.578845
#> iter 100 value 68409.495245
#> final  value 68409.495245
#> stopped after 100 iterations
#> # weights:  28
#> initial  value 325116.468080
#> iter  10 value 120203.210463
#> iter  20 value 115964.153279
#> iter  30 value 90105.298970
#> iter  40 value 79437.548718
#> iter  50 value 70168.831437
#> iter  60 value 69050.767980
#> iter  70 value 68627.297281
#> iter  80 value 68156.983626
#> iter  90 value 67374.568621
#> iter 100 value 67164.637038
#> final  value 67164.637038
```

```
#> stopped after 100 iterations
summary(m.ocF@spModel$learner.model$super.model$learner.model)
#>
#> Call:
#> stats::lm(formula = f, data = d)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -19.222  -0.952  -0.047   0.726  61.162
#>
#> Coefficients:
#>               Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   -0.47135    0.11510  -4.095 4.29e-05 ***
#> regr.ranger    1.10204    0.02467  44.667  < 2e-16 ***
#> regr.xgboost   0.03329    0.07566   0.440    0.660
#> regr.nnet      0.03135    0.02196   1.428    0.153
#> regr.ksvm     -0.02556    0.02965  -0.862    0.389
#> regr.cvglmnet -0.03575    0.02887  -1.238    0.216
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 2.967 on 4995 degrees of freedom
#> Multiple R-squared:  0.7146, Adjusted R-squared:  0.7143
#> F-statistic:  2501 on 5 and 4995 DF,  p-value: < 2.2e-16
```

which shows that the 100-m resolution covariates help make even more accurate predictions with R-square about 0.7. We can also make predictions at 5-cm depth by using (note: this takes almost 6x more time to compute predictions than for 250-m resolution data):

```
edgeroi.grids100$DEPTH <- 5
sel.grid = complete.cases(edgeroi.grids100@data[,m.ocF@spModel$features])
if(!exists("edgeroi.ocF")){
  edgeroi.ocF = predict(m.ocF, edgeroi.grids100[sel.grid, m.ocF@spModel$features])
}
#> Predicting values using 'getStackedBaseLearnerPredictions'...TRUE
#> Deriving model errors using forestError package...TRUE
out.tif = "output/edgeroi/pred_oc_100m.tif"
if(!file.exists(out.tif)){
  writeGDAL(edgeroi.ocF$pred["response"], out.tif, options = c("COMPRESS=DEFLATE"))
  writeGDAL(edgeroi.ocF$pred["model.error"], "output/edgeroi/pred_oc_100m_pe.tif", options = c("COMPRESS=DEFLATE"))
}
```

which shows the following:

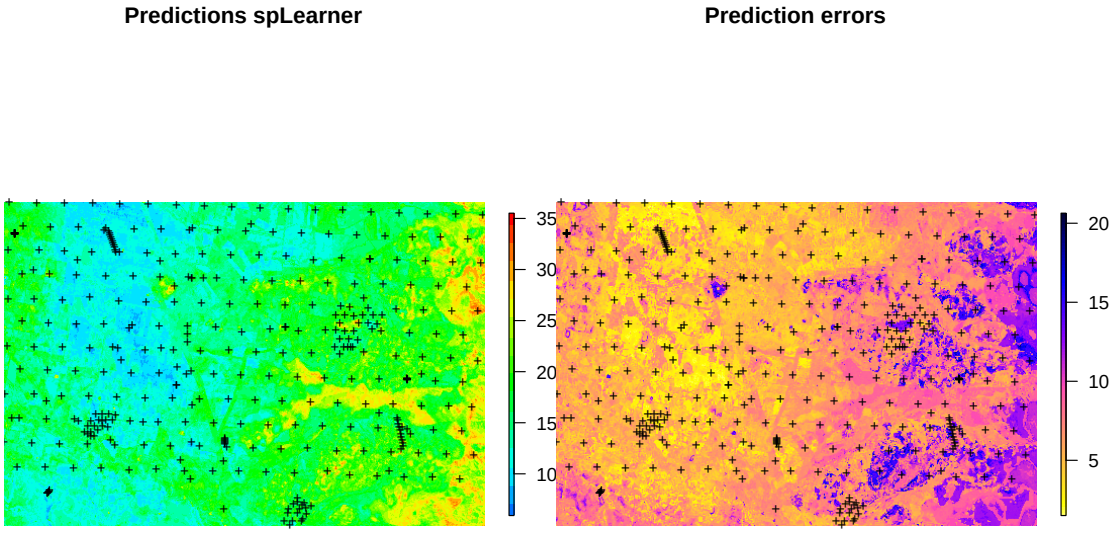


Fig. 5.3 Predicted SOC content using 100-m covariates.

