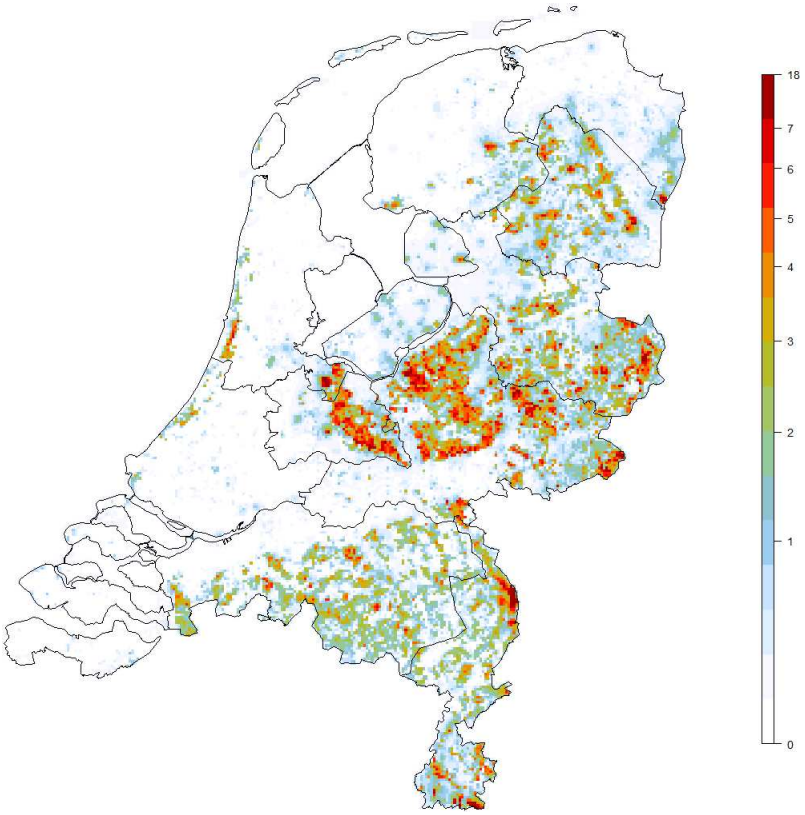
SDMaps: an R package for the analysis of species abundance and distribution data
Extended Manual

Christian Kampichler, Caspar Hallmann & Henk Sierdsema

January 9, 2020

Sovon Dutch Centre for Field Ornithology, Natuurplaza (gebouw Mercator 3), Toernooiveld 1, 6525 ED Nijmegen, The Netherlands



Contents

Abstract

`SDMaps` is an `R` package to implement spatial modelling of species distribution and abundance data. It is particularly designed for integrating the entire workflow from data preparation and modelling observations to making predictions for entire areas and map output for a large number of species in as few steps as possible. In this manual, all necessary information needed to use `SDMaps` is provided, including a description of the programme, basic principles of underlying methodologies, installation and usage guides, and a number of examples to demonstrate the use of `SDMaps` in `R`. It comes along with a tutorial that aims to introduce the user to `SDMaps` by working through a series of increasingly complex modellings tasks (*Remark: not yet available!*). This manual accompanies `SDMaps version 0.15-2`.

Part I
Introduction

1 What is `SDMaps`?

1.1 Aims

`SDMaps` is a package for `R` (R Development Core Team, 2015) to implement (spatio-temporal) modelling of species counts, presence-absence records and species assemblages. `SDMaps` has been developed to facilitate the automatic generation of distribution and abundance maps, based on input sources originating from e.g. standardised monitoring schemes or citizen science projects.

Essentially, `SDMaps` consists of a set of wrapper functions to implement otherwise common statistical procedures within the statistical language and environment `R`, hence can potentially take advantage of the full spectrum of statistical and non-statistical computing capabilities of `R`. The main strength of `SDMaps` is the coupling of spatially arranged biotic and abiotic information, essentially map information, with spatially structured species observations, hence enables the establishment of species-environment relationships that are vital for predicting spatial distributions, abundances and broad trends. A number of models are available for (batch) modelling either species' (relative) abundance (counts of species at particular locations), distribution (from counts or presence absence data), and in case of a multispecies datasets (abundance or presence-absence) species-richness (*not yet implemented!*). Additionally, `SDMaps` allows also to model and map normally-distributed data (e.g. trend indices) or densities (continuous positive values).

1.2 Modelling techniques used by SDMaps

Models facilitated in SDMaps are based on:

1. Generalized linear models (GLM´s) with Poisson or Binomial error structures, extendable with mixed effects, serial correlation, overdispersion and smooth terms. Mixture distribution models (often termed hurdle models: Mullahy, 1986, Zeileis et al., 2008b) and multivariate adaptive regression splines (MARS: Friedman, 1991) models can be implemented as well.

2. Tree-based models such as boosted regression trees[Schapire, 2001, Elith et al., 2008], random forests [Breiman, 2001, Cutler et al., 2007], quantile regression forests [Meinshausen, 2006], conditional inference trees and forests [Hothorn et al., 2006b,a], evolutionary learning of globally optimal trees [Grubinger et al., 2014] and model-based recursive partitioning [Zeileis et al., 2008a].

3. Artificial neural networks [Gallant, 1993, Ripley, 2008].

Spatial correlation correction in this models is accounted for by spatial interpolation (Kriging, Inverse Distance Weighted interpolation or Thin Plate Spline interpolation) of the residuals [Hengl et al., 2009], by the use of interpolated residuals as an autocovariate [Crase et al., 2012], or by incorporating geographical proximity effects into the prediction process by using buffer distances from observation points as explanatory variables Hengl et al. [2017].

Additionally, `SDMaps` facilitates the analysis of presence only datasets as well by modelling presence only information combined with automatically generated pseudoabsences (see Hengl et al., 2009). `SDMaps` depends on a number of other `R`-libraries including `rgdal`, `gstat`, `automap`, `lme4`, `MASS`, `pscl`, `raster` (partly), `spatstat`, `sp`, `randomForest`, `quantregForest` and dependencies thereof (see an overview in section 21). Much of the options and functionalities provided in functions of these packages is also directly accessible in `SDMaps`. Besides regression analyses, `SDMaps` also permits purely spatial data interpolation (Kriging or Inverse Distance Weighted Interpolation).

1.3 Intentions of this manual

This manual is intended to provide all necessary information needed to run `SDMaps` in `R`. It is not intended to review all methods employed within `SDMaps`, but where necessary the relevant literature is provided for the user to inform himself about the underlying assumptions and methodologies. Some basic theory as well as a basic descriptions of the models are nevertheless presented in section 2. Section 3 outlines installation procedures in order to successfully set `SDMaps` up for running. It is recommended that this section is read carefully. Section II describes how your data must be prepared and organized to be modelled with `SDMaps`. In section III we present the two core functions of `SDMaps` called `data2SDMaps()` and `SDMaps()` and various functions with specific functionalities: `dataExploration()`, `SDMapsSummary()`, `SDMapsInterpol()`, `SDMapsMakeMaps()`, `SDMapsPostHocPlots()`, `SDMapsApplyModel()`, `SDMapsCompareMaps()` and `SDMapsEnsemble()`. Section IV consists of some additional scripts that might be useful for special tasks during spatial modelling as well as some hints on troubleshooting.

We advise users new to `SDMaps` to use the tutorial which is also part of the `SDMaps` package (*Remark: not yet available!*). It leads through a series of increasingly complex modellings tasks thus familiarizing the user with the manifold possibilities of `SDMaps`.

1.4 Important remarks

`SDMaps` saves its output on the disk! As you will see further down this manual, the output of `SDMaps` can be very bulky, consisting of processed data, trained statistical models, intermediate and final maps, diagnostic figures etc. In contrast to many other R packages, `SDMaps` thus does not display its manifold products in the workspace (although there are exceptions) but stores all output as files on the hard disk.

Consult the manuals of the used R packages! As mentioned above, `SDMaps` uses many R packages. It is highly advisable to consult the manuals of these packages, most of all those related to the used modelling techniques

2 Model descriptions and basic statistical theory

Generalized linear models are the basis of most modelling practices in `SDMaps`. More approaches exist, such as Bayesian methods (WINBUGS). In the future `SDMaps` may be extended to include these as well. At the moment however the time to implement such models is huge, especially for large dataset originating from national-wide monitoring schemes, for which the program is mainly intended. Therefore, to meet time-demanding (relatively) fast generation of distribution maps we have restricted the implementation to simpler models.

The following sub-sections include some basic statistical concepts in modelling species observations. It is recommended for beginners to read through them and to get a hand on the relevant literature referred to at the various subsections. For statistically experienced users most of these concepts will appear too elementary, so that they probably would prefer moving to section 3 and beyond IIIII, installing and running `SDMaps`.

2.1 Generalized Linear Models (GLM)

As noted above, GLM form the basis of many modelling practices in `SDMaps`. An exception is warranted for presence only datasets, for which the non GLM related Ecological niche factor analysis (ENFA: Hirzel et al., 2002) and Maximum entropy modelling [Grandy and Schick, 1990] can be implemented (see presence-only data models below).

Generalized linear models are a generalization of the common Gaussian linear model (or general linear model) to allow for a broader class of response error distributions such as the Binomial and Poisson families, to which our observations usually belong. In particular, count data originating from yearly monitoring schemes are best modelled with Poisson error distributions due to their integer and non-negative nature. Similarly, presence absence data consisting of either 1's or 0's are best modelled by logistic regression with Binomial error distribution. Useful references to obtain a good understanding of implementing GLM's and their assumptions are McCullagh and Nelder [1989] and Dobson [2002]. For raw inplementation of GLM's in program R see VR and Crawley.

Briefly, the GLM consists of a response vector and a link function that relates a set of covariate values to the response.

$$f(\hat{y}) = a + \beta_j z_j$$

where $\hat{y}$ the expected value of the reponse, β_j the coefficient for the j-th covariate value z_j, and $f(.)$ the link function, in particular the logit link

$$f_{logit}(p) = log(\frac{p}{1-p})$$

for logistic regression of binomial data (in our case presence – absence data of species at particular locations), and log link

$$f_{logit}(p) = log(p)$$

for count data (counts of species at particular locations). The covariate values zj can be of both continuous as well as categorical nature, and represent the value of the j-th covariate z at location $x.y$ where observation $y_{x.y}$ has been made. A number of assumptions are implied in using the above models:

1. correct modelling specification,
2. observations are assumed independent, after including all relevant covariates.

The estimation of covariate coefficient is performed by maximum likelihood.

2.2 GLM extensions

Overdispersion

Both Poisson as well as Binomial GLM's assume a fixed, known variance related somehow to the mean of the distribution. In specific, the Poisson distribution assumes that the variance equals the mean, while in

the Binomial distribution and for a given probability p, the variance equals $p * (1 - p)$. In many datasets however the observed variance tends to be larger than the theoretical one, a condition coined overdispersion. The GLM's functions in program R (and also SDMaps) allow relaxing the distribution-imposed variance assumptions by the use of quasi-families. This implies the estimation of an additional parameter often referred to as overdispersion parameter or variance inflation factor. Note that overdispersion cannot be estimated for binary (0,1) binomial models. This is because likelihood of the saturated model is by definition 1. Hence We will not go into the deep here, but it suffices to know that it is

Joint models

Joint models, also termed hurdle models, employed on `SDMaps` through library `pscl` [Kleiber and Zeileis, 2008, Zeileis et al., 2008b]. Their aim is to fit to the presence-absence part of the data with a binomial distribution and the count data as a zero-truncated poisson model. The benefit of this approach is the circumvention of the distributional assumptions of the Poisson density for datasets with excess zeroes. Intuitively one may find two sources of zeroes in species abundance data. Namely absence because the habitat of the location is unsuitable for the species to live in, and secondly absence because it simply was not observed, even though the habitat is habitable for the species.

Multivariate Adaptive Regression Splines (MARS)

MARS are a method developed by Friedman [1991]. In brief, this method combines GLM and generalized additive models (GAM) resulting in multi break point models called multivariate adaptive regression splines. The method knows automatic term selection, and joint multispecies modelling. The latter is especially interesting for modelling the distribution of rare species together with the more common ones.

Smooth terms

`SDMaps` also allows for smooth terms in the model specification. This is accomplished though generalized additive models (GAM), en extension to GLM to accommodate smooth non-linear responses of observations to specific covariates. In `SDMaps`, smoothing could be for YEAR effect, x and y coordinates, or any other continuous covariate.

Serial correlation

Serial correlation is another form of dependency that can be modelled explicitly in `SDMaps` by the use of Generalized Estimating Equation (GEE) models. Serial correlation can be expected to arise in species counts records originating from long term monitoring data. In such dataset, specific locations are visited year after year (or any other time step) so that the assumption of independence between observations cannot not be met. To picture it, the numbers we count at time t on location x,y are to some extend related to the numbers we have counted at the same location at time t–1, and to some lesser extend to the numbers we counted at t–2, and so on. The term *serial* originates from time-series analysis and points at the inherited time structure in the data. In `SDMaps`, employing a serial correlation structure is accomplished by the use of package `geeglm`. Generalized Estimating Equations can be viewed as an intermediate step between Generalized Linear Models and Generalized Linear Mixed Models, as it does not include random effects for group levels, but does extend the GLM with a specific correlation structure.

2.3 Tree-based models

While GLMs and their extensions assume a stochastic data model whose parameters have to be estimated from the data, tree-based methods follow a completely different approach: They do not assume any data model but use an algorithm that operates on the independent variables ("predictors") to predict the dependent variable ("response"). Briefly, tree-based models are based on the idea that the relationship between predictors and response is not constant over the entire range of possible variable values, but can be approximated in smaller subdomains. Tree-based models are thus constructed by splitting the dataset into subsets based on the minimization of a variance criterion for the subsets. This procedure is iteratively repeated for each subset

and results in a tree-like structure where each branch is defined by a certain range of values of the independent variables. The endpoints of the branches, the "leaves", classify the observations into given categories when the response is a categorical ("decision tree") variable or show average values of the dependent variable when the response is a continuous variable ("regression tree")[Breiman et al., 1984]. Since their publication, researchers involved in machine learning and data mining have further extended these comparably simple tree-based models by adapting various approaches (e.g., boosting and ensemble modelling) and have developed methods that outcompete the original trees in terms of modelling precision and predictive power. In `SDMaps` only those more sophisticated modelling techniques are implemented.

2.4 Extensions of tree-based models

Boosted regression trees

This modelling approach is based on the gradient boosting technique developed by Friedman [2001]. It uses boosting to combine large numbers of relatively simple tree models adaptively, to optimize predictive performance. Originally aimed as a machine learning technique for regression problems, it can also be used for classification. Boosted regression trees are also known as Generalized Boosting Models and we use the abbreviations BRT and GBM interchangeably. The paper of Elith et al. [2008] gives an excellent introduction of how to use BRT in ecological and biogeographical research.

Random forests

Random forest [Breiman, 2001] is a classification and regression method based on the aggregation of a large number of decision or regression trees. It does not use the entire data set for tree construction, but for each tree a bootstrap sample (with replacement) is drawn from the original data set. At each splitting a the splitting predictor is selected from a randomly selected subset of all predictors. Finally, all trees together vote for the the class (in the case of classification) or calculate their mean prediction (in the case of regression). Cutler et al. [2007] describe the application of random forest in ecology. An overview over the current state of the development of random forest related techniques is given by Boulesteix et al. [2012]. Wright and Ziegler [2017] presented a fast implementation of random forests called *ranger* that is also implemented in `SDMaps`.

Quantile regression forests

Quantile regression forests [Meinshausen, 2006] are a generalisation of random forests. They provide information about the full conditional distribution of the response variable–not only about the conditional mean like random forests–thus conditional quantiles can be inferred. This is useful for the calculation of prediction intervals in order to evaluate the reliability of single grid cell predictions.

Conditional inference trees and conditional inference forests

Like classic regression and classification trees, conditional inference trees estimate a regression relationship by binary recursive partitioning but do this in a conditional inference framework [Hothorn et al., 2006b,a]. First, the global null hypothesis of independence between any of the covariates and the response is tested. If this hypothesis cannot be rejected, the analysis stops. If a significant relationship between covariates and the response can be detected, then the input variable with strongest association to the response is selected. Second, a binary binary split in the selected input variable is implemented. Both steps are recursively repeated for the data subsets. Like classic regression and classification trees, they can be aggregated to an ensemble method, the so-called conditional inference forests [Strobl et al., 2007]. They differ, however, from random forests not only in the learning algorithm of the single trees, but also in the aggregation approach: they average observation weights extracted from each of the trees and not by averaging predictions directly.

Evolutionary learning of globally optimal trees

Whereas conventional technique optimize the single splits while growing classification and regression trees, Grubinger et al. [2014] developed the construction of *globally* optimal trees by using evolutionary algorithms. In this approach, an initial set of trees is made with random split rules in their root nodes. Subsequently,

the structure of these trees is modified by mutation and crossover operators. A survivor selection mechanism selects the best candidate models for the next iteration of this procedure. The mean quality of the population of trees becomes better and better in this evolutionary process which comes to an end as soon as the best trees do not improve further.

Model based partitioning

Another tree-based approach is model-based recursive partitioning. Instead of constructing a tree with single values in its leaves, here, a tree is built with fitted models in its leaves [Zeileis et al., 2008a]. The used algorithm consists of four basic steps: (1) a parametric model is fitted to the data; (2) parameter instability is tested over a set of partitioning variables; (3) in case there is some overall parameter instability detected, the model is split with respect to the variable associated with the highest instability; (4) this procedure is repeated in each of the daughter nodes.

2.5 Artificial Neural Networks

Briefly, Artificial Neural Networks (NNs) consist of a number of interconnected computing units, which are termed cells or nodes. In the most popular type of NN, the backpropagation network, the nodes are typically organized in a layered structure: an input layer, whose nodes represent the independent variables; an intermediate layer (or more) of so-called hidden nodes; and an output layer, whose nodes represent the dependent variables. All nodes are unidirectionally joined and each connection has a numerical weight. Along these connections, the input values are propagated through the network. A NN can be trained by providing it with training patterns, e.g. patterns with known outputs for a given input or–in the context of distribution modelling–known presence/absence or density at a location with for a set of environmental covariates. The NN begins with a randomly chosen set of connection weights, compares the calculated output of the first training pattern with the desired output and, by applying the backpropagation algorithm, propagates the error backwards through the net. In the end of a backpropagation step, the connection weights are slightly altered and the output approaches by a small step the desired output. NNs are universal approximators since, by providing a suciently complex network iteratively with a large number of training patterns, it is able to model any differentiable relationship [Hornik et al., 1989, Warner and Misra, 1996]. This is particularly important for ecological modelling since standard statistical methods of relating independent and dependent variables can only insufficiently cope with the nonlinearities of high-dimensional ecological systems.

2.6 Presence-only data models

Ecologists often deal with datasets of opportunistically gathered data, consisting often of raw presence observations, or at best counts of birds, at particular geographical locations. The problem with using this type of data in generating distribution maps is that we have information on presence locations (having seen a bird at a location confirms this) but we are unaware of where the species is absent, as the data usually lacks zero counts, i.e. no birds seen at location x, y. For this kind of data, `SDMaps` uses Maximum entropy modeling (MaxEnt) [Elith et al., 2011, Merow et al., 2013, Phillips et al., 2006] for creating background points with species absences and subsequently applies on of the regression approaches described above.