5.5 Merging multi-scale predictions

If we compare the coarse scale and fine scale predictions we see:

Overall there is a match between general patterns but there are also differences locally. This is to expect as the two models are fitted independently using completely different covariates. We can merge the two predictions and produce the final ensemble prediction by using the following principles:

- User prediction errors per pixel as weights so that more accurate predictions get higher weights,
- Derive propagated error using the pooled variance based on individual predictions and errors,

Before we run this operation, we need to downscale the maps to the same grid, best using Cubic-splines in GDAL:

```
edgeroi.grids100@bbox
#>       min     max
#> x  741400  789000
#> y 6646000 6678100
outD.file = "output/edgeroi/pred_oc_250m_100m.tif"
```

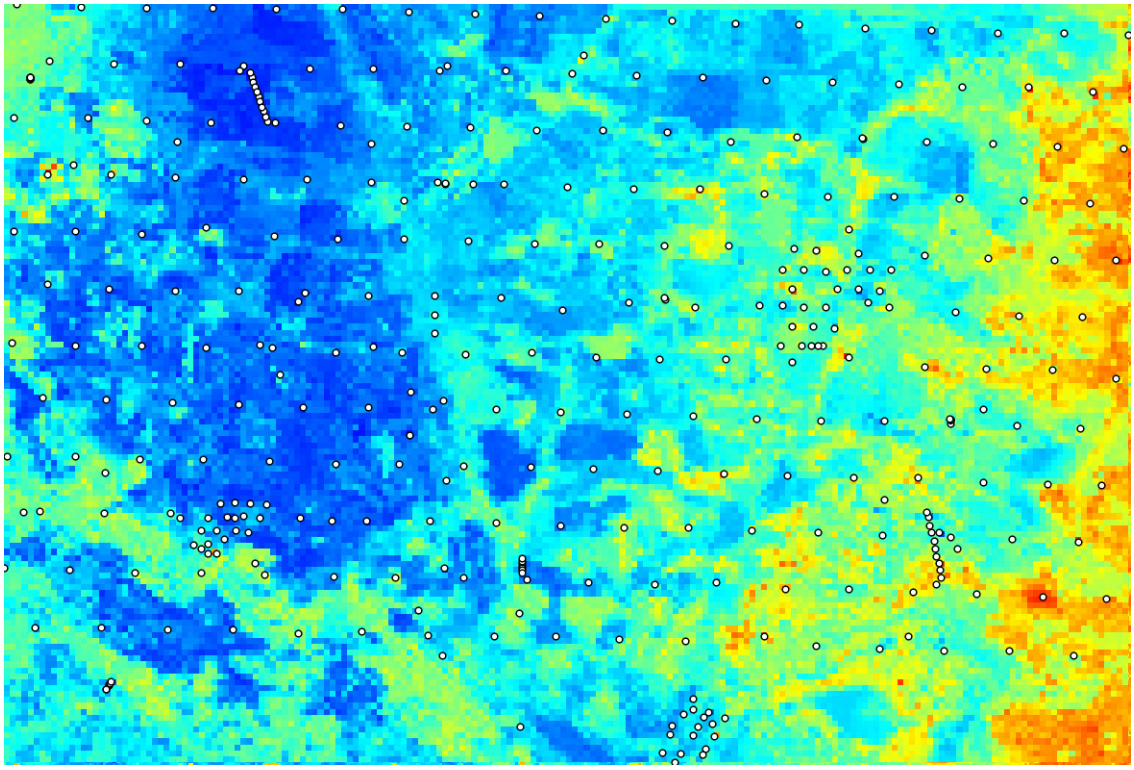


Fig. 5.4 Coarse-scale and fine-scale predictions of soil organic carbon at 5-cm depth for the Edgeroi study area.