2.7 Generating Pseudo-absences

A third method to deal with absence of zero counts described in Engler et al. [2004] and Hengl et al. [2009] and implemented in `SDMaps` is a combination of a presence-only model and a regression model. As you all already might have thought of, a regression model for presence-absence data with 1's only as a response cannot be run sensibly. One needs to somehow come up with zeros denoting absences of a species at particular locations. The aim in this approach is to generate by some independent and sophisticated way zero observations. These are then subsequently added to the original presence-only data, and we are set to proceed with modelling the data with the regression models as described above.

The approach undertaken in Hengl et al. [2009] is outlined briefly: one computes a habitat suitability map based on the presence values of the data and one of the two above mentioned Presence-only data models. The habitat suitability map in combination with a distance map (distance from each point on the map to the

nearest presence observation) are used to create a weight map to be used in drawing semi-random positions. That is, locations that are deemed unsuitable by MaxEnt and that are further away from locations where we are certain the species is present, obtain a higher probability to be drawn at random. These positions are commonly termed *pseudo-absences*.

2.8 Dealing with spatial autocorrelation

Spatial autocorrelation is an important issue in species distribution modelling and there exists a variety of methods to cope with it [Dormann et al., 2007]. In `SDMaps` we adopted three approaches. The first is based on the suggestions by Miller [2005] and Hengl et al. [2009]and consists of predicting the regression part, analyzing and interpolating the residuals, and adding them back to the predictions. This regression-kriging procedure delivers realistic maps but has a drawback: the relative contribution of spatial autocorrelation to the modelling success as compared to the contribution of the environmental covariates cannot be determined.

We thus also implemented the RAC (residuals covariate) approach proposed by Crase et al. [2012]. Here, the residuals of the regression models are interpolated, appended to the set of covariates, and the regression is run again. This procedure guarantees an advantage over other approaches to cope with spatial autocorrelation (for example, the autologistic approach), because the explanatory variables are fitted first and have an opportunity to account for spatial autocorrelation. By deriving the autocovariate from model residuals instead directly from the dependent variable (that is, presence-absence or density), only the variance unexplained by the explanatory variables is incorporated. The contribution of spatial autocorrelation to the modelling success (for example, statistical significance in a GLM or variable importance in a random forest) can thus be quantified in the same way as for any other covariate.

As a third possibility the RFsp (random forest for spatial predictions framework) approach developed by Hengl et al. [2017] is implemented in `SDMaps`. Here, geographical proximity effects are incorporated into the prediction process by using buffer distances from observation points as explanatory variables.

Both regression-kriging (adding the interpolated residuals to the regression predictions) and RAC (using the interpolated residuals as autocovariate) methods can be applied to any of the modelling techniques used by `SDMaps`; the RFsp approach (buffer distances as explanatory variables) are recommended only for random forest and related models because a huge number of covariates is generated in this process (one buffer for each observation point).

2.9 Terminology

Statistics (GLM and derivates) and machine learning (tree-based modelling techniques) have different traditions and terminological conventions with the potential to cause considerable confusion for the user [for example, van Iterson et al., 2012]. In this manual, we do not stick to a certain trade language but try to be as simple and comprehensible as possible. Nevertheless, there remain some terms that may confusing. For example, there exist at least five different words for those variables that explain the values of a dependent (also: response) variable: independent variable = explanatory variable = regressor = covariate = predictor = ... In this manual, we use the terms *covariate* and *predictor*.

Some terms that are recurrently used in the manual:

plot data The data characterising the plots (or points) where organisms were observed or counted, including coordinates, number of individuals per species, values of environmental covariates etc.

point predictions The predictions of the model for the locations of the plot data

prediction grid The grid for which the model that was trained based on the plot data will make predictions.

2.10 Abbreviations

Throughout the manual we use the following abbreviations for modelling techniques:

BRT, Boosted regression tree

CTREE, Conditional inference tree

CFOREST, Conditional inference forest

EVTREE, Globally optimal regression tree trained by evolutionary algorithms

GBM, Generalised boosting model

GLM, Generalised linear model

MARS, Multivariate adaptive regression spline

MOB, model based partitioning

NN, Artificial Neural Network

QRF, Quantile regression forest

RANGER, Random forest made with the R-package ranger (faster!)

RF, Random forest made with the R-package randomForest

3 Getting and installing SDMaps

In using `SDMaps` one needs program `R` installed and running. `R` is a free open source statistical software that can be downloaded for GNU/Linux, OS X and Windows at no cost at data.typehttps://cran.r-project.org.

By itself, `R` has already a wide range of statistical and data manipulation capabilities by default. However numerous contributions exist in the form of libraries that extend `R`'s capabilities even further. `SDMaps` is just another library designed to implement some (hopefully) useful tasks, aiding the generation of distribution and abundance maps of species. `SDMaps` does not stand alone; a number of other packages are vital for the implementation of `SDMaps` in `R`. When installing `SDMaps` these are downloaded (if all goes well) and installed, and are loaded each time `SDMaps` is called.

One exception is the installation of program MAXENT, needed for the generation of pseudo-absences. MAXENT is supplied with SDMaps. (*Note*: Use only the MAXENT version supplied with `SDMaps`; the corresponding jar is stored in directory `inst/java/` in the tarball for GNU/Linux users and the zip-file for Windows users, respectively. Other versions will not work!) For Windows users it is important to install program Maxent in a folder to which the pathway does not include spaces, for example not in `C:/Program Files/Maxent` but in `C:/Maxent`. For some reason spaces in pathways are a burden for the communication between the Java-based program Maxent and program `R`. Be sure to have Java installed on your computer, otherwise Maxent will not run. Java can be downloaded and installed from it from http://www.java.com/en/download/manual.jsp. *Note*: The standard installer on the java-website only installs the 32-bit version. If you have a 64-bits computer, you will have to install the 64-bits version of Java manually!

Once R and Java are installed, users of GNU/Linux can download the source code as a tarball from here: `http://s1.sovon.nl/SDMaps/SDMaps_1.15-0.tar.gz` and install SDMaps via

```
> install.packages("[path to the tarball]", repos = NULL, source = TRUE)
```

Windows users can download the binary as `http://s1.sovon.nl/SDMaps/SDMaps_0.15-0.zip`, open R and install it with command

```
> install.packages("[path to the zipfile]", repos = NULL).
```

If any of the libraries required by `SDMaps` is not installed properly, error messages will appear. In such cases the corresponding packages can be downloaded and installed manually by `Packages -> Load package` ... or by typing the command

```
> install.packages("[name of missing package]")
```

Choose some mirror close to you, wait until succeeded and retry installing SDMaps.

A second issue to notice is the size and resolution of datasets. R is known for its broad(est) range of statistical and non-statistical computing capabilities, but less famous for its ability in handling large files. This is mainly due to the fact that when using R all data is loaded and kept in the memory. Too large files may result in program failure to accomplish desired tasks. When proceeding to modelling the data, various local copies of the data are created, increasing even further the change of program failure due to lack of memory. SDMaps is designed to carefully avoid loading, locally copying data and running models at once (especially in batch analysis) by for example creating copies per species (in multispecies datasets) locally on the hard-drive, and saving predictions by year (in multiyear datasets). Nonetheless, the massive datasets becoming increasingly available through long term country wide monitoring schemes might still challenge the memory limits of R. The only solution at the moment if memory issues become a problem is to move to larger platforms.

Part II
Data preparation

Modelling your species observations is generally a two step approach in SDMaps:

1. get your input data in the right format for modelling through function data2SDMaps(), and

2. model your data and generate predictions through function SDMaps().

Additionally, you can create modelling summaries through SDMapsSummary() (*Note*: not yet available for all model techniques and not yet fully standardised) or make maps by spatial interpolation of the data using command SDMapsInterpol().

Having followed all steps in section 3 and successfully installed SDMaps and all dependencies, find out which data format suits your needs (see section 4) based on availability of maps and structure and dimensionality of datasets.

4 Input data

SDMaps is quite flexible concerning the formats of data input and allows the user to structure the data according to his own needs or resources. The many possible ways of presenting data to SDMaps, however, can also cause confusion and it is thus recommended to read this section very carefully.

Four pieces of information are always needed in order to run SDMaps models and create prediction maps:

1. *Information on plot location* (= geographic coordinates) (see section 4.1). This information can–but needs not to–be extended by plot specific covariates and the species observations.

2. *Observations* (see section 4.2). You have to provide the species data (counts, presence-absence records, density or trend indices) related to each plot.

3. *Covariate grid maps* (see section 4.3). You must provide a grid with the environmental variables that will be used as predictors, covering the entire area for which maps have to be made.

4. *Contour shapefile.* (see section 4.4). A shapefile containing national boundaries, large rivers, big cities, or the like, are very convenient if your maps should be easily readable.

Additionally you can provide a *textfile with species codes and names* for making comprehensible filenames and figure titles (see section 4.5).

4.1 Plot location

You always need information on the locations of the observations. This information will be passed to `data2SDMaps()` in the argument `plot.data`. If data has already been loaded in the `R` workspace, plot data should be of class data.frame or a SpatialPointsDataFrame. It minimally includes the columns "plotID", "x", and "y", and optionally may include the counts ("observed"), species code ("Species"), year ("YEAR") and further covariates. Note that these column names are case sensitive, and care should be taken to appropriately specify these names.

Attention: The plot ID's, the species code and all variable names should be *without spaces or hyphens* ("-"), but the use of the underscore ("_") is allowed. This is important so that `SDMaps` will know which plot corresponds to which information, at which spatial location and possibly on which year, and how it should be joined to the observation data.

Attention: Although missing data in the covariates do not pose problems for some modelling techniques, others depend on complete covariate information. `SDMaps` contains an imputation routine for these cases, but the procedure may cause considerably longer computing times. The general advice is, thus, to avoid missing data in the covariates.

The argument `plot.data` can be left empty, in which case a windows menu will open and ask for a file. Alternatively a path can be specified. In both cases it expects a comma, semicolon, space or tab separated csv or txt file. In addition, the `crs` argument (the coordinate reference information) should be provided as well. It allows the user to make predictions and generate maps using another projection than the one used for data input.

4.2 Observations

In most cases, the observations will be counts, presence-absence and/or presence-only data. SDMaps, however, can also deal with normally-distributed data (e.g. trend indices) or continuous positive values (e.g. densities). The observations to be modelled by `SDMaps` are specified by the argument `obs.dir` in function `data2SDMaps()`. When left empty, and if observations have not been supplied directly through `plot.data`, a windows dialog box will popup for the user to point a directory holding the data. The way `SDMaps` looks for data is as follows: If `plot.data` do not contain observations, it checks the arguments supplied to `obs.dir`. If `obs.dir` is empty, a dialog box appears. Alternatively, if a directory is specified, it searches the directory for files with extensions .csv or .txt. Other files are ignored. Note that if both csv and txt files exist in the folder specified, `SDMaps` will probably report an error, as only one of the two file types are allowed to be present in the `obs.dir`. You may provide a file for each species, but you can also have your observations in a single file. In the latter case a column named "Species" is expected (if missing or not specified correctly, `SDMaps` will assume that the file consists of a single species dataset) in order to run a batch analysis. You may specify the single file by either supplying a pathway directly, or by specifying its directory.

4.3 Covariate grid/raster maps

You probably need a set of raster/grid maps for use as covariates and definitely as predictor grids (otherwise you would not be using `SDMaps`). These can be either of ASCII or GeoTIFF format (one file per covariate). The argument `user.dir` specifies the directory in which the maps are stored (either tiff or asc). If any map is of categorical nature, please specify them through option `adjust.covariate`. Note that if one or more covariates are already included in the plot.data, for prediction purposes you will also need the corresponding covariate grid maps. Beware that there should be *no spaces and "-" characters* in the names of the variables and ascii-grids. If you have variable names or ascii-gridnames with spaces or "-", please replace these with points, or even better, underscores ("_"). (*Note*: Use a text editor like Emacs, Total Commander or Freecommander or a stream editor like sed for batch renaming of filenames!)

4.4 Shape-file with contour

Provide a polyline or polygon shape-file with the topography to be shown in the maps, like borders and rivers. The projection (coordinate reference system) has to be the same as the covariate grid maps for the predictions.

4.5 Textfile with species codes and names

During modelling often codes are used instead of full species names (e.g. Euring-code 5320 instead of Black-tailed godwit or *Limosa limosa*). For headers of maps and figures and for filenames, however, it may be convenient to use the species names. In this case provide a textfile called speciesnames.txt that contains two comma-separated columns "code" and "species", for example:

```
code, species
5320, Limosa limosa
5460, Tringa totanus
5610, Arenaria interpres
```

5 Data formats

There are a number of different ways to input your data through `data2SDMaps()`. To some degree the format type may depend on the way data is originally stored, but it may also be determined by the way how models are supplied with covariates, and may avoid memory size limitations within `R`. The formats are:

1. Plot locations, covariates and species observations are included in one single file (see section 5.1).
2. Plot locations and covariates are included in one file, the observations are in a separate file(see section 5.2).
3. One file includes plot locations and possibly covariates, and one or more additional files hold the species observations (see section 5.3).
4. Input data come from programme TRIM [Pannekoek and van Strien, 2005], and one file holds the plot information (see section 5.4).

You may find it also at this point useful to known that `SDMaps` does not do any sophisticated GIS analysis on the covariate maps. `SDMaps` merely makes overlays of plotIDs and extracts the covariates "as is", and at best, checks for consistency of dimensions and resolutions between maps. The same maps are used for modelling and projecting predictions.

In some cases, this simple overlaying may not be sufficient for modelling purposes. In such cases one may provide his covariate values through the plot location information file. In doing so, one still needs covariate maps (grids) for prediction. They must be named exactly as the column names in the plot information file (extension forgiven). To accomplish this, one supplies his `plot.data` with additional named columns (names matching the corresponding grid maps, extension excluded).

5.1 Locations, observations and covariates in one single file

The datafile must be specified in `data2SDMaps()` argument `plot.data` and has to be prepared as a comma separated txt or csv file with columns PlotID, YEAR, Species, observed, x, y, [covariate 1], [covariate 2], ..., [covariate n]. The column headers PlotID, YEAR, Species, observed, x, y are compulsory and must not be changed. The column headers of the covariates have to be chosen according to the names of the corresponding grid maps that will be used for making the prediction and that will be specified in argument `user.dir`. For example, if a map bears the name prop_grass.asc (meaning percentage of the grid cell covered with grassland), then the colum header corresponding column in the data file must be prop_grass, leaving out the extension .asc. An example of the head of such a data file could be the following:

```
plotID, YEAR, Species, observed, x, y, prop_grass, av_rain, [other variables], perc_for
1, 2012, colliv, 3, 46.3, 28.2, 0.34, 1987, [...], 91
1, 2012, strdec, 15, 46.3, 28.2, 0.34, 1987, [...], 91
2, 2012, colliv, 8, 51.0, 29.7, 0.67, 1570, [...], 87
2, 2012, colliv, 11, 51.0, 29.7, 0.67, 1570, [...], 87
```

For explanatory reasons here in a more readable form:

plotID	YEAR	Species	observed	x	y	prop_grass	av_rain	...	perc_for
1	2012	colliv	3	46.3	28.2	0.34	1987	...	91
1	2012	strdec	15	46.3	28.2	0.34	1987	...	91
2	2012	colliv	8	51.0	29.7	0.67	1570	...	87
2	2012	strdec	11	51.0	29.7	0.67	1570	...	87
...	...	...	...	...	...	...	...	...	...

5.2 Locations and covariates in a file, separate observations

The locations and covariates must be provided as a txt or csv file with the columns plotID, x, y, [covariate 1], [covariate 2], ..., [covariate n] and the file specified by `data2SDMaps()` argument `plot.data`. As described above, the column headers plotID, x are compulsory and the headers of the covariate columns must correspond to the filenames of the grid maps that are specified in argument `user.dir`.

Example for file specified in `plot.data`:

plotID	YEAR	x	y	prop_grass	av_rain	...	perc_for
1	2012	46.3	28.2	0.34	1987	...	91
1	2012	46.3	28.2	0.34	1987	...	91
2	2012	51.0	29.7	0.67	1570	...	87
2	2012	51.0	29.7	0.67	1570	...	87
...	...	...	...	...	...	...	...

You must specify `data2SDMaps()` argument `obs.dir` in order to tell `SDMaps` where it can find the observations. It may specify a path to a csv or txt file (when there is a single text file containing observations of all species) or the path to a directory (when there is a separate observations file for each species). Via `data2SDMaps()` argument `user.dir` it must be specified where the grid maps of the covariates can be found.

Observations in a single file require the columns plotID, YEAR, Species and observed (column headers compulsory).

Example:

plotID	YEAR	Species	observed
1	2012	colliv	3
1	2012	strdec	15
2	2012	colliv	8
2	2012	strdec	11
...	...	...	...

If there is a separate file for each species, these files require columns plotID, YEAR and observed (headers compulsory). Their filenames will be used as speciescodes (excluding the extension).

Example: First species ...

plotID	YEAR	observed
1	2012	3
2	2012	8
3	2012	21
4	2012	7
...	...	...

... next species

plotID	YEAR	observed
1	2012	15
2	2012	11
3	2012	12
4	2012	27
...	...	...

5.3 Locations and observations in separate files, no covariates

In `data2SDMaps()` argument `plot.data` you specify where to find the plot location. The corresponding txt or csv file needs the columns plotID, x and y (column headers compulsory).

Example:

plotID	x	y
1	46.3	28.2
1	46.3	28.2
2	51.0	29.7
2	51.0	29.7
...	...	...

In `data2SDMaps()` argument `obs.dir` you tell `SDMaps` where to find the species observations. As mentioned in section 5.2 this may be the path to a single file containing observations of all species or the path to a directory containing separate files, one for each species. See the examples in section 5.2 of how the observation file(s) must be provided.

In `data2SDMaps()` argument `user.dir` you specify where to find the maps with the covariates. As explained in section 5 the covariate values for each plot are determined by an overlay of plotIDs and covariate grid maps.

5.4 Input data from programme TRIM

TRIM [Pannekoek and van Strien, 2005] is a programme for the analysis of time series of counts with missing observations developed by CBS Statistics Netherlands. It can be used to estimate indices and trends and to asses the effects of covariates on these indices and trends.