```
if(!file.exists(outD.file)){
  system(paste0('gdalwarp output/edgeroi/pred_oc_250m.tif ', outD.file,
         ' -r \"cubicspline\" -te 741400 6646000 789000 6678100 -tr 100 100 -overwrite'))
  system(paste0('gdalwarp output/edgeroi/pred_oc_250m_pe.tif output/edgeroi/pred_oc_250m_100m_pe.tif',
         ' -r \"cubicspline\" -te 741400 6646000 789000 6678100 -tr 100 100 -overwrite'))
}
```

We can now read the downscaled predictions, and merge them using the prediction errors as weights (weighted average per pixel):

```
sel.pix = edgeroi.ocF$pred@grid.index
edgeroi.ocF$pred$responseC = readGDAL("output/edgeroi/pred_oc_250m_100m.tif")$band1[sel.pix]
#> output/edgeroi/pred_oc_250m_100m.tif has GDAL driver GTiff
#> and has 321 rows and 476 columns
edgeroi.ocF$pred$model.errorC = readGDAL("output/edgeroi/pred_oc_250m_100m_pe.tif")$band1[sel.pix]
#> output/edgeroi/pred_oc_250m_100m_pe.tif has GDAL driver GTiff
#> and has 321 rows and 476 columns
X = comp.var(edgeroi.ocF$pred@data, r1="response", r2="responseC", v1="model.error", v2="model.errorC")
edgeroi.ocF$pred$responseF = X$response
```

```
out.tif = "output/edgeroi/pred_oc_100m_merged.tif"
if(!file.exists(out.tif)){
  writeGDAL(edgeroi.ocF$pred["responseF"], out.tif, options = c("COMPRESS=DEFLATE"))
}
```

The final map of predictions is a combination of the two independently produced predictions (Hengl et al, 2021):

```
plot(raster(edgeroi.ocF$pred["responseF"]), col=R_pal[["rainbow_75"]][4:20],
  main="Merged predictions spLearner", axes=FALSE, box=FALSE)
points(edgeroi.sp, pch="+", cex=.8)
```

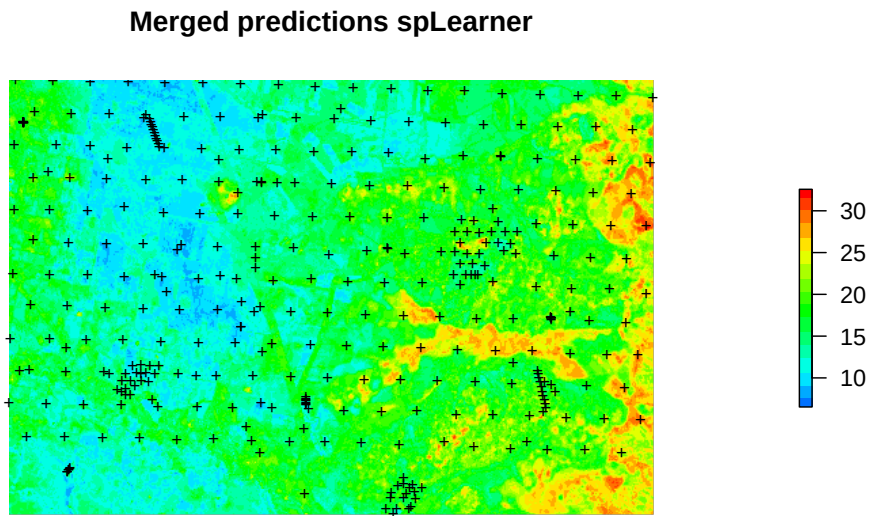


Fig. 5.5 Merged predictions (coarse+fine scale) of SOC content at 100-m.

To merge the prediction errors, we use the pooled variance formula (Rudmin, 2010):

```
comp.var
#> function (x, r1, r2, v1, v2)
#> {
#>     r = rowSums(x[, c(r1, r2)] * 1/(x[, c(v1, v2)]^2))/rowSums(1/(x[,
#>         c(v1, v2)]^2))
```

```
#>     v = sqrt(rowMeans(x[, c(r1, r2)]^2 + x[, c(v1, v2)]^2) -
#>         rowMeans(x[, c(r1, r2)])^2)
#>     return(data.frame(response = r, stdev = v))
#> }
#> <bytecode: 0x39f5e3f8>
edgeroi.ocF$pred$model.errorF = X$stdev
out.tif = "output/edgeroi/pred_oc_100m_merged_pe.tif"
if(!file.exists(out.tif)){
  writeGDAL(edgeroi.ocF$pred["model.errorF"], out.tif, options = c("COMPRESS=DEFLATE"))
}
```

So in summary, merging multi-scale predictions is a straight forward process, but it assumes that the reliable prediction errors are available for both coarse and fine scale predictions. The pooled variance might show higher errors where predictions between independent models differ significantly and this is correct. The 2-scale Ensemble Machine Learning method of Predictive Soil Mapping was used, for example, to produce predictions of soil properties and nutrients of Africa at 30-m spatial resolution[1] (Hengl et al, 2021).

[1] https://www.isda-africa.com/isdasoil/

Chapter 6
Summary

You are reading the work-in-progress Spatial and spatiotemporal interpolation using Ensemble 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-prediction-eml/.

The tutorial above demonstrates how to use Ensemble Machine Learning for predictive mapping going from numeric 2D, to factor and to 3D variables. Have in mind that the examples shown are based on relatively small datasets, but can still become computational if you add even more learners. In principle we do not recommend:

- adding learners that are significantly less accurate than your best learners (i.e. focusing on the top 4–5 best performing learners),
- fitting EML for <50–100 training points,
- fitting EML for spatial interpolation where points are heavily spatially clustered,
- using landmap package with large datasets,

For derivation of prediction error and prediction interval we recommend using the method of Lu and Hardin (2021). This method by default produces three measures of uncertainty:

- Root Mean Square Prediction Error (RMSPE) = the estimated conditional mean squared prediction errors of the random forest predictions,
- bias = the estimated conditional biases of the random forest predictions,
- lower and upper bounds / prediction intervals for a given probability e.g. 0.05 for the 95% probability interval,

You can also follow an introduction to Ensemble Machine Learning from the Open Data Science Europe workshop video recordings[1].

Please note that the mlr package is discontinued, so some of the example above might become unstable with time. We are working on migrating the code in the landmap package to make the `train.spLearner` function work with the new mlr3 package[2].

[1] https://av.tib.eu/series/1146/opendatascience+europe+workshop+2021

[2] https://mlr3.mlr-org.com/

If you have a dataset that you have used to test Ensemble Machine Learning, please come back to us and share your experiences by posting an issue[3] and/or providing a screenshot of your results.