Part III
Running SDMaps

6 General

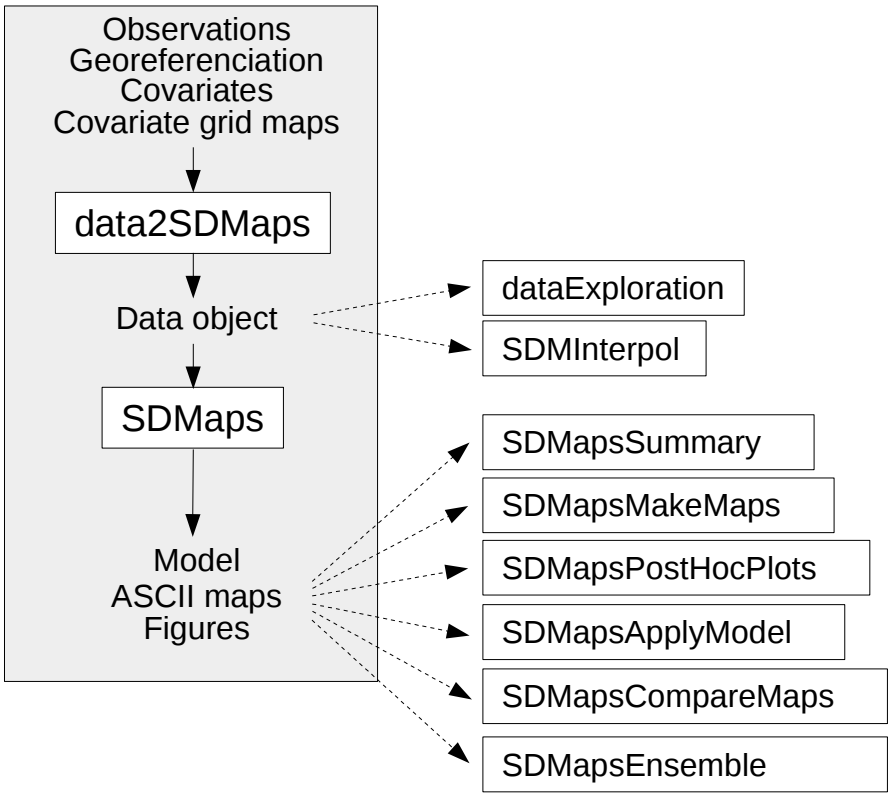
The aim for the development of `SDMaps` was to facilitate the automated analysis of a large number of species and the automated production of a large number of distribution maps. In contrast to other R-packages, most of its functions thus do not return values or objects that can be viewed in the `R` workspace, but store files (maps of the raw data, maps of the modelled distribution, tables with variable importance etc.) on the hard disc.

You can specify any model that can be run in `R`, and accepts arguments formula and data. Additional arguments are passed through `...`, that is: named arguments! Alternatively, you can specify a number of extensions to the GLM, so that `SDMaps` will internally point to the right models to fit. The extensions are serial correlation, estimate for overdispersion, random effects, joint modelling of presence/absence and abundance (for count data only), adaptive regression splines, smooth terms and interpolation of residuals. Some of these extensions—but definitely not all—will also work together. You can also run tree-based models, namely boosted regression trees, random forests, conditional inference trees, conditional inference forests, globally optimal trees, quantile regression forests, and model based partitioning.

A number of example data sets and command scripts are available for `SDMaps` along with a tutorial [Remark: Not yet available!]. Try these first and have a good look at the provided data set before you start with your own data.

`SDMaps` consists of two core functions that comprise the complete workflow from preparing the data for modelling, training a model on the data, making predictions for the entire prediction grid and putting out maps in various formats, either for further processing in a GIS or as plotable figures, while a number of additional functions allow to perform specific tasks (Fig. 1).

Figure 1: Main structure of `SDMaps`: the core functions `data2SDMaps` and `SDMaps` comply the entire workflow, additional functions can do specific tasks.



The `SDMaps` core functions are:

- `data2SDMaps()` - prepares the data and maps for the analysis with SDMaps
- `SDMaps()` - analyses the relationship between observations and environmental covariates, makes predictions for a specified area and generates maps

The additional functions available for specific purposes are:

- `dataExploration()` - makes various diagnostic plots to screen the data for outliers, correlation among covariates etc.
- `SDMapsSummary()` - delivers information on modelling quality, covariate importance etc.
- `SDMapsInterpol()` - makes a spatial interpolation of the observed data
- `SDMapsPostHocPlots()` - displays maps of different species or runs with the same colour codes to make them comparable
- `SDMapsMakeMaps()` - makes maps based on a `SDMaps` model
- `SDMapsApplyModel()` - applies a `SDMaps` model to new covariate data
- `SDMapsCompareMaps()` - compares distribution maps with the structural similarity index
- `SDMapsEnsemble()` - makes ensemble predictions based on two or more `SDMaps` models.

In this chapter the arguments to run these functions are explained. All possible arguments can be retrieved in an R session by means of the function `args()`, for example `args(data2SDMaps)`. The number of options for each function is very large; we suggest to go line for line through the options list when designing a command. We strongly advise using scripted analysis instead of working on the command line (see section 17 Using scripts)!

7 Function `data2SDMaps()`

The function `data2SDMaps()` reads in all necessary data (see section 5 Data formats) and converts them into an object of class "list". This object–we call it the *sdmdata*–contains all data, paths and modelling information specified by the following arguments. The object can directly be used with `SDMaps()` or can be saved in the working directory (see section 17 Using scripts) for later use and loaded into the R working space.

Attention: The sdmdata object should not be moved into another directory because of the path definitions it contains! Using it from within another directory or on another computer would cause the analysis to crash (unless you know how to edit it manually). If it is necessary to repeat the analysis on another computer or somewhere else in the directory structure of a computer, invoke `data2SDMaps()` again at the new position prior to using `SDMaps()`.

7.1 Arguments for function `data2SDMaps()`

plot.data Character. A data-file that includes minimally a plot or record identification number ("plotID"), and x and y coordinates ("x" and "y"). Default is `NULL`. This is usually a pathway to a .csv or .txt file, but can also be a R data.frame or a SpatialPointsDataFrame. In the latter case x and y columns are not expected to be necessarily present. plot.data can include already the observations (in which case the next argument, `obs.dir`, is left to its default) and possibly covariates. The expected columns names should be plotID, x, y, observed and potentially YEAR and Species. Other columns are interpreted as potential covariates (are not excluded) while the above mentioned ones are treated specially. *Note*: Names are case sensitive: x should be x and not Lon, X or Eastings.

obs.dir Character. A data file that includes minimally a plot or record identification number (plotID), observed counts or presences (observed), and possibly YEAR, and Species. Default is `NULL`. `obs.dir` can be left unspecified if `plot.data` already includes observations. Usually, `obs.dir` specifies a pathway to a .csv or .txt file. It can however also be a pathway to a directory which includes separate .csv files for each species. *Attention*: Write plotID, observed and YEAR precisely in this way, respecting lower case and upper case letters!

crs Character. coordinate reference system or 'projection' (crs) of the plot data. One can use function `choose.crs()` to retrieve one from a list and obtain the parameters of the crs.

named Character. the name of the modelling run. Default is `"sdmdata"`.

outdir Character. a pathway that specifies the directory where results and temporary files will be stored

add.zeroes Logical. Should zero-observations be included if no observations exist of a given species in a given plot? Default is `FALSE`.
If specified as `TRUE`, `data2SDMaps()` will add zeroes to the data for missing species/plot information. This is particularly helpful when in standardized monitoring schemes the absence of information corresponds to the observed absence of the species in a given plot (and not just to "not looked for").

generate.zeroes Logical. Suitable for presence only data, where the aim is to generate zeroes at (presumable) locations of low suitability where a given species is absent. Default is `FALSE`.
This function generates so-called pseudoabsence data. Used in concordance with `gen.zer.options`. *Attention:* You cannot use both `add.zeroes` and `generate.zeroes` at the same time!

gen.zer.options A list with specifications on how to generate and add the zeroes. Use the syntax `gen.zer.options=list(argum = [ ], argument2 = [ ], ...)`. It consists of the following arguments:

yearspecific Logical. Should zeroes be generated for each year separately? Default is `FALSE`. A column names YEAR should be present in the plot.data.

n Numeric. Number of zeroes to generate. If left unspecified, it will equal the number of presences. Can also be a vector of numbers of length equal to the number of years in the dataset.

add.to.csv Logical. Whether or not to add the generated zeroes to the dataset now, or leave it at the modelling stage. Default is `FALSE`.
Choose the default if the dataset is large.

tdir Character. directory to save the modelling results of generating zeroes. If not specified the directory defined by argument `outdir` will be used.

maxent.options There are more than 50 options in programme Maxent; their values are predefined in `SDMaps` and need not to be specified by the user. Only one two of them are of relevance for the generation of zeroes, `path2maxent` and `mb`. In any case the option `path2maxent` must be specified with the path to the executable file maxent.jar including the its filename, for example, `path2maxent = "C:/SDMaps/Maxent/maxent.jar"`. Additionally, argument `mb` may be used to determine the heap size used by Java, expressed in MB, for example by `mb = 500`. This is not mandatory, however, because `SDMaps` sets the heap size automatically to 75 % of free physical memory when the argument is not used. Both options are used to invoke function `maxentSDMOptions`(). Use the syntax `maxent.options = maxentSDMOptions(path2maxent=[ ], mb=[ ])`. Default is the function `.maxentSDMOptions()`.

use.maps Logical. Use standard covariable maps delivered with `SDMaps`? Currently only default (`FALSE`) can be used.

user.maps Logical. Use user-defined covariable maps? Currently only default (`TRUE`) can be used.

user.dir Character. Path to a directory holding the covariate grid maps. Maps should be one of ascii or tif files, assumed to be all defined on the same coordinate reference system, have same resolution and extent.

user.crs Character. The coordinate reference information of the plotdata coordinates. One *must* be supplied one at the moment to ensure proper running of data input. It can be used when plot.data and map data are defined in different coordinate systems.

user.all.question Logical. Provide a list of objects in the directory specified in `user.dir` to choose from to add to the data? When the dafult (`FALSE`) is chosen, all variable grids present in directory `user.dir` will be added.

categ.covariates Character. Used to specify categorical covariates. Default is `NULL` (no categorical covariates).

subset2plotdata Logical. Trim covariate map to the geographical extent of the plotting data? Default is `FALSE`.

TRIM.source Logical. Are the data in any format that is supported by TRIM (both input out output files)? Default is `FALSE`.

remove.RAC Logical. When the residual autocovariate (RAC) approach [Crase et al., 2012] is used to handle spatial autocorrelation, a grid file representing the RAC is stored in the directory with the covariable maps (defined by argument `user.dir`). The RAC is species-specific since it is calculated during a preliminary modelling run for a given species. It will be used by making new sdmdata–this time including the RAC to the covariables–and another modelling run. However, when sdmdata must be generated for another species based on the same covariables, the directory has to be *clean*, that is, no RAC grid generated during an earlier modelling run must be present. When `remove.RAC` is set to `TRUE`, the RAC grid is automatically deleted from the directory with the covariable maps after an execution of `data2SDMaps()`, so for the generation of sdmdata for the next species the directory is clean. Default is `FALSE`.

7.2 Products delivered by `data2SDMaps()`

[sdmdata].RDATA This is a binary file that contains an `R`-object of classes "sdmdata" and "list" with the name you specified with option `named` (Default is `sdmdata`). It is saved in directory `[outdir]/OBSERVATIONS/Temp1/`. The object stores general information about a run of `data2SDMaps()` and consists of the following elements that can be viewed with the command `[sdmdata]$[element]`:

data A "SpatialPointsDataFrame" of the plotdata that contains a dataframe with the covariates and their geographic reference

pred.data A "SpatialGridDataFrame" representing the grid topology (geographical bounding box and cell size) and the covariate values in the grid cells

data.name Character string that store the name of the object itself (default: `sdmdata`).

files Character string that stores the path to the results directory where the object is stored

Userdir Character string that stores the path to the grid maps of the covariates

nyears and years Number of years and year dates in the plot data. When no years are provided, `data2SDMaps()` assumes that all data come from the same year with year date 1 (`nyears = 1`, `years = 1`).

n.unique.plots The number of unique plots in the plot data

specnames Character strings with namesof the files where the species observations are stored, consisting of the code used for each species and the extension DAT

not_run Character vector of species which produced an error and will not be included to the analysis

not_run_errors Error messages for the species which produced an error

missing.count.data Number of plot ID's from the observation file that are missing in the plot data

missing.plot.data Number of plot ID's from the plot data that are missing in the observation file

call Call of `data2SDMaps()` in which all of the specified arguments are specified by their full names

attrs Specification of argument `add.zeroes`

8 Function `dataExploration()`

Before running a SDMaps analysis it might be useful to screen the data for outliers, correleated covariates etc. as was suggested by Zuur et al. [2010]. The function dataExploration() makes various diagnostic figures of the data prepared for a SDMaps run. It plots a map with the locations of the observations, makes bivariate figures and a correlation analysis for all pairs of covariates, plots observed values versus all covariates and and makes a Cleveland dotplot for identifying outliers.

8.1 Arguments for function `dataExploration()`

SDMdata The data object generated with function `data2SDMaps()` (see section 7) in the workspace or the path to a `SDMaps` data object saved on the hard disk. The data are by default saved under the name sdmdata.RDATA in the directory `[outdir]/OBSERVATIONS/Temp1/` (directory `outdir` was specified in function `data2SDMaps()`).

spec.subs Character. Vector of species names or codes for which the data have to be explored (e.g. `spec.subs = c(5300, 12000)` or `spec.subs = c("Fox", "Badger")`).

outdir Path to the directory where the results of the data exploration have to be stored. If the specified directory does not exist yet, it will be generated.

data.type Character. One of "presence", "count", "density", "normal", "trendindex". See more information on the same argument in function `SDMaps()` (section 9). The specification of the data type is necessary for plotting the observations correctly (see next argument).

covar.plots Logical. Should plots of the covariates be made? The resulting plot shows the distributions of the covariates for the plot data and the prediction grid. Default is `FALSE`.

plot.location Logical. Plot a map with the observations? Default is `FALSE`.

path.to.contour Character. Specify the path to the shape-file (see section 4.4) including its file name. If no path is specified, `SDMaps` looks for a file named contour.shp in the working directory. If this file is not available neither, then a warning is displayed and the location plot cannot be drawn.

path.to.speciesnames.file Character. Path to the directory where file `speciesnames.txt` is stored. The file permits the use of full species names in file names and graphics instead of codes (e.g. EURING numbers). It must consist of two comma-separated columns named 'code' and 'species', for example:

```
code, species
5320, Limosa limosa
5460, Tringa totanus
5610, Arenaria interpres
```

pairwise.covars Logical. Make pairwise covariate plots and calculate correlation coefficients between covariates? Default is `FALSE`.

variance.inflation.factors Logical. Variance inflation factors (VIF) are used to identify variables with collinearity problems in two ways [Naimi et al., 2014]. In the first, VIF are calculated for all covariates and the covariate with the highest VIF (if >3) is removed. The procedure is iteratively repeated until no variables with a VIF>3 are left. In the second, the pair of variables which has the maximum linear correlation (greater than `min.corr`) is identified, and the one of them with the larger VIF is excluded. The procedure is repeated until there is no variable remaining with a corrrelation coefficient larger than `min.corr`. Default is `FALSE`.

min.corr Numeric. Threshold for correlation coefficient to be used for a table with correlated covariates (see section 8.2) and for the variance inflation factor approach. Default is 0.75.

obs.vs.covars Logical. Plot observed values against all single covariates? Default is `FALSE`

outlier.detection Logical. Make a Cleveland dotplot to identify outliers in the data? Default is `FALSE`.

all Logical. If set to `TRUE` all plots will be made. Default is `FALSE`.

dec Character. Tables stored by `dataExploration()` can conveniently be inspected with a spread sheet. Specify if you use "`.`" or "`,`" as decimal symbol. Default is "`.`".

8.2 Products delivered by function dataExploration()

Dependent on the chosen data exploration plots, the files `covariate_plots.pdf`, `Locations_lancol_[species code].png`, `Outlier_detection_[species code].pdf`, `Observed_vs_covars_2_[species code].pdf`, `Pairwise_covar code].png`, `Variance_Inflation_Factor_[species code].txt` and `Observed_vs_covars_[species code].pdf` are saved in the specified output directory. When the `sdmdata` contain more than ten covariates, the file `Pairwise_covariates_[species code].csv` will be saved instead of a png file, containing the correlation matrix. For large numbers of covariates this file can be very bulky. In this case `dataExploration()` also stores a table called `Correlated_covariates_[species code].csv` containing only covariates with am absolute correlation coefficient of equal to or larger than the value specified in argument `min.corr`.

9 Function `SDMaps()`

`SDMaps()` accepts a large number of arguments that specify various steps and options in the modelling process. Roughly, you can divide the arguments into three groups:

Arguments that describe the model With these arguments you determine whether you use presence/absence or count data, whether you use a GLM (or one of its various extensions) or a tree-based model, whether you desire a graphical map output, which years, environmental variables or species should be included to or excluded from the analysis, and various other specifications.

Arguments that specify residual interpolation and crossvalidation With these arguments you specify whether you want to perform an n-fold crossvalidation to assess the power of the model, whether you want to make a spatial interpolation of the model residuals and which interpolation method you want to apply.

Output arguments With these arguments you specify whether the maps should be plotted in a graphics file, where the results should be saved, and some other output options.

9.1 Arguments that describe the model

This section is divided into subsections:

Here you determine whether you use presence/absence or count data, whether you use a GLM (or one of its various extensions) or a tree-based model, whether you desire a graphical map output, which years, environmental variables or species should be included to or excluded from the analysis, and various other specifications.

- Basic model arguments: With these arguments you specify whether you do a GLM of count or of presence/absence data, which formula you use, whether there are interactions between the environmental variables and whether you use polynomials.
- Arguments for GLM extensions: With these arguments you provide the necessary control arguments for various GLM extensions such as GAMs or MARS models.
- Arguments for tree based models: Here you can provide the necessary control arguments for various flavours of tree based models such as boosted regression trees or random forests.
- Other arguments: With these arguments you can control specific additional calculations, e.g. prediction intervals for each local prediction, an estimation of total population size (only for count data) or the use of geographic coordinates as covariates. This collection of arguments is rather heterogeneous and not all specific calculations are available for all methods.

9.1.1 Basic model arguments

SDMdata The data object generated with `data2SDMaps()`.

model.type Character. You may choose between the following options (default is `"glm"`):

`"glm"` Use a generalised linear model (GLM).

`"quasi"` Estimate an overdispersion parameter for simple binomial or poisson GLM models. (Do not mix up this argument with R function `stats::quasi`!)

`"mixed"` Use a generalised linear model (GLM) with random effects (= mixed model). [NOTE: is still in experimental stage. Possible bugs.]

`"serial"` Use a generalised linear model (GLM) adjusting for serial correlation in data (e.g. from site by year count data).

"smooth" Use an generalised additive models (GAM). (Do not mix up this argument with R function stats::smooth!)

"joint" Use Joint (=Hurdle) models of presence/absence and count data.

"mars" Use multivariate adaptive regression splines (MARS) models.

"gbm" Use a generalized boosting model (GBM) (aka boosted regression trees, BRTs).

"rf" Use a random forest (RF) model.

"ranger" Use a random forest model with the fast ranger algorithm (RANGER).

"qrf" Use a quantile regression forest (QRF) model.

"ctree" Use conditional inference trees (CTREEs).

"cforest" Use a conditional inference forest (CFOREST) (experimental).

"evtree" Use evolutionary learning of a globally optimal tree (EVTREE).

"mob" Use model based partitioning (MOB).