[3] https://github.com/Envirometrix/landmap/issues

Chapter 7
References

References

Appel M, Pebesma E (2019) On-demand processing of data cubes from satellite image collections with the gdalcubes library. Data 4(3):92, DOI 10.3390/data4030092

Bischl B, Lang M, Kotthoff L, Schiffner J, Richter J, Studerus E, Casalicchio G, Jones ZM (2016) mlr: Machine Learning in R. The Journal of Machine Learning Research 17(1):5938–5942, URL https://jmlr.org/papers/v17/15-066.html

Bivand R, Pebesma E, Rubio V (2013) Applied Spatial Data Analysis with R, 2nd edn. Use R Series, Springer, Heidelberg

Bonannella C, Hengl T, Heisig J, Parente L, Wright MN, Herold M, de Bruin S (2022?) Forest tree species distribution for Europe 2000-2020: mapping potential and realized distributions using spatiotemporal Machine Learning. PeerJ DOI 10.21203/rs.3.rs-1252972/v1

Diggle PJ, Ribeiro Jr PJ (2007) Model-based Geostatistics. Springer Series in Statistics, Springer

Erwig M, Gu RH, Schneider M, Vazirgiannis M (1999) Spatio-temporal data types: An approach to modeling and querying moving objects in databases. GeoInformatica 3(3):269–296

Gasch CK, Hengl T, Gräler B, Meyer H, Magney TS, Brown DJ (2015) Spatio-temporal interpolation of soil water, temperature, and electrical conductivity in 3D+ T: The Cook Agronomy Farm data set. Spatial Statistics 14:70–90, DOI 10.1016/j.spasta.2015.04.001

Hengl T, MacMillan R (2019) Predictive soil mapping with R. OpenGeoHub Foundation, Wageningen, URL https://soilmapper.org

Hengl T, Heuvelink G, Perčec-Tadić M, Pebesma E (2012) Spatio-temporal prediction of daily temperatures using time-series of MODIS LST images. Theoretical and applied climatology 107(1):265–277, DOI 10.1007/s00704-011-0464-2

Hengl T, Roudier P, Beaudette D, Pebesma E (2015) plotKML: Scientific visualization of spatio-temporal data. Journal of Statistical Software 63(5):1–25, DOI 10.18637/jss.v063.i05

Hengl T, Nussbaum M, Wright M, Heuvelink G, Gräler B (2018) Random forest as a generic framework for predictive modeling of spatial and spatio-temporal variables. PeerJ 6:e5518, DOI 10.7717/peerj.5518

Hengl T, Miller MA, Križan J, Shepherd KD, Sila A, Kilibarda M, Antonijević O, Glušica L, Dobermann A, Haefele SM, McGrath SP, Acquah GE, Collinson J, Parente L, Sheykhmousa M, Saito K, Johnson JM, Chamberlin J, Silatsa FB, Yemefack M, Wendt J, MacMillan RA, Wheeler I, Crouch J (2021) African soil properties and nutrients mapped at 30 m spatial resolution using two-scale ensemble machine learning. Scientific Reports 11(1):1–18, DOI 10.1038/s41598-021-85639-y

Hijmans RJ (2019) Spatial data in R. United States: GFC for the Innovation Lab for Collaborative Research on Sustainable Intensification URL https://rspatial.org/

Kilibarda M, Protić D (2019) Introduction to geovisualization and web cartography. University of Belgrade, Faculty of Civil Engineering, Belgrade, Serbia, URL http://osgl.grf.bg.ac.rs/books/gvvk-en/

Kilibarda M, Hengl T, Heuvelink GB, Gräler B, Pebesma E, Perčec Tadić M, Bajat B (2014) Spatio-temporal interpolation of daily temperatures for global land areas at 1 km resolution. Journal of Geophysical Research: Atmospheres 119(5):2294–2313, DOI 10.1002/2013JD020803

Lamigueiro O (2014) Displaying Time Series, Spatial, and Space-Time Data with R. Chapman & Hall/CRC The R Series, CRC Press

Lovelace R, Nowosad J, Muenchow J (2019) Geocomputation with R. Chapman & Hall/CRC The R Series, CRC Press

Lu B, Hardin J (2021) A unified framework for random forest prediction error estimation. Journal of Machine Learning Research 22(8):1–41, URL http://jmlr.org/papers/v22/18-558.html

Malone B, McBratney A, Minasny B, Laslett G (2009) Mapping continuous depth functions of soil carbon storage and available water capacity. Geoderma 154(1-2):138–152, DOI 10.1016/j.geoderma.2009.10.007