"neuralnet" Use an artificial neural network (NN).

data.type Character. You may choose between "presence" for binary presence/absence data, "count" for discrete count data, "density" for continuous density data (lower bound at 0) and "normal" for normally-distributed data. A special data type "trendindex" is used for trend indices bounded between -1 and 1. Default is "count". The input for presence/absence data has not to be binary; input values > 0 will automatically be converted to 1. The use of density is still experimental. If you get strange results with it, consider to use count, in which case all density values are rounded to the next integer. The definition of data.type is relevant for GLMs and their extensions and for GBMs (for the latter, density is not yet available); quantile regression forest can only be used for regression purposes (data.types count, normal, density or trendindex). The following table gives an overview, which combinations of model.type and data.type can currently be used (glm includes quasi, mixed, serial, smooth and joint).

data.type	model.type		
	GLM, MARS, RF, RANGER, CTREE, CFOREST, EVTREE, NEURALNET	GBM, MOB	QRF
presence	X	X	
count	X	X	X
density	X		X
normal	X	X	X
trendindex	X	X	X

spec.rich Currently not used.

formula Formula: Can be used to specify a relationship between response and covariates. e.g. observed ~ YEAR + [covariates]. When left unspecified, all covariates in the dataset are used (but see vars.subs below).

interactions Numeric. Specifies the interaction order (0 = no interaction, 1 = one-way, 2 = two.way, etc). Only for BRT and GLM and its extensions. Default is 0.

polynomials Numeric. Includes higher order polynomials of the covariates (1 = no polynomials, 2 = polynomials of 2nd degree, 3 = polynomials of 3rd degree). Default is 1.

9.1.2 Arguments for GLM extensions

mx.control List with arguments to specify mixed effects model (model.type = "mixed"). Use in conjunction with `.mixedSDMOptions()`. [To be completed.]

sr.control List of control options for serial correlation (model.type = "serial"). See `.serialSDMOptions()`. [To be completed.]

sm.control List of control parameters for a GAM (model.type = "gam"). see `.smoothSDMOptions()`. [To be completed.]

jt.control List of control parameters for joint models (model.type = "joint"). See `.jointSDMOptions()`. [To be completed.]

mr.control List of control parameters <for MARS models (model.type = "mars"). See `.marsSDMOptions()`. [To be completed.]

offset Logical. Use one of the covariates as offset? Can be specified by either `TRUE/FALSE` (when `TRUE` a dropdown list will appear), or by a character string matching exactly one of the column names of plot.data. Default is `FALSE.`

weights Logical. Add weights? For specification possibilities see `offset`. Default is `FALSE.`

selectAIC Character. Use stepwise model selection (=`"step"`) or not (=`NULL`).

9.1.3 Arguments for tree based models

gbm.control This option is used to set control parameters for GBMs via invoking function `gbmSDMOptions()`. Use the syntax `gbm.control = gbmSDMOptions(option1=[ ], option2=[ ], ...)`, for example, `gbm.control = gbmSDMOptions(learning.rate=0.001, step.size=1, tree.complexity=2)`. Defaults are:

tree.complexity Numeric. Sets the complexity of individual trees. Default is 1.

learning.rate Numeric. Sets the weight applied to individual trees. Default is 0.01.

bag.fraction Numeric. Sets the proportion of observations used in selecting variables. Default is 0.75.

fold.vector Allows a fold vector to be read in for CV with offsets. Default is NULL.

prev.stratify Prevalence stratify the folds – only for presence-absence data. Default is TRUE.

n.trees Numeric. Number of initial trees to fit. Default is 50.

step.size Numeric. Number of trees to add at each cycle. Default is 50.

max.trees Numeric. Maximum number of trees to fit before stopping. Default is 10000.

tolerance.method Character. Method to use in deciding to stop: "fixed" or "auto". Default is "auto".

tolerance Numeric. Tolerance value to use: if method == "fixed", it is absolute; if "auto" it is multiplied with the total mean deviance. Default is 0.001.

keep.data Logical. Keep raw data in final model? Default is FALSE.

plot.main Logical. Plot hold-out deviance curve? Default is FALSE.

plot.folds Logical. Plot the individual folds as well? Default is FALSE.

verbose Logical. Controls amount of screen reporting. Default is FALSE.

silent Logical. Allow running without output for simplifying model? Default is TRUE.

keep.fold.models Logical. Keep the fold models from crossvalidation? Default is FALSE.

keep.fold.vector Logical. Allow the vector defining fold membership to be kept? Default is FALSE.

keep.fold.fit Logical. Allow the predicted values for observations from CV to be kept? Default is FALSE.

rf.control This option is used to set control parameters for random forests via invoking function `rfSDMOptions()`. Use the syntax `rf.control = rfSDMOptions(argument1=[ ], argument2=[ ], ...)`, for example, `rf.control = rfSDMOptions(ntree=1000)`. The arguments that can be set are:

ntree Number of trees in the random forest. Default is 500.

maxnodes Maximum number of terminal nodes trees in the forest can have. As default, trees are grown to the maximum possible (subject to limits by nodesize).

nodesize Minimum size of terminal nodes. Larger numbers cause smaller trees to be grown and thus enhance modelling speed. Default is 5.

rf.mtry Numeric or character string 'tuneRF'. If set to an integer, the corresponding number of covariates will be used in each split. If set to 'tuneRF', the optimal number of covariates used in each split of the single trees in a random forest is determined by function `tuneRF()` of the randomForest package. According to Breiman [2001] the value of mtry has little influence on the result, it may be wise, however, to use this feature for large numbers of covariates. Default is `NULL` (= the package randomForest uses the default for regression trees which is number_of_covariates/3). *Remark*: This argument probably will be part of `rf.control` in the future.

ranger.control This option is used to set control parameters for random forests via invoking function `rangerSDMOptions()`. Use the syntax `ranger.control = rangerSDMOptions(argument1=[ ], argument2=[ ], ...)`, for example, `rf.control = rangerSDMOptions(num.trees=1000)`. Relevant arguments are (for full list of arguments see the ranger manual):

num.trees Number of trees. Default is 500.

mtry Number of variables to split at each node. Default is the (rounded down) square root of the number of variables.

min.node.size Minimal node size. Default is 5.

impute Character. One of `"roughfix"` or `"impute"`. RF, RANGER and QRF models cannot deal with missing covariate values in the plot data, they must thus be imputed. If set to `"roughfix"`, missing values of covariates are replaced with the respective covariate medians. For factor covariates, missing values are replaced with the most frequent levels. If set to `"impute"`, missing values are first imputed by the `na.roughfix` algorithm. Then random forest is called with the completed data. The proximity matrix from the randomForest is used to update the imputation of the missing values. For continuous predictors, the imputed value is the weighted average of the non-missing obervations, where the weights are the proximities. For categorical predictors, the imputed value is the category with the largest average proximity [Liaw and Wiener, 2002]. Imputing with option `"impute"` needs much longer than `na.roughfix` which is quite inconvenient when a modelling run loops over many species, repeatedly imputing the same plot covariates. Imputed data are therefore saved in the working directory on the hard disk, named `imputed_plot_covars.RData`, and can be recycled for species that are modelled with the same covariates. Default is `"roughfix"`.

ctree.control This option is used to set control parameters for conditional inference trees via invoking function `ctreeSDMOptions()`. Use the syntax `ctree.control = ctreeSDMOptions(argument1=[ ], argument2=[ ])`, for example, `ctree.control = ctreeSDMOptions(mtry=10, maxdepth = 3)` where `mtry` is the number of input variables randomly sampled as candidates at each node and `maxdepth` is the maximum depth of the tree. For full range of control parameters and its default values see the manual of package `party` [Hothorn et al., 2019].

cforest.control This option is used to set control parameters for conditional inference forests via invoking function `cforestSDMOptions()`. Use the syntax `cforest.control = cforestSDMOptions(argument1=[ ], argument2=[ ])`, for example, `cforest.control = cforestSDMOptions(ntree=1000, mtry=10)`. While with random forests `mtry` (the number of variables randomly sampled as candidates at each split) is automatically determined according to the number of explanatory variables, its default is 5 for conditional inference forests because of technical reasons. It thus has to be adjusted by the user. We advise to use the settings of random forest: `mtry = floor(n_of_covariates/3)`, for example, `mtry=5` for 16 explanatory variables. Default for `ntree` is 500. For full range of control parameters and its default values see the manual of package `party` [Hothorn et al., 2019].

evtree.control This option is used to set control parameters for globally optimal trees via invoking function `evtreeSDMOptions()`. Use the syntax `evtree.control = evtreeSDMOptions(argument1=[ ], argument2=[ ], ...)`, for example, `evtree.control = evtreeSDMOptions(niterations=20000, ntrees=500)`. Defaults are:

minbucket Integer. The minimum sum of weights in a terminal node. Default is 7.

minsplit Integer. The minimum sum of weights in a node in order to be considered for splitting. Default is 20.

maxdepth Integer. Maximum depth of the tree. Memory requirements increase by the square of the maximum tree depth. Default is 9.

niterations Integer. In case the run does not converge, it terminates after a specified number of iterations defined by niterations. Default is 10000.

ntrees Integer. The number of trees in the population. Default is 100.

alpha Numeric. Regulates the complexity part of the cost function. Increasing values of alpha encourage decreasing tree sizes. Default is 1.

operatorprob List or vector of probabilities for the selection of variation operators. May also be specified partially in which case the default values are still used for the un- specified arguments. Always scaled to sum to 100 percent. Default is `list(pmutatemajor = 0.2, pmutateminor = 0.2, pcrossover = 0.2, psplit = 0.2, pprune = 0.2)`.

seed Numeric. A numeric seed to initialize the random number generator (for reproducibility).

mob.model Character. Vector of covariates that are to be used for the models in the leaves when `model.type` is `"mob"`.

mob.part Character. Vector of covariates that are to be used for partitioning when `model.type` is `"mob"`.

9.1.4 Arguments for Artificial Neural Networks

neuralnet.control This argument is used to set control parameters for artificial neural networks via invoking function `neuralnetSDMOptions()`. Use the syntax `neuralnet.control = neuralnetSDMOptions(argument1=[ ], argument2=[ ], ...)`, for example, `neuralnet.control = neuralnetSDMOptions(hidden=20, stepmax=500000)`. SDMaps uses the `R` package `neuralnet` [Günther and Fritsch, 2010, Fritsch and Günther, 2016] to train neural networks which allows for closely defining the network complexity and selecting among various training algorithms. Defaults of the most relevant arguments are:

hidden A vector of integers specifying the number of hidden layers and the number of nodes in each layer. For example, `hidden=10` induces a network with one hidden layer with 10 nodes in it; `hidden=c(3,2,1)` creates a neural network with three hidden layers, the first one with three, the second one with two and the third one with one node. `SDMaps` uses one hidden layer by default and sets the number of nodes to the number of covariates.

threshold A numeric value specifying the threshold for the partial derivatives of the error function as stopping criteria. Default is 0.01. For small thresholds, the network may not converge within the maximum number of steps defined by stepmax (see below). In this case, `SDMaps` automatically multiplies the threshold by 5 and trains a new network. In case of further non-convergence this process is iteratively repeated with a maximum of five times. Alternatively, it may be wise to make some preliminary modelling in order to choose a reasonable threshold value.

stepmax The maximum number of steps allowed for the training of the neural network. The training process stops when stepmax is reached. Default is 100 000.

algorithm A string containing the algorithm type to calculate the neural network. The following types are possible: `"backprop"` (backpropagation), `"rprop+"` (resilient backpropagation with weight backtracking, Riedmiller [1994]), `"rprop-"` (resilient backpropagation without weight backtracking, Riedmiller and Braun [1993]) and `"sag"` or `"slr"` (the modified globally convergent algorithm, Anastasiadis et al. [2005]). Default is "rprop+". For a short explanation of the different algorithms see Günther and Fritsch [2010].

For a complete list of arguments see the manual of R package neuralnet Fritsch and Günther [2016].

9.1.5 Other arguments

spec.subs Character. Vector with species (e.g. `spec.subs = c("Fox", "Badger")`) that should be used in the modelling run. If left unspecified, it will run for all species you included to your plot data (see `plot.data` in section 7.1). Default is `NULL`.

year.subs Character. vector of years (e.g. `year.subs = c("2008", "2009")`) that should be used in the modelling run. If left unspecified, all years that you provided in your plot data will be included (see `plot.data` in section 7.1). Default is `NULL`.

vars.subs Character vector of predictor variable names that should be used or skipped in the modelling run. If left unspecified, all covariates will be included. Example: `vars.subs = c("Farmland", "Forest")` (only include Farmland and Forest) or `vars.subs = c("-YEAR")` (don't use YEAR). Attention: You cannot mix using and skipping mode of vars.subs (for example, `vars.subs = c("Farmland", "Forest", "-YEAR")`! The predictor variables defined in vars.subs will be used *for all species* in the modelling run. If you desire species-specific covariates, then use `vars.subs = "species-specific"`. In this case you must provide a text file named `species_specific_predictors.txt`; the path to the directory where it is stored has to be specified in argument `path.to.species.specific.predictors` (see below). The file has to contain two komma separated columns, the first called `species`, the second called `variable`. For each predictor used for a given species a line has to be provided in the file. The following file

```
species, variable
SpecA, Var1
SpecA, Var2
SpecA, Var3
SpecB, Var1
SpecB, Var4
SpecB, Var5
```

thus defines that you use the predictor variables 1, 2 and 3 for making the model of species A, and the predictor variables 1, 4 and 5 for the model of species B.

The Arguments `spec.subs`, `year.subs` and `vars.subs` guarantee that you need to create only one `sdmdata` object (see 5.1. Function `data2SDMaps()`) for a number of different modelling runs using different species/year/predictor combinations.

Attention: We observed that `vars.subs` does not work when there is a large number of explanatory variables and too many variables are chosen to be used (or to be skipped). In this case it

seems to be better to provide a new directory with the actually used maps and to generate a new `SDMaps`-object.

path.to.species.specific.predictors Character. Path to the directory where the file `species_specific_predictors.txt` is stored. Default is NULL.

var.select Character or character plus a numeric. Variable selection can have two motivations:

(1) Variables that show problems of collinearity have to be skipped for analysis with GLM and its derivatives. Variable selection for avoiding collinearity among covariates is made by means of inspecting variance inflation factors (VIF, see argument variance.inflation.factors in section 8.1) and skipping covariates accordingly. The corresponding arguments are `"vifstep"` along with an integer (for example, `var.select = c("vifstep", 5)`) and `"vifcor"` along with a numeric <1 (for example, `var.select = "c(vifcor", 0.9)`). `"vifstep"` calculates VIF for all covariates and removes the covariate with the highest VIF (if larger than the specified integer) is removed. The procedure is iteratively repeated until no variables with a VIF larger than the specified integer are left. `"vifcor"` identifies the pair of variables which has the maximum linear correlation (larger than the specified numeric) and excludes the one of them with the larger VIF. The procedure is repeated until there is no variable remaining with a correlation coefficient larger than the specified numeric. Relevant values for the integer or numeric accompanying `"vifstep"` or `"vifcor"`, respectively, can be chosen based on the results of the data exploration.

(2) According to Evans et al. [2011], the trees in a random forest and its relatives become much shallower when spurious variables are skipped which in turn reduces the size of the plurality vote matrix by reducing votes that account for noise. This causes a higher signal to noise ratio and overall reduction in error and thus calls fro some variable selection. Random forest provide an excellent means to determine the importance of each variable (see Liaw and Wiener [2002] for details). When argument `"cvrf"` is used, the prediction performance of models with sequentially reduced number of predictors (ranked by variable importance) via a nested cross-validation procedure is used to identify the smallest possible set of covariates in the analysis that does not increase the cross-validated prediction error by more than 5 % as compared to a random forest using all covariates. Package `VSURF` [Genuer et al., 2015] provides a two-stage strategy based on a preliminary ranking of the covariates and a stepwise forward strategy for variable introduction. It is possible to choose between an approach yielding a subset of covariates of important variables including some redundancy which can be relevant for interpretation ("`VSURF.interp`") and another approach trying to avoid redundancy focusing more closely on prediction ("`VSURF.pred`") [Genuer et al., 2010].

The variables selected by any of the described procedures are passed on to the analysis (which may be by means of any other implemented modelling technique, not just GLM or random forest). `var.select` can be combined with `vars.subs`: a variable is included to the analysis if it is chosen via `var.subs` even though `var.select` identifies it as being not important; a variable is excluded from the analysis if it is excluded via `var.subs` even though it may be identified as important by `var.select`. Default is NULL.

trend Logical. Use YEAR covariate (if present) as a continuous variable? If `FALSE` assumes categorical nature. Default is `TRUE`.

surface Logical. Use x and y variables as covariates? Default is `FALSE`.

popsize Logical. Calculate total population size (with prediction intervals)? Default is `FALSE`.

rf.popsize.intervals Numeric. When argument `popsize` is set to `TRUE` and model.type is set to `rf` (random forest, see 9.1.1) confidence and prediction intervals around the estimated total population size can be calculated. For example, use `rf.popsize.intervals = 95` or `rf.popsize.intervals = 90` for 95% or 90% confidence and prediction intervals, respectively. Default is `NULL` (no intervals will be calculated).

rf.popsize.intervals.arg Character. Should de model outputs ("raw") or the outputs that are corrected with the interpolated residuals ("int.resids") be used for calculating the intervals around the population size estimation. Default is raw.

pois.pred.int Logical. Should prediction intervals be calculated for a Poisson GLM? Default is FALSE.

rf.local.confint Numeric. Which confidence interval width for local predictions do you want to specify? For example, use rf.local.confint = 95 for a 95%, rf.local.confint = 90 for a 90% confidence interval. Default is NULL (no confidence interval will be calculated).

rf.local.confint.arg Character. Should de model outputs ("raw") or the outputs that are corrected with the interpolated residuals ("int.resids") be used for calculating the local confidence intervals. Default is raw.

gbm.pred.int Numeric. Should local prediction intervals be calculated for a GBM? If so, provide the number of resampling iterations. Default is 0.

gbm.pred.int.args Numeric. Provide the lower and the upper quantile for the local prediction interval. Default is c(0.025, 0.975), corresponding to a 95% prediction interval.

qrf.quantiles Vector with quantiles that should be calculated; should always be one (the quantile to be modelled, normally the median) or three (corresponding to the lower limit of the prediction interval, the median, and the upper limit of the prediction interval). When set to NULL (default) the quantiles will be calculated for c(0.1, 0.5, 0.9).

qrf.maxvect Numeric. Maximum permissible length of vector. For making quantile regression forest predictions on the prediction grid very large vectors must be generated which can be too large for available memory and cause the model run to crash. In this case the maximum vector length can to be defined by the user and the prediction grid will be filled piecewise. If left unspecified (NULL), SDMaps will try to check available memory automatically and identify a limit for vector length. Default is NULL.

qrf.bias.corr Logical. QRFs tend to yield lower count estimates than RFs. If you use QRFs to get a prediction interval for a RF prediction you can set qrf.bias.corr to TRUE. In this case SDMaps calculates a linear regression between the point predictions of RF and QRF and applies it for the predictions on the prediction grid. Default is FALSE.

popsize.arg Numeric. Needed to convert from abundance scale (e.g. territories per km²) to the scale of the prediction grid (e.g. 250x250 m). In this case, for example, values have to be multiplied by popsize.arg = 62 500 m² / 10 000 000 m² = 0.00625 to get the correct estimate for a grid cell. Default is 1.

treat.outliers Vector with three elements. With this argument you can exclude or cut off outliers for count and density data. The first element is "percentile" or "value", the second "exclude" or "truncate" and the third a numeric in the interval [0 1] or in the range of the data. All values larger than the given percentile or the given value are either excluded or truncated. For example, treat.outliers = c('percentile', 'truncate', 0.99) sets all values larger than the 99% percentile to the value of the 99% percentile. Default is NULL (= all data used).

set.to Value to set when categorical covariate levels of prediction data are missing in plot.data. Default is NA.

cores Numeric. How many cores should be used when executing code in parallel? (currently only used for the prediction interval and population size estimation with gbm, that is the combination model.type = "gbm" and gbm.pred.int = TRUE) Default is NULL.

9.2 Arguments for crossvalidation and dealing with spatial autocorrelation

cross.validation Numeric. Use cross.validation=*n* for *n*-fold cross validation. Default is 0 (no crossvalidation).

For dealing with spatial autocorrelation, only one of the three approaches (regression-kriging, RAC or RFsp, see section 2.8) can be applied at a time. Currently, for regression-kriging six arguments (`resid.int.method`, `variogram.input`, `int.args`, `interp.weights`, `inttype` and `strat.interpol`) must be set (will be simplified in future versions).

resid.int.method One of `NULL`, `"IDW"`, `"krige"` or `"TPS"`, corresponding to none, Inverse Distance Weighting, Kriging or Thin Plate Spline interpolation of response residuals. The resulting maps with interpolated residuals are added to the maps with model predictions. Default is `NULL`.

variogram.input [How is this information to be provided?] If external information available, one can specify manually the variogram for use with kriging.

int.args List with arguments that can be passed to the functions `krige()` and `idw()` from R-package `gstat`, for example, `int.args = list(maxdist = 10000, block = c(2000, 2000))`, Here, only the most relevant arguments are presented; for a complete list of possible arguments see the manual of `R`-package `gstat` [Pebesma, 2004].

nmax Numeric: The number of nearest observations that should be used. Default is `Inf` (all observations are used).

maxdist Numeric: Only observations within a distance of maxdist from the prediction location are used. Default is `Inf` (all observations are used).

nmin Numeric: If the number of nearest observations within distance maxdist is less than nmin, a missing value will be generated. Default is `0`.

block Numeric: A vector with two values containing the size of a rectangle in x- and y- dimension. Default is the scale of the prediction grid (HENK ODER CASPAR FRAGEN!).

interp.weights Logical. Use distance based weighting of residuals before adding to data so that interpolated values far from sampled locations are less affected by the residual processes. Default is `FALSE`.

inttype Character. Type of residuals that should be used for interpolation (either `"link"`, or `"response"`). Default is `"response"`.

strat.interpol Character: The path to an ASCII-grid that represents a stratum or the the path to a directory that contains ASCII-grids of several strata. There are two possibilities of performing a stratified interpolation of the residuals: (1) A core stratum is defined and only in the core stratum an interpolation of the residuals is done. This is useful for species with a narrow habitat choice. Observations outside the core stratum are preserved in the presence-absence or abundance maps, but they are not used for interpolation. Observations outside the core stratum may be coincidential or due to mistakes during data input or the like. For example, waterfowl does not appear in purely terrestrial habitats; the residual interpolation thus should not spill over into forests or agricultural areas adjacent to water bodies. The path to an ASCII file representing the grid map of the stratum for which the residual interpolation should be done, is required. When only the name of the grid map is provided, it is looked for in the working directory. The grid map must be of the same resolution as the covariate grid maps used for `data2SDMaps()` (see section 4.3). It should contain numerical values for the grid cells belonging to the stratum and NA's for the grid cells outside. (2) The entire map area is divided into non-overlapping strata. Within each stratum an interpolation is done, but the interpolated areas cannot flow out into adjacent strata. This is useful for birds with broader habitat choice, appearing in various strata. Here, provide the path to a directory storing the non-overlapping strata. When the option is set to a file name or a directory containing strata grids, these map will be used for all species in the analysis. You may also set the option to `"species-specific"`; in this case a text file with the name `strata.txt` must be provided and the path to it specified via argument `path.to.species.specific.strata` (see below). The text file must consist of the columns code (the species codes used in the analysis) and `stratum` (path to the map) as in the example table below. Default is `NULL` (residual interpolation made for the entire prediction grid).

```
code, stratum
5320, /my_home_directory/stratum_maps/map_for_godwit.asc
5460, map_02.acs
5610, stratum_map.asc
```

path.to.species.specific.strata Character. Path to the directory where file `strata.txt` is stored. Default is NULL.

RAC Character. When set to the path to the directory holding the covariate grid maps, the interpolated residuals will be saved as an ascii-grid file in the directory, named RAC.asc (*residual covariate*). Subsequently, you can generate sdmdata again–now including the RAC–and execute another modelling run with `SDMaps()`, this time without interpolation of residuals. Default is `NULL`, the interpolated residuals will not be stored as covariable.

RFsp Logical. If set to TRUE, the random forest for spatial predictions framework is applied and for each observation point in the plot data a layer with buffer distances will be added to the covariates plot data as well as to the prediction grid. Default is `FALSE`.

9.3 Output arguments

save.text.files Character. Define which objects you want to save on the hard disk as text files. You may choose between `the options "preds"` (writes the file preds_[modelling technique]_[analysis type]_[species code].txt), `"points"` (writes the file PointPredictions_[modelling technique]_[analysis type]_[species code].txt) or `"modelpreds"` (writes the file model.preds_[modelling technique]_[analysis type]_[species code].txt) and combinations of them, e.g. as `c("points", "preds")`. Use option `"all"` when you want to save all objects as text files, but be careful: Textfiles can be very large and need much time for saving! See section 9.4 for a closer description of the contents of the text files. When the file speciesnames.txt (see 4.5) is provided, also the speciesname is included in the file names following the species code. Default is `NULL` (no text files are saved on the disk).

dec Character. Tables stored can conveniently be inspected with a spread sheet. Specify if you use "`.`" or "`,`" as decimal symbol. Default is "`.`".

plots Logical. Make pictures of resulting maps? Default is `TRUE`. When option `data.type` is set to `"presence"` the legend is bounded between 0 and 1 and divided into 10 classes (0, 0.1, 0.2, ..., 0.9, 1); for data.type `"trendindex"` it is bounded between -1 and 1, divided into 10 classes (-1, -0.8, -0.6, ..., 0.8, 1). For data.types `"count"`, `"density"` and `"normal", for` each map the optimal colour legend is determined, depending on the values in the map. This may hamper direct comparison between single maps. If you desire to display selected maps with an identical colour legend, run `your SDMaps` analysis – with `plots = TRUE` or `plots = FALSE` – and afterwards invoke function `SDMapsPostHocPlot()` (see section 13).

lowest.value.white Logical? Should the lowest value in the map be white? Default is `FALSE` (lowest value is light blue).

plot.raw.residuals Logical. Make also a picture of the raw residuals? Can need much time! Default to FALSE.

path.to.contour Character. Specify the path to the shape-file (see section 4.4) including its file name. If no path is specified, `SDMaps` looks for a file named contour.shp in the working directory. If this file is not available neither, then a warning is displayed and the maps are drawn without contours or additional features such as national borders, rivers etc. *Attention*: The location plot (see section 9.4) cannot be drawn without the contour file!

path.to.speciesnames.file Character. Path to the directory where file `speciesnames.txt` is stored. The file permits the use of full species names in file names and graphics instead of codes (e.g. EURING numbers). It must consist of two comma-separated columns named 'code' and 'species', for example:

```
code, species
5320, Limosa limosa
5460, Tringa totanus
5610, Arenaria interpres
```

plotargs (CURRENTLY NOT USED) List of options with arguments to plotting predictions. See `plot.options()`.

scale.map.legend Numeric. Multiplicative factor that scales the legend of the map with the final prediction; for example, `scale.map.legend=100` for a model that has been calculated based on territories per ha will produce a map that reports territoria per square kilometer (= 100 ha). Default is `1`.

driver Character. Use `"tif"`, `"asc"` or `"shp"` (tif = geotiff-grid, asc= ESRI ASCII-grid, shp = ESRI shapefile) or a combination of them (e.g. `c("asc", "tif")`, in which case maps will be produced in both formats).

asc.long.name Logical. Given the file speciesnames.txt is provided (see section 4.5), should full species names also be included to the map filenames? While it is quite convenient to have the full species names in plotted maps etc., ascii, tiff and shp files often are used for further processing and bulky filenames might be quite difficult to handle. If you want to see full file names, anyway, set this option to `TRUE`. Default is `FALSE` (no full species names used in map names).

out.dir Specify output directory.

return.on.console Logical. Return supplied data? Default is `TRUE`.

named Character. Name of the modelling run.

9.4 Products delivered by `SDMaps()`

`SDMaps()` saves its outputs in a number of files stored in the results directory defined by argument `out.dir`. Within the result directory, consecutive modelling runs are stored in subdirectories RESULTS, RESULTS1, RESULTS2 and so on. Per modelling run, for each of the species in the analysis the following files (with the exception of sdmdata_analysis.RDATA) will be stored (in brackets the used modelling techniques, analysis types, species codes etc. are given; optional elements of the file names are shown in parantheses):

9.4.1 Locations of the observations

- File name: `Locations_[species code](_[species name]).png`

For each species in the analysis a map is drawn with the locations of their observations. Dependent on the data type either presences and absences or values (counts, densities, indices) on the locations are shown.

9.4.2 Result of variable selection

- File name: `Variable_selection_[species code](_[species name]).pdf`

If options `"cvrf"`, `"VSURF.interp"` or `"VSURF.pred"` were chosen for argument `var.select`, informative plots are delivered. Consult Genuer et al. [2015] for the interpretation of the VSURF related plots.

9.4.3 Model per species

- File name: `[modelling technique]_[analysis type]_[species code](_[species name]).RData`

The name of this binary `R`-file is composed of the used modelling technique (GLM, MARS, ...), the type of analysis (count or presence) and the species code. It contains an object with the name [modelling technique]_[analysis type]_[species code] where model-specific information is stored. For a closer description the documentation of the used `R` commands or packages (`glm`, `gbm`, `randomForest`, ...) should be consulted.

9.4.4 Predictions for the plot locations per species

- File name: `PointPredictions_[modelling technique]_[analysis type]_[species code](_[species name]).txt`

For each species in the analysis a text file is stored that includes the plot data (plotID, observed, x-coordinates, y-coordinates, covariates) along with the model predictions and the residuals for each plot. Only done when option `save.text.files` is set accordingly.

9.4.5 Model residuals per plot location per species

- File name: `Residuals_[modelling technique]_[analysis type]_[species code](_[species name]).png`

For each species in the analysis the model residuals are displayed. The size of the symbols in the map corresponds to the absolute value of the residuals. Positive residuals (model overestimates the response variable) are shown in blue, negative residuals (model underestimates the response variable) are shown in red.

9.4.6 Predictions for the prediction grid (= the entire area) per species

- File name: `preds_[modelling technique]_[analysis type]_[species code](_[species name]).RData`

This binary `R`-file contains an object named `predgrid`. There are separate `preds_*.RData` files for each species in the analysis. Each file contains an object consisting of class "SpatialGridDataFrame" containing the grid topology and a dataframe with (a) the prediction grid coordinates, (b) the cell weights, (c) the values of the covariates, (d) the raw predictions (= predictions without spatial interpolation of residuals), (e) the interpolated residuals and (f) the predictions (= raw predictions + interpolated residuals) as well as a column indicating the species. If you choose the option `save.text.files = "preds"` the object is also saved as a text file.

- File names: `[prediction_type]_[analysis type]_[species code](_[species name]).[asc or tif]`
- File name: `[analysis type]_[species code](_[species name]).shp`

Dependent on which option you chose for argument driver (tif, asc or shp), for each species in the analysis geotiff, ascii or shapefiles are produced that represent the entire prediction grid. The names of geotiff and ascii files are composed of the *prediction type* (predictedr = raw predictions without spatial interpolation of residuals, int.resids = interpolated residuals, predicted = raw predictions + interpolated residuals, conf.int.low = lower confidence limit, conf.int.upp = upper confidence limit, pred.int.low = lower limit of prediction interval, pred.int.upp = upper limit of prediction interval), the *type of analysis* (count, presence, ...) and the *species code*. When the file speciesnames.txt is provided (see 4.5), not only the code but also the name of the species is included in the file name. Geotiff and ascii files are accompanied by files with the same name and extension prj, which contain text representations of the spatial reference system metadata. Shapefiles include all prediction types—as well as all other information stored in `preds_[modelling technique]_[analysis type]_[species code](_[species name]).RData`—and therefore do not bear the prediction type in their names.

- File name: `model.preds_[modelling technique]_[analysis type]_[species code](_[species name]).txt`

For each species in the analysis a file is generated that contains a text representation of the dataframe stored in object predgrid (and saved in file preds_[modelling technique]_[analysis type]_[species code].RData, see above) but restricted to the columns Species, x, y, weights, predicted, predictedr and int.resids.

9.4.7 Map visualisations per species

- File names: [prediction_type]_[analysis type]_[species code](_[species name]).png

When `SDMaps()` option `plot` is set to `TRUE` (default), for each map a png file is produced. Their names are composed in the same way as the geotiff or ascii files (see 9.4.6). Labels and legends are generated due to a standard procedure and default values are used for map size, font size etc. If other representations of the maps are desired, use function `SDMapsMakeMaps()` (see section 12) or use the geotiff, ascii or shape files for processing.

9.4.8 Comprehensive information per modelling run

- File name: `[sdmdata]_analysis.RDATA`

This is a binary file that contains an `R`-object of classes "sdmdata" and "list" with the name you specified with option `named` for function `data2SDMaps()` (Default is `sdmdata`). The object stores general information about a run of `SDMaps()` and consists of the following elements that can be viewed with the command `[sdmdata]_analysis$[element]`:

data A "SpatialPointsDataFrame" of the plotdata that contains the dataframe with the covariates and their geographic reference and can be inspected with `[sdmdata]_analysis$data@data` and `[sdmdata]_analysis$data@coords`. `[sdmdata]_analysis$data@coords`, `[sdmdata]_analysis$data@bbox` and `[sdmdata]_analysis$data@proj4string`.

pred.data A "SpatialGridDataFrame" representing the grid topology (geographical bounding box and cell size) and the covariate values in the grid cells. They can be consulted by `[sdmdata]_analysis$pred.data@data` and `[sdmdata]_analysis$pred.data@grid`, `[sdmdata]_analysis$pred.data@bbox` and `[sdmdata]_analysis$pred.da`

data.name Character string that store the name of the object itself (default: `sdmdata_analysis`).

files Character string that stores the path to the results directory where the result files are stored

Userdir Character string that stores the path to the grid maps of the covariates

nyears and years Number of years and year dates in the plot data. When no years are provided, `data2SDMaps()` assumes that all data come from the same year with year date 1 (`nyears = 1`, `years = 1`).

n.unique.plots The number of unique plots in the plot data

specnames Character strings with namesof the files where the species observations are stored, consisting of the code used for each species and the extension DAT

not_run Character vector of species which produced an error and will not be included to the analysis

not_run_errors Error messages for the species which produced an error

missing.count.data Number of plot ID's from the observation file that are missing in the plot data

missing.plot.data Number of plot ID's from the plot data that are missing in the observation file

call Call of `SDMaps()` in which all of the specified arguments are specified by their full names

attrs Specification of `data2SDMaps()` argument `add.zeroes`

datacall Call of `data2SDMaps()` in which all of the specified arguments are specified by their full names

nyears.used and years.used Number of years and year dates used as explanatory variables (set in `SDMaps()` option `year.subs`).

Times Time (in s) used for complete modelling run

9.4.9 Crossvalidation output

- File name: `cv_[modelling technique]_[analysis type]_[species code](_[species name]).csv`
- File name: `cv_[modelling technique]_[analysis type]_[species code](_[species name]).pdf`

The csv-file reports several cross-validation statistics. For presence-models it presents the AUC values for each iteration of the cross-validation. For all other analysis types it currently reports the following statistics (more and perhaps more relevant statistics will be implemented in the future):

- Pearson correlation coefficient between observed and fitted values
- Mean error (ME): It is used to characterise the bias of the model, that is, systematic over- or underestimation) and calculated as
$$ME = \frac{\sum(observed_i - fitted_i)}{n}$$
- Mean average error (MAE): It serves for characterising the model accuracy, that is, the proximity of the predictions to the observed values.
$$MAE = \frac{\sum(abs(observed_i - fitted_i))}{n}$$
- Root mean squared error (RSME): It is used to diagnose the variation in the errors in a set of forecasts. The RMSE can only be larger or equal to the MAE; in the case of RMSE=MAE all errors are of the same magnitud.
$$RMSE = \sqrt{\frac{\sum(observed_i - fitted_i)^2}{n}}$$
- Coefficient of determination R^2. Here, it is used in a naive way to estimate the amount of variation explained by the model since it does not account for model complexity.
$$R^2 = 1 - SS_{res}/SS_{tot}$$
where $SS_{res} = \sum(observed_i - fitted_i)^2$ and $SS_{tot} = \sum(observed_i - mean(observed))^2$.

For data types `count`, `normal`, `density` or `trendindex` the pdf shows a scatterplot of observed vs. fitted values for each iteration. For data type `presence` the ROC for each iteration along with the AUC value is displayed.

10 Function `SDMapsSummary()`

This function provides a summary of the fitted models in a modelling run along with various plots and statistics. Which plots and statistics can be displayed depends on the used model technique. By setting the respective options on `TRUE` or `FALSE` you can choose which plots should be generated. In the current version GLM, MARS, GBM, RF, CTREE, CFOREST, EVTREE, MOB and NN models can be evaluated, but the other modelling approaches are soon to follow (see the following table for an overview of current availability of `SDMapsSummary()`; n.a., not available; n.y.i.; not yet implemented).

data.type	model.type				
	GLM, RF, RANGER, CTREE, CFOREST, EVTREE, NN	MARS	GBM	QRF	MOB
presence	X	X	X	n.a.	X
count	X	X	X	n.y.i.	X
density	X	n.y.i.	n.a.	n.y.i.	n.a.
normal	X	n.y.i.	X	n.y.i.	X
trendindex	X	n.y.i.	X	n.y.i.	X

10.1 Arguments

SDMres `SDMaps` analysis object in the workspace that is to be evaluated or the path to a `SDMaps` analysis object saved on the hard disk. Typically you save a modelling run under a certain name – the default is sdmdata_analysis.RDATA (see 9.4.8); here you provide that name (see example below in section 17 Using scripts). When `SDMaps` crashes, no analysis object will be present in the workspace; the copy on the hard disk, however, can still be used for invoking `SDMapsSummary()` and making a summary for those species that have already succesfully been modelled before the crash.

calc.varimp Logical. Should variable importance be calculated? Default is `FALSE`.

make.varimp.plot Logical. Should variable importance be plotted? Default is `FALSE`.

make.observed.vs.fitted.plot Logical. Should observed values be plotted against fitted values? Default is `FALSE`.

make.partial.plots Logical. Should partial dependence plots be made? Default is `FALSE`.

partial.plot.vars Character or numeric. For which covariates do you want to see the partial plots? Currently, applies only to RF models because making partial dependence plots needs very long for this model type. By providing a vector with selected covariate names processing time can be reduced considerably. Alternatively, by providing a number n the production of partial dependence plots can be restricted to the n most important covariates (Attention: Does not work with modeltypes `CTREE`, `EVTREE` and `MOB`!). Default is `NULL` (partial plots will be made for all covariates).

make.roc.plot Logical. Should a Receiver Operating Characteristic curve be plotted? Default is `FALSE`.

calc.moran Logical. Should spatial autocorrelation of residuals be tested with Moran's I [Bivand et al., 2013, pp.276-284]? Attention: computing can take a long time. Default is `FALSE`.

all Logical. If set to `TRUE`, all relevant options above are set to `TRUE`. For example, if you ran a model with `data.type = "density"`, all options but `make.roc.plot` (only relevant for data.type `presence`) are set to `TRUE`. Default is `FALSE`.

dec Character. Tables stored by `SDMapsSummary()` can conveniently be inspected with a spread sheet. Specify if you use "`.`" or "`,`" as decimal symbol. Default is "`.`".