Marcott SA, Shakun JD, Clark PU, Mix AC (2013) A reconstruction of regional and global temperature for the past 11,300 years. Science 339(6124):1198–1201, DOI 10.1126/science.1228026

McBratney A (1998) Some considerations on methods for spatially aggregating and disaggregating soil information. Nutrient Cycling in Agroecosystems 50:51–62

Meinshausen N (2006) Quantile regression forests. Journal of Machine Learning Research 7(Jun):983–999

Meyer H, Pebesma E (2021) Predicting into unknown space? estimating the area of applicability of spatial prediction models. Methods in Ecology and Evolution 12(9):1620–1633, DOI 10.1111/2041-210X.13650

Mitas L, Mitasova H (1999) Spatial interpolation. In: Longley P, Goodchild MF, Maguire DJ, Rhind DW (eds) Geographical Information Systems: Principles, Techniques, Management and Applications, vol 1, Wiley, pp 481–492

Møller AB, Beucher AM, Pouladi N, Greve MH (2020) Oblique geographic coordinates as covariates for digital soil mapping. SOIL 6(2):269–289

Molnar C (2020) Interpretable Machine Learning. Lulu.com, URL https://christophm.github.io/interpretable-ml-book/

Nelson A, Weiss DJ, van Etten J, Cattaneo A, McMenomy TS, Koo J (2019) A suite of global accessibility indicators. Scientific data 6(1):1–9, DOI 10.1038/s41597-019-0265-5

Pebesma E (2012) spacetime: Spatio-temporal data in R. Journal of statistical software 51(7):1–30, DOI 10.18637/jss.v051.i07

Pebesma E, Cornford D, Dubois G, Heuvelink GB, Hristopulos D, Pilz J, Stöhlker U, Morin G, Skøien JO (2011) INTAMAP: the design and implementation of an interoperable automated interpolation web service. Computers & Geosciences 37(3):343–352, DOI 10.1016/j.cageo.2010.03.019

Polley EC, van der Laan MJ (2010) Super Learner In Prediction. Working Paper Series. Working Paper 266, U.C. Berkeley Division of Biostatistics, URL https://biostats.bepress.com/ucbbiostat/paper266

Ramcharan A, Hengl T, Nauman T, Brungard C, Waltman S, Wills S, Thompson J (2018) Soil property and class maps of the conterminous united states at 100-meter spatial resolution. Soil Science Society of America Journal 82(1):186–201, DOI 10.2136/sssaj2017.04.0122

Roberts DR, Bahn V, Ciuti S, Boyce MS, Elith J, Guillera-Arroita G, Hauenstein S, Lahoz-Monfort JJ, Schröder B, Thuiller W (2017) Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. Ecography 40(8):913–929, DOI 10.1111/ecog.02881

Rudmin JW (2010) Calculating the exact pooled variance. arXiv preprint 1007.1012

Sekulić A, Kilibarda M, Heuvelink G, Nikolić M, Bajat B (2020) Random Forest Spatial Interpolation. Remote Sensing 12(10):1687, DOI 10.3390/rs12101687

Seni G, Elder J (2010) Ensemble Methods in Data Mining: Improving Accuracy Through Combining Predictions. Synthesis lectures on data mining and knowledge discovery, Morgan & Claypool Publishers

Smith DBC, Woodruff WF, Solano LG, Ellefsen F, Karl J (2014) Geochemical and mineralogical maps for soils of the conterminous United States. USGS Geology, Energy, and Minerals Science Center, Denver, CO, URL https://pubs.usgs.gov/ds/801/

Sothe C, Gonsamo A, Arabian J, Snider J (2022) Large scale mapping of soil organic carbon concentration with 3d machine learning and satellite observations. Geoderma 405:115,402, DOI 10.1016/j.geoderma.2021.115402

Wikle C, Zammit-Mangion A, Cressie N (2019) Spatio-Temporal Statistics with R. Chapman & Hall/CRC The R Series, CRC Press

Witjes M, Parente L, van Diemen CJ, Hengl T, Landa M, Brodsky L, Halounova L, Krizan J, Antonic L, Ilie CM (2022?) A spatiotemporal ensemble machine learning framework for generating land use/land cover time-series maps for Europe (2000–2019) based on LUCAS, CORINE and GLAD Landsat. PeerJ DOI 10.21203/rs.3.rs-561383/v2

Wright MN, Ziegler A (2017) ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. Journal of Statistical Software 77(1):1–17, DOI 10.18637/jss.v077.i01

Zhang C, Ma Y (2012) Ensemble Machine Learning: Methods and Applications. Springer New York