10.2 Products delivered by `SDMapsSummary()`

[modelling technique]_model_summary_[species code](_[speciesname]).txt Each R-package used for a specific modelling technique also has a specific summary or print method for the objects representing the fitted model. The information they provide thus differs widely and are stored in a txt-file.

[modelling technique]_statistics.csv For each species in the analysis various statistics (depending on data type and model type) are calculated and saved in this file. For presence models all model techniques show the AUC, For models other than presence the following statistics are calculated: mean absolute difference, mean forecast error, root mean squared error, correlation between observations and fitted values, the regression of observed on fitted values [Piñeiro et al., 2008] and explained variation (see a description of these statistics in section 9.4.9). The statistic delievered for explained variation depends on the specific modelling technique (e.g. explained deviance for GLMs, explained variation by a "pseudo-R2" for RF models, not available at all for CFOREST) and can thus not by compared across modelling techniques.

Table and plot of variable importance[modelling technique]_[Variable_Importance]_[species code](_[species name]).csv The calculation and display of variable importance differs according to the used modelling technique.

GLM Parameter estimates and covariate significance are reported in file `glm_model_summary_[species code](_[speciesname]).txt`.

MARS, GBM, RF, RANGER and CFOREST For these modelling technique variable importance is saved in file `[modelling technique]_Variable_Importance_[species code](_[species name]).csv` and the 30 most important variables are displayed in a dotchart in `[modelling technique]_Variable_Importance_[spec code](_[species name]).pdf` when argument `make.varimp.plot` is set to `TRUE`. These files contain information on the relative variable importance (the importance of all variables sums up to 100 %). There are, however, differences between the modelling techniques how to interpret variable importance. For example, a RF consists of a large number (default: 500) of single trees; for each of these trees only a subset of randomly chosen covariates is used. This allows to evaluate the importance of each single covariate, because in many of the trees correlated covariates will not appear together among the randomly chosen predictors [Liaw and Wiener, 2002]. The relative importance plot for a RF model, thus, gives an objective overview over all covariates. In GBMs, in contrast, the original tree is iteratively improved during the boosting process. Relative variable importance thus depends on which of the correlated covariates have entered to the model. For example, if covariates A and B are highly correlated and A has already entered to the model during tree generation or a given step of the boosting process, B adds only insignificantly to the predictive efficiency of the model and will receive a low relative importance score. If A were skipped from the analysis, however, B would have high relative importance. Consequentially, covariates with low relative importance in a GBM must not uncritically be regarded as generally unimportant; the importance score only applies to their importance *in the model*.

Attention: Variable importance is currently not available for modelling type CFOREST due to extremely high RAM requirements!

CTREE, EVTREE and MOB For these modelling techniques, variable contribution is not expressed as relative importance but is displayed by the tree itself. It is stored in file `[modelling technique]_Tree_[species code](_[species name]).pdf`.

NN For artificial neural networks two files are produced: one shows the network topography along with the weights of the single node connections, saved as `neuralnet_Net_[species code](_[species name]).pdf`. Another displays figures of the generalized weights, one for each covariate, and is saved as `neuralnet_generalized_weights_[species code](_[species name]).pdf`. Roughly, generalized weight plots are comparable with partial dependence plots and show which values of the covariates exert a small or large influence on the response.

[modelling technique]_ROC-plot_[species code](_[species name]).pdf This plot is made when the corresponding argument `make.roc.plot` is set to `TRUE`. It shows the receiver operating characteristic, calculates the area under the curve (AUC) and show the AUC of thresholds for making binary maps according to three approaches: the True Skill Statistic by maximising sensitivity and specificity (abbreviated MaxSens+Spec in the plot), maximising the percentage of true classification (MaxPCC) and observation prevalence (ObsPrev) [Allouche et al., 2006].

[modelling technique]_Observed_vs_Fitted_[species code](_[species name]).pdf For data types other than `presence`, the frequency distributions of observed and predicted values are displayed and a scatterplot of observed versus fitted values is generated. According to Piñeiro et al. [2008] observed values are regressed on fitted values, and it is tested whether the intercept is different from 0 (which would indicate a bias) and whether the the regression slope is different from 1. The values of intercept and slope and their p-values are included in the csv-file reporting model statistics (see paragrapgh on file [modelling technique]_statistics.csv in section 10.2). For `presence`, instead of the scatterplot a goodness-of-fit plot for presence/absence data is shown.

[modelling technique]_Partial_Dependence_Plots_[species code]([_species name]).pdf For a given species, this file contains a series of partial dependence plots (aka partial plots or response plots). In partial dependence plots each covariate is plotted vs the model prediction after considering the average effect of the other covariates in the model. Partial dependence plots are tools for visualizing the effects of variables on the predictions of a "blackbox" model, but they do not account for interactions between the covariates. They

are available for modelling techniques MARS, GBM and RF. For GLMs similarly determined relationships between covariates and observations are stored in file `glm_response_curves_[species code]([_species name]).pdf`.

[modelling technique]_Morans_I_[species code]([_species name]).pdf This file displays Moran's I of the model residuals on distance classes and allows to judge on which scale autocorrelation is present in the residuals. Statistically significant values ($p<0.05$) are plotted in red. An accompanying text file with bin centers, the coefficient values, the probability of the Null hypothesis and the number of pairs per distance class is saved as [modelling_technique]_Morans_I_[species_code]([species name)].txt.

glm_diagnostic_plots_[species code]([_species name]).pdf For GLMs a series of diagnostic plots (residuals vs. fitted values, normal Q-Q, scale-location, residuals vs. leverage) are stored in this file.

11 Function `SDMapsInterpol()`

This function can be used when not a regression model but a spatial interpolation of the data is desired. In some cases this can be as efficient as the regression approach (e.g.)

11.1 Function arguments

SDMdata The data object generated with `data2SDMaps()`.

data.type Character. As for function `SDMaps()` (see section 9.2); choose between "`presence`", "`count`", "`density`", "`normal`", or "`trendindex`".

interpol.method Character. Choose between Inverse Distance Weiging Interpolation (`"IDW"`) and Kriging (`"krige"`).

strat.interpol Character. As for function `SDMaps()`, see section 9.2.

int.args List with arguments that can be passed to the functions `krige()` and `idw()` from R-package `gstat`, for example, `int.args = list(maxdist = 10000, block = c(2000, 2000))`, Here, only the most relevant arguments are presented; for a complete list of possible arguments see the manual of `gstat`. Default is `NULL`.

nmax Numeric: The number of nearest observations that should be used. Default is `Inf` (all observations are used).

maxdist Numeric: Only observations within a distance of maxdist from the prediction location are used. Default is `Inf` (all observations are used).

nmin Numeric: If the number of nearest observations within distance maxdist is less than nmin, a missing value will be generated. Default is `0`.

block Numeric: A vector with two values containing the size of a rectangle in x- and y- dimension. Default is the scale of the prediction grid (Henk oder Caspar fragen!).

spec.subs Character. vector with species (e.g. `spec.subs = c("Fox", "Badger")`) that should be used in the interpolation run. If left unspecified, it will run for all species you included to your plot data (see `plot.data` in section 7.1). Default is `NULL.`

year.subs Character. vector of years (e.g. `year.subs = c("2008", "2009")`) that should be used in the interpolation run. If left unspecified, all years that you provided in your plot data will be included (see `plot.data` in section 7.1). Default is `NULL.`

plots Logical. Make pictures of resulting maps? Default is `TRUE.`

driver Character. Use `"tif"`, `"asc"` or `"shp"` (tif = geotiff-grid, asc = ESRI ASCII-grid, shp = ESRI shapefile) or a combination of them (e.g. `c("asc", "tif")`, in which case maps will be produced in both formats).

path.to.contour Character. Specify the path to the shape-file (see section 4.4) including its file name. If no path is specified, `SDMaps` looks for a file named contour.shp in the working directory. If this file is not available neither, then a warning is displayed and the maps are drawn without contours or additional features such as national borders, rivers etc. *Attention*: The location plot (see section 9.4) cannot be drawn without the contour file!

path.to.speciesnames.file Character. Path to the directory where file `speciesnames.txt` is stored. The file permits the use of full species names in file names and graphics instead of codes (e.g. EURING numbers). It must consist of two comma-separated columns named 'code' and 'species', for example:

```
code, species
5320, Limosa limosa
5460, Tringa totanus
5610, Arenaria interpres
```

out.dir Character. Specify output directory.

lowest.value.white Logical. Should lowest value in the map be white? Default is `FALSE` (lowest value is shown as light blue).

11.2 Products delivered by `SDMapsInterpol()`

The function saves following files in the output directory specified by argument `out.dir` (species names are only included in the file names when the file speciesnames.txt is provided, see section 4.5):

1. a png-plot of for each species with the locations of their observations, distinguishing between positive and negative ("zeroes") observations, named `Locations_[species code](_[species name]).png`;
2. a png-plot of the interpolated map named `[interpol.method]_[species code](_[species name]).png`;
3. data files in the formats specified by argument `driver` bearing the names `[interpol.method]_[species code](_[species name]).[driver]`.

12 Function `SDMapsMakeMaps()`

The production of the maps is not necessarily part of a `SDMaps` modelling run. You can set the `SDMaps` output option `plots` to `FALSE`, in which case only the ASCII grids of the resulting presence-absence or abundance predictions will be stored but no maps will be produced. With function `SDMapsMakeMaps()` you can create the maps afterwards, for example, for selected species or selected prediction types (raw model predictions or predictions corrected by interpolated residuals). While during a `SDMaps()` run maps are drawn with default values for map size, colour scheme, font size etc., `SDMapsMakeMaps()` allows for more flexibility.

12.1 Function arguments

SDMres The results object of a `SDMaps` modelling run or the path to a stored results object. Typically you save a modelling run under a certain name; here you provide that name (see example below in section 17 Using scripts)

species Character: A vector of species codes or species names (if they have been provided in the file speciesnames.txt, see section 4.5) of species for which maps shall be made.

maps Character: A vector of prediction types ("predicted", "predictedr", ...) that shall be used for making maps. Look up result file `preds_[modelling technique]_[analysis type]_[species code](_[species name]).RData` or `.txt` and see which predictions were produced in the modelling run.

path.to.contour Character. Specify the path to the shape-file (see section 4.4) including its file name. If no path is specified, `SDMaps` looks for a file named contour.shp in the working directory. If this file is not available neither, then a warning is displayed and the maps are drawn without contours or additional features such as national borders, rivers etc.

titles Character: A vector with title and subtitle. Default is `NULL` (= the standard titles will be used).

cex.titles Numeric: A vector with the font size of title and subtitle using `cex.main` and `cex.sub`. Default is `c(3, 3)`.

format Character: "`png`" or "`pdf`". Default is "`png`".

size.png Numeric: Vector with height and size for a png map in pixels. Default is `c(1200, 1200)`.

size.pdf Numeric: Vector with height and size for a pdf map in inches. Default is `c(7, 7)`.

pointsize Numeric: pointsize of plotted text; applies to png maps only. Default is `14`.

cex.axis Numeric: Size of the legend labels. Default is `2`.

colours Character: colour scheme for the map. The standard palettes "`rainbow`", "`heat.colors`", "`terrain.colors`", "`topo.colors`" and "`cm.colors`" can be used. When a vector with single colours is specified (maximum four), a gradient is established by interpolating these colours, for example `c("white", "yellow", "red")`. If a specific colour scheme is required, also RGB colour codes can be used and be specified as a list with three vectors for R, G and B, for example `colours = list(R = c([...]), G = c([...]), B = c([...]))`. This argument currently applies only to count or density maps. Default is NULL (= the `SDMaps` standard palette is used).

legend.shrink Numeric: Determines the size of the legend in relation to the map. Default is `0.8`.

legend.space Numeric: Space to the right of the legend. Depends on the number of decimals used for the legend labels, perhaps you have to play around to find the optimal value. Default is `1`.

legend.type Character: Character: One of "`Jenks`", "`linear`", "`geometric`" or "`quantiles`". For "`Jenks`" the Jenks natural breaks optimization algorithm (Jenks, 1967) is applied to find the optimal classification of values for visualisation. In general Jenks does best, but in specific cases it can be useful to choose another classification. This argument applies only to count or density maps. Default is "`Jenks`".

n.classes Numeric: Number of classes that have to be used for classification and legend. Default is `10`. In combination with `legend.type = quantiles`, either a number of quantiles can be chosen (for example, `n.classes = 4` divides the data in quartiles) or a vector with freely chosen percentiles can be provided (for example, `n.classes = c(0.20, 0.45, 0.75, 0.9)`).

n.decimals Numeric or character: Number of decimals to be used for the legend labels. Default is `1`. If option 'pretty' is specified, the legend labels are rounded to two significant numbers to avoid pseudo-precision, dependent on the size of the number. For example, 21345 will be rounded to 21000, 145.23 to 150 and 1.2537 to 1.3.

ticks.at.class.limits Logical: Should the legend labels be placed precisely at the class limits (that is, at the breaks defined by legend.type)? In this case the labels give very precise numbers. Default is `FALSE`.

from.min Logical: If TRUE the map legend is not shown from 0 to the maximum value, but from the smallest value in the map to the maximum value. Default is `FALSE`.

12.2 Products delivered by `SDMapsMakeMaps()`

If argument `SDMres` is used, the new figures are stored in the same directory as the products delivered in the respective `SDMaps()` run; otherwise they are stored in `OutDir`. If `SDMaps()` argument `plots` was set to `TRUE`, the already existing maps are not overwritten by the new maps. Instead, the new files receive the index "2". For example, if a map 'predicted_count_speciescode_speciesname.png' has been generated during the `SDMaps()` run, the new map made with `SDMapsMakeMaps()` is named 'predicted_count_speciescode_speciesname_2.png'.

13 Function SDMapsPostHocPlots()

Typically, for each abundance map the optimal legend is used, determined by a special algorithm and depending on minimum and maximum values in the map and on the number of discrete predictions. This is a nice feature for single maps, but can hamper the comparison of different maps, because same colours may represent different abundance. Function SDMapsPostHocPlots() can be used to plot selected maps with an identical colour legend to facilitate comparison among them. The function can be used in two different ways: 1) Provide a SDMaps results object and choose species and map types from the results directory specified therein (arguments SDMres, species, maps, see below). This approach is suitable for making comparisons between several species modelled in the same modelling run. 2) Provide complete paths to saved ascii files (see products of function SDMaps() in section 9.4.6). The ascii files may be stored in different directories. This approach is suitable to compare different modelling outputs (e.g. RF against GBMs) for one or more given species.

Remark: In a future version of SDMaps, this function will probably be merged with function SDMapsMakeMaps().

13.1 Function arguments

SDMres The results object of a SDMaps modelling run or the path to a stored results object. Typically you save a modelling run under a certain name; here you provide that name (see example below in section 17 Using scripts)

species Character: A vector of species codes or species names (if they have been provided in the file speciesnames.txt, see section 4.5) of species for which maps shall be made.

maps Character. A vector of prediction types ("predicted", "predictedr", ...) that shall be used for making maps. Look up result file preds_[modelling technique]_[analysis type]_[species code](_[species name]).RData or .txt and see which predictions were produced in the modelling run.

paths.to.maps Character: A vector of paths to the ascii files that are to be drawn as png maps.

autocorr Character: A vector with information on which approach for tackling spatial autocorrelation was adopted for the map(s). It is of length 1 when argument SDMres is used to specify an SDMaps results object; it must must have the same length as argument paths.to.maps when single paths to maps are specified. Options are "none", "regression_kriging", "RAC" or "RFsp". Default is NULL.

outDir Character: Path to the directory where the maps are to be stored. The directory can be freely chosen and will be created by the function.

path.to.contour Character. Specify the path to the shape-file (see section 4.4) including its file name. If no path is specified, SDMaps looks for a file named contour.shp in the working directory. If this file is not available neither, then a warning is displayed and the maps are drawn without contours or additional features such as national borders, rivers etc.

cex.titles Numeric: A vector with the font size of title and subtitle using cex.main and cex.sub. Default is c(3, 3).

format Character: "png" or "pdf". Default is "png". (Currently, only "png" is supported.)

size.png Numeric: Vector with height and size for a png map in pixels. Default is c(1200, 1200).

size.pdf Numeric: Vector with height and size for a pdf map in inches. Default is c(7, 7).

pointsize Numeric: pointsize of plotted text; applies to png maps only. Default is 14.

cex.axis Numeric: Size of the legend labels. Default is 2.

legend.shrink Numeric: Determines the size of the legend in relation to the map. Default is 0.8.

legend.space Numeric: Space to the right of the legend. Depends on the number of decimals used for the legend labels, perhaps you have to play around to find the optimal value. Default is 1.

legend.type Character: One of "`Jenks`", "`linear`" or "`geometric`". For "`Jenks`" the Jenks natural breaks optimization algorithm [Jenks, 1967] is applied to find the optimal classification of values for visualisation. In general Jenks does best, but in specific cases it can be useful to choose a linear or geometric classification. This argument applies only to count or density maps. Default is "`Jenks`". (Currently, only "`Jenks`" is supported.)

n.classes Character: Number of classes that have to be used for classification and legend. Default is `15`.

n.decimals Character: Number of decimals to be used for the legend labels. Default is `1`.

13.2 Products delivered by `SDMapsPostHocPlots()`

The maps are stored in the defined `OutDir`. Since the ascii files used for making new maps may be reside in different directories and thus may have identical file names (e.g. predicted_density_speciesA.asc), the entire path will be used for making the filenames of the resulting png's (which thus can be rather bulky) to differentiate between them.

14 Function `SDMapsApplyModel()`

This functions allows to apply an existing model to new prediction data. When observation related covariates such as time of day, daynumber or observer quality are used, maps can be produced with it for different times of day, daynumbers or observers of different quality without the necessity to run the entire modelling process again.

14.1 Function arguments

results.dir Character. Path to the directory where the model object `[modelling technique]_[analysis type]_[species code](_[species name]).RData` is stored.

model.object Character. Name of the model object which has to be applied to new prediction data. It must be present in `results.dir`.

sdmdata.analysis Character. Name of the results object of the `SDMaps()` run; by default bears the name `sdmdata_analysis.RDATA`. It must be present in `results.dir`.

point.preds Character. Name of the point predictions file `PointPredictions_[modelling technique]_[analysis type]_[species code](_[species name]).txt`. It must be present in `results.dir`.

new.sdmdata The new sdmdata in the workspace that have been generated with function `data2SDMaps()` and the new covariate grids. Alternatively, also a path to new sdmdata stored on the hard disk can be specified.

path.to.contour Character. Specify the path to the shape-file (see section 4.4) including its file name. If no path is specified, `SDMaps` looks for a file named contour.shp in the working directory. If this file is not available neither, then a warning is displayed and the maps are drawn without contours or additional features such as national borders, rivers etc.

path.to.speciesnames.file Character. Path to the directory where file `speciesnames.txt` is stored. The file permits the use of full species names in file names and graphics instead of codes (e.g. EURING numbers). It must consist of two comma-separated columns named 'code' and 'species', for example:

```
code, species
5320, Limosa limosa
5460, Tringa totanus
5610, Arenaria interpres
```

results.subdir Character. Name of the subdirectory that has to be generated in the `results.dir` and where the new maps will be stored. If none is provided, directories with the name `maps1`, `maps2`, ... are created. Default is `NULL` (no name provided)

save.text.files Character. Define which objects you want to save on the hard disk as text files. See 9.3. Default is `NULL` (no text files are saved on the disk).

14.2 Products delivered by `SDMapsApplyModel()`

The function writes png maps and ascii files to the hard disc. Also a `preds_[modelling technique]_[analysis type]_[species code](_[species name]).RData` object is saved. See 9.4 for more information on these files.

15 Function SDMapsEnsemble()

Ensemble models ("model averaging") often outperform their single ensemble members. This function allows to make ensemble predictions for count and density models. The ensembles may consist of models of the same type (e.g., all ensemble members are random forest models) or of different types (e.g., an ensemble consisting of a GLM, a random forest and a MARS model).

15.1 Function arguments

species Character: For which species has an ensemble prediction to be made?

path.to.models Character: A vector of paths to the directories where the results of the single models are stored.

ensemble.out.dir Character: Path to the directory where the ensemble maps are to be stored. If not yet existing, the directory will be created.

path.to.contour Character. Specify the path to the shape-file (see section 4.4) including its file name. If no path is specified, `SDMaps` looks for a file named contour.shp in the working directory. If this file is not available neither, then a warning is displayed and the maps are drawn without contours or additional features such as national borders, rivers etc.

pred.type Character: One of `"predictedr"` (raw model predictions) or `"predicted"` (model predictions + interpolated residuals).

method Character: One of `"average"`, `"metrics"` and `"modelled"`. For `"average"` the new predictions are calculated as the simple mean of the original models. Even a simple average often outperforms the single ensemble members[Rougier, 2016]. For `"metrics"`, the statistics of the single models chosen for the ensemble (currently MAE, RMSE and the correlation ceofficient r between observations and fitted values) are converted into model weights by the following procedure: (1) The error statistics (MAE and RSME) are expressed as the reciprocal of the multiple of the smallest error (thus, the smallest MAE and RSME are transformed to 1, larger errors are transformed to values < 1). (2) All correlation coefficients r are divided by the largest one (the largest r thus is transformed to 1, all smaller ones to values < 1). (3) The weighing factors for the single models are calculated by Euklidean distance (ED) of the combined statistics from the origin via $ED = \sqrt{MAE'^2 + RMSE'^2 + r'^2}$ where MAE', RSME' and r' are the transformed statistics. (4) For each cell in the grid the predictions by the single models are multiplied with the weighing factor, summed over the models and divided by the sum of the weights, thus producing the ensemble prediction as the weighted mean of the single models. For `"modelled"`, a neural net and a random forest try to map the point predictions of the original models (= the predictions for the locations where the observations were made) on training data (75 % of the observations). With the testdata (the 25 % set-aside data) it is evaluated which one is better. Subsequently the better model is used to convert the original predictions into ensemble predictions for the entire prediction grid.

main, xlab, ylab Plotting arguments for scatterplots; when none are provided, default title (averaging method) and axis labels ("observed" and "fitted") are used.

lowest.value.white Logical. Should lowest value in the map be white? Default is `FALSE` (lowest value is light blue).

15.2 Products delivered by `SDMapsEnsemble()`

The function writes a png map and an ascii file with the ensemble predictions to the hard disk. A text file documents the processing of the original model stats to the weighing factors (for `method = "metrics"`) and the training of the neural net and the random forest (for `method = "modelled"`), respectively. Statistics on the fit of the predictions with the observations are reported (MAE, MFE, RMSE, correlation coefficient and regression statistics of observed regressed on fitted values); in case of modelled ensemble predictions the statistics describe the fit between predictions and observations in the test data only.

16 Function `SDMapsCompareMaps()`

This function allows to compare maps with the same extent and resolution with the Structural Similarity (SSIM) index according to the approach of Jones et al. [2016]. This index applies a local moving window to calculate statistics based on local mean, variance and covariance between the compared maps.

16.1 Function arguments

paths.to.maps Character. Complete path to the maps that have to be compared, including their file name.

path.to.contour Character. Specify the path to the shape-file (see section 4.4) including its file name. If no path is specified, `SDMaps` looks for a file named contour.shp in the working directory. If this file is not available neither, then a warning is displayed and the maps are drawn without contours or additional features such as national borders, rivers etc.

output.map.name Character. File name of the maps visualising the differences in local mean, variance and covariance.

outDir Character. Path to the directory where the comparison information is to be be saved. The directory can be freely chosen and will be created by the function.

w Integer. This argument defines the size of the moving window. It is transformed into a square neighbourhood with area $(2w+1)^2$. `w = 2` thus defines a square of $(2*2+1)^2 = 5 \times 5$ cells. Internally, the cells in the neighbourhood are weighted by a Gaussian weighting function obtained from a Gaussian kernel centred on the focal cell [Wang et al., 2004]. Default is 1.

masked Logical. Should areas where both species are absent be masked? Default is `FALSE`.

dec Character. Tables stored by `SDMapsCompareMaps()` can conveniently be inspected with a spread sheet. Specify if you use "`.`" or "`,`" as decimal symbol. Default is "`.`".

16.2 Products delivered by `SDMapsCompareMaps()`

When only two maps are to be compared, four comparison maps (local mean, local variance, local covariance and the product of the three which serves as an overall measure for comparison) are stored in the defined `outDir` as ascii grids, along with their visualisation as a png figure and a csv file containing the numerical values of the statistics. For three or more compared maps, no comparison maps are produced since the number of pairwise comparisons grows as the binomial coefficient of n over 2 (3 comparisons for 3 maps, 6 for 4, 10 for 5, ...). The statistics of all pairwise comparisons are stored in four csv files, one for local mean, local variance, local covariance and local overall measure each. All files bear the name defined in `output.map.name` with the corresponding extension.

17 Using scripts

Due to the large number of arguments for `data2SDMaps()` and `SDMaps()` it is advised to use scripts and to invoke `R` via Tinn-R, emacs-ESS, RStudio or other user interface instead of working on the `R` command line. Here comes a dummy example code which shows how a script might be composed. Check also the real examples shipped with `SDMaps` and work through them!

First define your working directory, the directory where the model results will be stored, and the coordinate reference system. Then load `SDMaps` into the `R` working space with the command `library(SDMaps)` if you have installed the package `SDMaps` (not yet available for Windows) or use the command `source()` to load the source code.

```
## set your working directory
setwd("/home/user/SDMaps/example") ## on Linux
setwd("C:/user/SDMaps/example")    ## on Windows
## set a directory for where you want to store the results
outdir <- "/home/user/SDMaps/example/results/"
## create this directory
dir.create(outdir)
## set coordinate reference information of the data
crs <- "+proj=sterea +lat_0=46 +lon_0=25 +k=0.99975
   +x_0=500000 +y_0=500000 +ellps=krass +units=m +no_defs"
## alternatively you could use choose.crs() and select
## your own coordinate reference system from a pop-up menu
## load SDMaps functions into the workspace
source("/home/user/SDMaps/SDMaps.r")
## if package SDMaps is already installed you can load it by
## library(SDMaps)
```

Then make the sdmdata object. Check the list of arguments for the function `data2SDMaps()` and their defaults. Many of them you will not have to specify explicitly, because the defaults meet your needs. Save the sdmdata object in the working directory. If you do so you need not to make a new SDMata object each time you want to repeat the analysis (or add new ones) but can load it into the `R` workspace with the command `load()`.

```
## make a sdmdata object
sdmdata <- data2SDMaps(
   named = "sdmdata", ## name of the data (for saving
                      ## purposes)
   outdir = outdir,   ## name of output directory
   plot.data = "lancol.csv", # name of the plot data
   obs.dir = NULL,    ## NULL in this case because
                      ## observations are included in
                      ## plotdata
   crs = crs,         ## coordinate reference
   user.maps = TRUE,  ## use your own maps
   user.dir = "/home/user/SDMaps/example/grids10km",
                      ## where to find the covariate maps
   user.all.question = FALSE, ## use all the maps in
                              ## user.dir
   add.zeroes = FALSE, ## do not use observations with
                      ## zeros
   user.crs = crs     ## the prediction grid uses the
                      ## same coordinate reference
                      ## system as the plot data
   )
```

```
## save the sdmdata object
save(sdmdata, file = "sdmdata.example.RData")
```

Now use the sdmdata object for making your analyses. Again check the different arguments and their defaults! Use the possibility to define specific directories that store the results of different modelling runs! If you save the modelling runs in the R workspace, you can easily continue with the model evaluation using function `SDMapsSummary()`.

```
## load sdmdata object if not already present in workspace
load("sdmdata.example.RData")
## run model
SDM.glm <- SDMaps(
   sdmdata = sdmdata,
   out.dir = paste(outdir, "glm", sep="/"), ## where to
                       ## store the results of this model
   vars.subs=c("-YEAR","-gr04","-bio01","-bio05","-bio06",
        "-bio12","-prec","-tmean","-etp"),
                       ## exclude some covariates from
                       ## analysis
   driver="asc",       ## store output maps as ascii-grids
   selectAIC = "step"  ## use stepwise model selection
   )
## save analysis object
save(SDM.glm, file = "SDM.glm.Rdata")
## run another model with the same data
SDM.gbm <- SDMaps(
   sdmdata = sdmdata,
   model.type = "gbm",
   out.dir = paste(outdir, "gbm", sep="/"), ## where to
                       ## store the results of this model
   vars.subs=c("-YEAR","-gr04","-bio01","-bio05","-bio06",
        "-bio12","-prec","-tmean","-etp"),
                       ## exclude some covariates from
                       ## analysis
   driver="asc",       ## store output maps as ascii-grids
   )
## save analysis object
save(SDM.gbm, file = "SDM.gbm.Rdata")
## evaluate analysis object
SDMapsSummary(SDM.gbm)
```

The directory structure of the above sample analysis would be like this:

```
/home/user/SDMaps/          ## SDMaps is stored here
/home/user/SDMaps/example/        ## working directory
/home/user/SDMaps/example/grids10km/  ## here the grid
                    ## maps of the covariates are stored
/home/user/SDMaps/example/results/  ## general results
                                      ## directory
/home/user/SDMaps/example/results/glm/
/home/user/SDMaps/example/results/gbm/    ## specific
          ## results directories for GLMs and GBMs
```

Within the specific result directories you will find directories for the single modelling runs. If you run three glm models (for example, each with different `SDMaps()` arguments) their results are stored in directories within

/home/user/SDMaps/example/results/glm/ consecutively named RESULTS, RESULTS1 and RESULTS2. Do not underestimate the number of models you may run for a single dataset and maintain a well-structured directory tree!

Within the RESULTS directories you will find the files generated by function SDMaps() (see section 9.4 Products delivered by SDMaps).

18 Using the SDMaps GUI

SDMaps is provided with a simple graphical user interface (GUI). It can be started separately for each of the functions data2SDMaps(), SDMaps(), SDMapsSummary() and SDMapsInterpol() by typing the commands GUI.data2SDMaps(), GUI.SDMaps(), GUI.SDMapsSummary() and GUI.SDMapsInterpol() on the R command line. In the GUI all function arguments are set on their default values. They can be changed by means of droplists or by entering text.

Note that all text entered must be quoted (for example, 'NULL', 'FALSE', 'A:/my_home/my_SDMaps_results'). Character arguments have thus to be double-quoted (for example, '"quasi"', '"presence"'). Exceptions are the specification of the sdmdata object (since it is an object in the workspace) and the arguments passed to gbm.control, rf.control etc.; they have to be written *without* quotes. For example, to set the number of trees in a random forest to 1000 you have to enter rfSDMOptions(ntree=1000) in field rf.control.

Currently, all paths and file names have to entered manually. In future SDMaps versions a browser for selecting files and directories will be made available.

Part IV
Utility-scripts and troubleshooting

19 Utility-scripts

For the data-preparation and fine-tuning of the maps a number of utility-scripts have been made. These scripts can be found in the folder 'Useful R-scripts' shipped with SDMaps. In most cases these scripts will have to be adjusted for they can be used and may require quite a bit of R-experience before they will work properly in your case. In the folder the following scripts are present:

19.1 Scripts to create and manipulate grids

dbf2grid.r transform a dbf-file with covariables into a separate grid for each covariable. In most cases spatial overlays in GIS-systems will result in a dbf-table. These dbf-tables have to be transformed into ascii-grids before they can be used within SDMaps. This scripts facilitates this procedure. This script uses dbf-files where all covariables are in one column (like 'VARIABLE') and the values in another (like 'VALUE). A typical example would be the output of union or intersect of two polygon shape-files. Note that the x- and y-coordinates in the table have to be in an orthogonal grid: an overlay from a polygon-shapefile that is not perfectly rectangular (like most UTM-grids) can NOT be transformed into an ascii-grid.

xtab2grid.r like dbf2grid, this scripts transforms a dbf-table with covariables to separate ascii-grids. Here each variable has to be a separate column in the dbf-file, comparable to a crosstable in a spreadsheet. A typical example would be the result of a tabulate-area table as a result from the overlay of a polygon-grid with a factorial grid (like land use).

Aggregate_grid aggregate a set of ascii-grids to a larger scale, for example from a 250m-grid to a 1km-grid.

Clip_grid clip a set of grids with an polygon shapefile. With this script a selection of the grid, like a province from a national grid, can be clipped.

Crop_grid extract a rectangular part of a set of grids

19.2 Other scripts

csv_import_generic import all csv-files in a directory to dbf-files

print_maps_environmental_grids.r with this script maps can be made of all ascii-grids in a folder. This is convenient if one want to make maps from all covariables.

20 Troubleshooting

There are some simple rules that, if followed, will prevent some of the most common causes for having problems with `SDMaps`:

1. Do not use spaces in species codes and covariate names! Use the underscore instead!
2. Use filenames and column headers for plot data, speciesnames, species-specific covariates etc. exactly as described in 4.1, 5.1, 4.5 and 9.1.5!
3. Try to avoid missing data in the plot-specific covariates!
4. Try to avoid missing data in the covariate grids for the prediction grid! For cells with missing data no predictions can be made!

20.1 Dependent packages

`SDMaps` is dependent on many other `R`-packages. Many of these dependent packages change over time and this may result in errors when running `SDMaps`. We try to keep up with changes in other packages, but please let us know if you encounter suddenly appearing errors after an update of the packages on your computer. Currently used packages are shown in Appendix 21.

20.2 No models for presence-only data

`SDMaps` is dependent on Maxent for one of its methods to generate zeroes in the case of presence-only data. Maxent, however, is dependent on the installation of Java on your computer (be sure to have the 64-bits version of Java installed when you have a 64-bits operating system). When you try to run Maxent from `SDMaps` without the installation of Java this will result in strange errors because no zero-observations were generated.

Another reason why presence-only models will not run, is the presence of spaces in the speciescode or speciesname (field “Species”). So use a code like “BUTRUF”, “2880”, or “Buteo_rufinus”, but not “Buteo rufinus”.

Turn off your screen saver when you make use of Maxent for generating zeroes: the screen saver may halt the functioning of Maxent.

20.3 Warnings

Warnings generally don’t influence the functionality of `SDMaps`, but should not be discarded as unimportant. Some warnings may tell you that the distribution maps will be created without contour since no contour-shapefile was found, while others may warn for extreme events in regression results. It is therefore always wise to have at least a quick look at the warnings.

The warning `"NO ZEROES GENERATED FOR..."` implies that this species is removed from the modelling process. This may be caused by the fact that a species is too common and no Maxent-model can be made for this species. Also adding too many variables (>75) may cause Maxent to crash. Another important reason why Maxent may not run, is the screensaver: so turn of your screensaver when you use Maxent! *Attention*: When on a 64-bit computer a 32-bit Maxent version is used, zeroes cannot be generated at all, and you get the warning for all species.

20.4 Errors

Errors are in general a much bigger problem and generally halt the execution of `SDMaps`. But in some cases `SDMaps` will function properly nevertheless (non-fatal errors). However, in all cases should the user of `SDMaps` try to solve errors, since they make the behaviour of `SDMaps` uncertain.

Non-fatal errors An example of a non-fatal error is described below:
`Loading required package: lme4 Error in inDL(x, as.logical(local), as.logical(now), ...): function 'cholmod_l_start' not provided by package 'Matrix'`
Solution: Uninstall the packages `lme4` and `Matrix` by means of the command: `remove.packages(c("lme4", "Matrix"))` and remove the corresponding folders from the R-library (normally in `C:\Program Files\R\library`). Then re-install the packages by means of the command: `install.packages(c("lme4", "Matrix"), repos="http://www`

Fatal errors

'undefined columns selected' The most common reason for this error is the presence of columns (fields) in the file with observations other than 'Species', 'plotID', 'YEAR', 'x', 'y' and covariables like text-fields. Remove these columns from the csv-file or dataframe.

'Error in file(file, ifelse(append, "a", "w")) ': cannot open the connection': This error occurs when `SDMaps` cannot write to the results-directory. This may be solved by creating a new results-directory and referring to this directory in the option `outdir`.

'Error in if (!trivial) { : missing value where TRUE/FALSE needed' : This error occurs when `SDMaps` cannot write the results to maps. The most common reason for the is that the prediction grid holds no valid data, but only "NA"-values or "Inf"-values. These grids are generally the result of models that included variables with only missing values or missing values on all the data points. Another reason for this error is that one or more parameters cannot be estimated due to singularities. Remove these variables or grids or select modelselection (`selectAIC="step"`) to solve this error.

Escape-characters in text-files Text-files resulting from an export from a spreadsheet- or database-programme may put an escape character like ^z (also shown as something odd in a text-editor) at the end of file. This results in errors like 'missing values' not enough values, etc.

Part V
Appendix

21 R-packages and foreign R-code used by SDMaps

21.1 R-packages

AICcmodavg (>= 2.1.1) Mazerolle [2017]

automap (>= 1.0.14) Hiemstra et al. [2008]

cairoDevice (>= 2.25) Lawrence [2018]

corrgram (>= 1.13) Wright [2018]

car (>= 3.0.2) Fox and Weisberg [2011]

evtree (>= 1.0.7) Grubinger et al. [2014]

fields (>= 9.6) Nychka, D. et al. [2017]

filehash (>= 2.4.1) Peng [2006]

gam (>= 1.14.4) Hastie [2017]

gbm (>= 2.1.5) Greenwell et al. [2019]

GSIF (>= 0.5.4) Hengl [2017]

gstat (>= 1.1.6) Pebesma [2004], Gräler et al. [2016]

gWidgets (>= 0.0.54) Verzani [2014]

gWidgetsRGtk2 (>= 0.0.86) Lawrence and Verzani [2018]

lattice (>= 0.20.38) Sarkar [2008]

lme4 (>= 1.1.19) Bates et al. [2015]

maptools (>= 0.9.4) Bivand and Lewin-Koh [2018]

mda Leisch et al. [2016]

MASS (>= 7.3.50) Venables and Ripley [2002]

mda (>= 0.4.10) Hastie et al. [2017]

neuralnet (>= 1.33) Fritsch and Günther [2016]

nlme (>= 3.1.137) Pinheiro et al. [2018]

outliers (>= 0.14) Komsta [2011]

party (>= 1.3.1) Hothorn et al. [2006a], Strobl et al. [2007]

pgirmess (>= 1.6.9) Giraudoux [2018]

PresenceAbsence (>= 1.1.9) Freeman and Moisen [2008]

pscl (>= 1.5.2) Zeileis et al. [2008b]

quantregForest (>= 1.3.7) Meinshausen [2006, 2017]

ramps (>= 0.6.14) Smith et al. [2008]

randomForest (>= 4.6.14) Liaw and Wiener [2002]

ranger (>= 0.10.1) Wright and Ziegler [2017]

raster (>= 2.8.4) Hijmans [2018]

rgdal (>= 1.3.6) Bivand et al. [2018]

RGtk2 (>= 2.20.35) Lawrence and Temple Lang [2010]

scales (>= 1.0.0) Wickham [2018]

sp (>= 1.3.1) Bivand et al. [2013]

spatstat (>= 1.58.2) Baddeley et al. [2013, 2015]

usdm (>= 1.1.18) Naimi et al. [2014]

VSURF (>= 1.0.4) Genuer et al. [2015]

21.2 R-code published in scientific papers

- for the use of boosted regression trees in species distribution modelling: Elith et al. [2008]
- for the use of multivariate adaptive regression splines (MARS) in species distribution modelling:
- for applying the Structural similarity index to the comparison of species distribution maps: Jones et al. [2016]

References

O. Allouche, A. Tsoar, and R. Kadmon. Assessing the accuracy of species distribution models: prevalence, kappa and the true skill statistic (tss). *Journal of Applied Ecology*, 43:1223–1232, 2006. doi: 10.1111/j.1365-2664.2006.01214.x.

A.D. Anastasiadis, G.D. Magoulas, and M.N. Vrahatis. New globally convergent training scheme based on the resilient propagation algorithm. *Neurocomputing*, 64:253–270, 2005. doi: 10.1016/j.neucom.2004.11.016.

A. Baddeley, R. Turner, J. Mateu, and A. Bevan. Hybrids of gibbs point process models and their implementation. *Journal of Statistical Software*, 55(11):1–43, 2013. doi: 10.18637/jss.v055.i11.

A. Baddeley, E. Rubak, and R. Turner. *Spatial Point Patterns: Methodology and Applications with R*. London, Chapman and Hall/CRC Press, 2015.

D. Bates, M. Mächler, B. Bolker, and S. Walker. Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, 67(1):1–48, 2015. doi: 10.18637/jss.v067.i01.

R. Bivand and N. Lewin-Koh. *maptools: Tools for Reading and Handling Spatial Objects*, 2018. URL `https://CRAN.R-project.org/package=maptools`. R package version 0.9-4.

R. Bivand, T. Keitt, and B. Rowlingson. *rgdal: Bindings for the Geospatial Data Abstraction Library*, 2018. URL `https://CRAN.R-project.org/package=rgdal`. R package version 1.3-6.

R.S. Bivand, E. Pebesma, and V. Gómez-Rubio. *Applied Spatial Data Analysis with R*. New York: Springer, second edition, 2013.

A.-L. Boulesteix, S. Janitza, J. Kruppa, and I.R. Konig. Overview of random forest methodology and practical guidance with emphasis on computational biology and bioinformatics. *WIREs Data Mining and Knowledge Discovery*, 2:493–507, 2012. doi: 10.1002/widm.1072.

L. Breiman. Random forests. *Mach. Learn.*, 45:5–32, 2001.

L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Belmont: Wadsworth, 1984.

B. Crase, A.C. Liedloff, and B.A. Wintle. A new method for dealing with residual spatial autocorrelation in species distribution models. *Ecography*, 35(10):879–888, 2012. ISSN 1600-0587. doi: 10.1111/j.1600-0587.2011.07138.x. URL `http://dx.doi.org/10.1111/j.1600-0587.2011.07138.x`.

D.R. Cutler, T.C. Edwards Jr., K.H. Beard, A. Cutler, K.T. Hess, J. Gibson, and J.J. Lawler. Random forest for classification in ecology. *Ecology*, 88:2783–2792, 2007. doi: 10.1890/07-0539.1.

A.J. Dobson. *An Introduction to Generalized Linear Models (Second edition)*. London: Chapman & Hall, 2002.

C.F. Dormann, J.M. McPherson, M.B. Araújo, R. Bivand, J. Bolliger, G. Carl, R.G. Davies, A. Hirzel, W. Jetz, W.D. Kissling, I. Kühn, R. Ohlemüller, P.R. Peres-Neto, B. Reineking, B. Schröder, F.M. Schurr, and R. Wilson. Methods to account for spatial autocorrelation in the analysis of species distributional data: a review. *Ecography*, 30(5):609–628, 2007. doi: 10.1111/j.2007.0906-7590.05171.x.

J. Elith, J.R. Leathwick, and T. Hastie. A working guide to boosted regression trees. *Journal of Animal Ecology*, 77:802–813, 2008. doi: 10.1111/j.1365-2656.2008.01390.x.

J. Elith, S.J. Phillips, T. Hastie, M. Dudik, Y.E. Chee, and C.J. Yates. A statistical explanation of maxent for ecologists. *Diversity and Distributions*, 17:43–57, 2011. doi: 10.1111/j.1472-4642.2010.00725.x.

R. Engler, A. Guisan, and L. Rechsteiner. An improved approach for predicting the distribution of rare and endangered species from occurrence and pseudo-absence data. *Journal of Applied Ecology*, 41:263–274, 2004. doi: 10.1111/j.0021-8901.2004.00881.x.

J.S. Evans, M.A. Murphy, Z.A. Holden, and S.A. Cushman. Modeling species distribution and change using random forest. In Drew. C.A., Y. Wiersma, and F. Huettmann, editors, *Predictive Species and Habitat Modeling in Landscape Ecology: Concepts and Applications*, pages 139–159. Springer, New York, 2011. doi: 10.1007/978-1-4419-7390-0{_}8.

J. Fox and S. Weisberg. *An R Companion to Applied Regression*. Sage, Thousand Oaks CA, 2nd edition, 2011. URL `http://socserv.socsci.mcmaster.ca/jfox/Books/Companion`.

E.A. Freeman and G. Moisen. PresenceAbsence: An r package for presence absence analysis. *Journal of Statistical Software*, 23(11):1–31, 2008. URL `http://www.jstatsoft.org/v23/i11`. eafreeman@fs.fed.us.

J.H. Friedman. Multivariate adaptive regression splines. *Annals of Statistics*, 19:1–67, 1991.

J.H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29: 1189–1232, 2001.

S. Fritsch and F. Günther. *neuralnet: Training of Neural Networks*, 2016. URL `https://CRAN.R-project.org/package=neuralnet`. R package version 1.33.

S.I. Gallant. *Neural Network Learning and Expert Systems*. Cambridge: MIT Press, 1993.

R. Genuer, J.-M. Poggi, and C. Tuleau-Malot. Variable selection using random forests. *Pattern Recognition Letters*, 31(14):2225âĂŞ2236, 2010. URL `https://hal.archives-ouvertes.fr/hal-00755489`.

R. Genuer, J.-M. Poggi, and C. Tuleau-Malot. Vsurf: An r package for variable selection using random forests. *R Journal*, 7(2):19–33, 2015. URL `https://journal.r-project.org/archive/2015/RJ-2015-018/RJ-2015-018.pdf`.

P. Giraudoux. *pgirmess: Spatial Analysis and Data Mining for Field Ecologists*, 2018. URL `https://CRAN.R-project.org/package=pgirmess`. R package version 1.6.9.

B. Gräler, E. Pebesma, and G. Heuvelink. Spatio-temporal interpolation using gstat. *The R Journal*, 8: 204–218, 2016. URL `https://journal.r-project.org/archive/2016-1/na-pebesma-heuvelink.pdf`.

W.T.Jr. Grandy and L.H. Schick. *Maximum Entropy and Bayesian Methods*. Kluwer, 1990.

B. Greenwell, B. Boehmke, J. Cunningham, and GBM Developers. *gbm: Generalized Boosted Regression Models*, 2019. URL `https://CRAN.R-project.org/package=gbm`. R package version 2.1.5.

T. Grubinger, A. Zeileis, and K. P. Pfeiffer. evtree: Evolutionary learning of globally optimal classification and regression trees in r. *Journal of Statistical Software*, 61(1):1–29, 2014. doi: 10.18637/jss.v061.i01.

F. Günther and S. Fritsch. neuralnet: Training of neural networks. *R Journal*, 2(1):30–38, 2010. URL `https://journal.r-project.org/archive/2010-1/RJournal_2010-1_Guenther+Fritsch.pdf`.

T. Hastie. *gam: Generalized Additive Models*, 2017. URL `https://CRAN.R-project.org/package=gam`. R package version 1.14-4.

T. Hastie, F. Tibshirani, R. (S original); Original R port by Leisch, K. Hornik, and B.D. Ripley. *mda: Mixture and Flexible Discriminant Analysis*, 2017. URL `https://CRAN.R-project.org/package=mda`. R package version 0.4-10.

T. Hengl. *GSIF: Global Soil Information Facilities*, 2017. URL `https://CRAN.R-project.org/package=GSIF`. R package version 0.5-4.

T. Hengl, H. Sierdsema, A. Radović, and A. Dilo. Spatial prediction of species' distributions from occurrence-only records: combining point pattern analysis, ENFA and regression-kriging. *Ecological Modelling*, 220: 3499–3511, 2009. doi: 10.1016/j.ecolmodel.2009.06.038.

T. Hengl, M. Nussbaum, Wright M.N., G.B.M. Heuvelink, and B. Gräler. Random forest as a generic framework for predictive modeling of spatial and spatio–temporal variables. *PeerJ*, 6, 2017. doi: 10.7717/peerj.5518.

P.H. Hiemstra, E.J. Pebesma, C.J.W. Twenhöfel, and G.B.M. Heuvelink. Real-time automatic interpolation of ambient gamma dose rates from the dutch radioactivity monitoring network. *Computers & Geosciences*, 2008. doi: 10.1016/j.cageo.2008.10.011.

R.J. Hijmans. *raster: Geographic Data Analysis and Modeling*, 2018. URL `https://CRAN.R-project.org/package=raster`. R package version 2.8-4.

A.H. Hirzel, J. Hausser, D. Chessel, and N. Perrin. Ecological-niche factor analysis: How to compute habitat-suitability maps without absence data? *Ecology*, 83:2027–2036, 2002. doi: 10.1890/0012-9658(2002)083[2027:ENFAHT]2.0.CO;2.

K. Hornik, M. Stichcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359âĂŞ366, 1989.

T. Hothorn, K. Hornik, and A. Zeileis. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, 15:651–674, 2006a. doi: 10.1198/106186006X133933.

T. Hothorn, Hornik K., M. A. van de Wiel, and A. Zeileis. A lego system for conditional inference. *The American Statistician*, 60:257–263, 2006b. doi: 10.1198/000313006X118430.

T. Hothorn, K. Hornik, C. Strobl, and A. Zeileis. *Package 'party'*, 2019. URL `https://CRAN.R-project.org/package=party`. R package version 1.3-3.

G.F. Jenks. The data model concept in statistical mapping. *International Yearbook of Cartography*, 7: 186–190, 1967.

E.L. Jones, L. Rendell, E. Pirotta, and J.A. Long. Novel application of a quantitative spatial comparison tool to species distribution data. *Ecological Indicators*, 70:67–76, 2016. doi: 10.1016/j.ecolind.2016.05.051.

C. Kleiber and A. Zeileis. *Applied Econometrics with R*. Springer, 2008.

L. Komsta. *outliers: Tests for outliers*, 2011. URL `https://CRAN.R-project.org/package=outliers`. R package version 0.14.

M. Lawrence. *cairoDevice: Embeddable Cairo Graphics Device Driver*, 2018. URL `https://CRAN.R-project.org/package=cairoDevice`. R package version 2.25.

M. Lawrence and D. Temple Lang. RGtk2: A graphical user interface toolkit for R. *Journal of Statistical Software*, 37(8):1–52, 2010. doi: 10.18637/jss.v037.i08.

M. Lawrence and J. Verzani. *gWidgetsRGtk2: Toolkit implementation of gWidgets for RGtk2*, 2018. URL `https://CRAN.R-project.org/package=gWidgetsRGtk2`. R package version 0.0-86.

F. Leisch, K. Hornik, and B.D. Ripley. *mda: Mixture and Flexible Discriminant Analysis*, 2016. URL `https://CRAN.R-project.org/package=mda`. R package version 0.4-9.

A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2:18–22, 2002. URL `http://CRAN.R-project.org/doc/Rnews/`.

M.J. Mazerolle. *AICcmodavg: Model selection and multimodel inference based on (Q)AIC(c)*, 2017. URL `http://CRAN.R-project.org/package=AICcmodavg`. R package version 2.1-1.

P. McCullagh and J. A. Nelder. *Generalized linear models (Second edition)*. Chapman & Hall, 1989.

N. Meinshausen. Quantile regression forests. *Journal of Machine Learning Research*, 7:983–999, 2006.

N. Meinshausen. *quantregForest: Quantile Regression Forests*, 2017. URL `https://CRAN.R-project.org/package=quantregForest`. R package version 1.3-7.

C. Merow, M.J. Smith, and J.A. Silander, Jr. A practical guide to maxent for modeling species' distributions: what it does, and why inputs and settings matter. *Ecography*, 36:1058–1069, 2013. doi: 10.1111/j.1600-0587.2013.07872.x.

J. Miller. Incorporating spatial dependence in predictive vegetation models: residual interpolation methods. *The Professional Geographer*, 57:169–184, 2005. doi: 10.1016/j.ecolmodel.2006.12.012.

J. Mullahy. Specification and testing of some modified count data models. *Journal of Econometrics*, 33: 341–365, 1986.

B. Naimi, N.A.S. Hamm, T.A. Groen, A.K. Skidmore, and A.G. Toxopeus. Where is positional uncertainty a problem for species distribution modelling? *Ecography*, 37:191–203, 2014.

Nychka, D., Furrer, R., Paige, J., and Sain, S. fields: Tools for spatial data, 2017. URL `www.image.ucar.edu/~nychka/Fields`. R package version 9.6.

S. Pannekoek and A. van Strien. Trim 3 manual (trends & indices for monitoring data), 2005. URL `http://www.cbs.nl/NR/rdonlyres/2E9912EB-534B-4A32-AD22-17A73402C083/0/trim3man.pdf`. R package version 1.04.1.

E.J. Pebesma. Multivariable geostatistics in s: the gstat package. *Computers & Geosciences*, 30:683–691, 2004. doi: 10.1016/j.cageo.2004.03.012.

R.D. Peng. Interacting with data using the filehash package. *R News*, 6(4):19–24, 2006. URL `http://CRAN.R-project.org/doc/Rnews/`.

S.J. Phillips, R.P. Anderson, and R.E. Schapire. Maximum entropy modeling of species geographic distributions. *Ecological Modelling*, 190:231–259, 2006. doi: 10.1016/j.ecolmodel.2005.03.026.

G. Piñeiro, S. Perelman, J.P. Guerschman, and J.M. Paruelo. How to evaluate models: Observed vs. predicted or predicted vs. observed? *Ecological Modelling*, 216:316–322, 2008. doi: 10.1016/j.ecolmodel.2008.05.006.

J. Pinheiro, D. Bates, S. DebRoy, D. Sarkar, and R Core Team. *nlme: Linear and Nonlinear Mixed Effects Models*, 2018. URL `https://CRAN.R-project.org/package=nlme`. R package version 3.1-137.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2019. URL `http://www.R-project.org/`. ISBN 3-900051-07-0.

M. Riedmiller. Rprop - description and implementation details. Technical report, University of Karlsruhe, 1994.

M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 586–591, San Francisco, 1993.

B.D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge: Cambridge University Press, 2008.

J. Rougier. Ensemble averaging and mean squared error. *Journal of Climate*, 29:8865–8870, 2016. doi: 10.1175/JCLI-D-16-0012.1.

D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer, New York, 2008. URL `http://lmdvr.r-forge.r-project.org`. ISBN 978-0-387-75968-5.

R. E. Schapire. The Boosting Approach to Machine Learning: An Overview. In MSRI Workshop on Nonlinear Estimation and Classification, Berkeley, CA, USA, 2001.

B.J. Smith, J. Yan, and M.K. Cowles. Unified geostatistical modeling for data fusion and spatial heteroskedasticity with r package ramps. *Journal of Statistical Software*, 25(10):1–21, 2008. doi: 10.18637/jss.v025.i10.

C. Strobl, A.-L. Boulesteix, A. Zeileis, and T. Hothorn. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, 8:25, 2007. doi: 10.1186/1471-2105-8-25.

M. van Iterson, H.H.H.B.M. van Haagen, and J.J. Goeman. Resolving confusion of tongues in statistics and machine learning: A primer for biologists and bioinformaticians. 12:543–549, 2012. doi: 10.1002/pmic.201100395.

W.N. Venables and B.D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. URL `http://www.stats.ox.ac.uk/pub/MASS4`. ISBN 0-387-95457-0.

J. Verzani. *gWidgets: gWidgets API for building toolkit-independent, interactive GUIs*, 2014. URL `https://CRAN.R-project.org/package=gWidgets`. R package version 0.0-54.

Z. Wang, A.C. Bovik, H.R Sheikh, and E.P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13:600–612, 2004. doi: 10.1109/TIP.2003.819861.

B. Warner and M. Misra. Understanding neural networks as statistical tools. *American Statistician*, 50:284–293, 1996.

Hadley Wickham. *scales: Scale Functions for Visualization*, 2018. URL `https://CRAN.R-project.org/package=scales`. R package version 1.0.0.

K. Wright. *corrgram: Plot a Correlogram*, 2018. URL `https://CRAN.R-project.org/package=corrgram`. R package version 1.13.

M.N. Wright and A. Ziegler. ranger: A fast implementation of random forests for high dimensional data in c++ and r. *Journal of Statistical Software*, 77(1):1–17, 2017. doi: 10.18637/jss.v077.i01.

A. Zeileis, T. Hothorn, and K. Hornik. Model-based recursive partitioning. *Journal of Computational and Graphical Statistics*, 17(2):492–514, 2008a. doi: 10.1198/106186008X319331.

A. Zeileis, C. Kleiber, and S. Jackman. Regression models for count data in R. *Journal of Statistical Software*, 27(8), 2008b. doi: 10.18637/jss.v027.i08.

A.F. Zuur, E.N. Ieno, and C.S. Elphick. A protocol for data exploration to avoid common statistical problems. *Methods in Ecology & Evolution*, 1:3–14, 2010. doi: 10.1111/j.2041-210X.2009.00001.x.