Package 'blockCV'

February 23, 2020

Type Package

Title Spatial and Environmental Blocking for K-Fold Cross-Validation

Version 2.1.1

Date 2020-02-16

URL https://github.com/rvalavi/blockCV

Maintainer Roozbeh Valavi <valavi.r@gmail.com>

Description Creating spatially or environmentally separated folds for cross-validation to provide a robust error estimation in spatially structured environments; Investigating and visualising the effective range of spatial autocorrelation in continuous raster covariates to find an initial realistic distance band to separate training and testing datasets spatially described in Valavi, R. et al. (2019) <doi:10.1111/2041-210X.13107>.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)

Imports raster (>= 2.5-8), sf (>= 0.8-0), progress

RoxygenNote 7.0.2

Suggests knitr, ggplot2 (>= 3.2.1), cowplot, automap (>= 1.0-14), rgeos, future, future.apply, shiny (>= 1.0.3), shinydashboard, geosphere, methods, rmarkdown, testthat, covr

VignetteBuilder knitr

NeedsCompilation no

Author Roozbeh Valavi [aut, cre], Jane Elith [aut], José Lahoz-Monfort [aut], Gurutzeta Guillera-Arroita [aut]

Repository CRAN

Date/Publication 2020-02-23 20:00:02 UTC

R topics documented:

`blockCV` *blockCV: A package for generating spatially or environmentally separated folds for k-fold cross-validation of species distribution modelling.*

Description

Simple random selection of training and testing folds in the structured environment leads to an underestimation of error in the evaluation of spatial predictions and may result in inappropriate model selection (Telford and Birks, 2009; Roberts et al., 2017). The use of spatial and environmental blocks to separate training and testing sets has been suggested as a good strategy for realistic error estimation in datasets with dependence structures, and more generally as a robust method for estimating the predictive performance of models used to predict mapped distributions (Roberts et al., 2017). Package **'blockCV'** provides functions to separate train and test sets using *buffers*, *spatial* and *environmental* blocks (Valavi et al., 2019). It provides several options for how those blocks are constructed. It also has a function that applies geostatistical techniques to investigate the existing level of spatial autocorrelation in the covariates to inform the choice of a suitable distance band by which to separate the data sets. In addition, some visualization tools are provided to help the user choose the block size and explore generated folds. The package has been written with *species distribution modelling* in mind, and the functions allow for a number of common scenarios (including presence-absence and presence-background species data, rare and common species, raster data for predictor variables). Although it can be applied to any spatial modelling e.g. multi-class responses for remote sensing image classification.

Author(s)

Roozbeh Valavi, Jane Elith, José Lahoz-Monfort and Gurutzeta Guillera-Arroita

References

Roberts et al., (2017). Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. Ecography. 40: 913-929.

Telford, R.J., Birks, H.J.B., (2009). Evaluation of transfer functions in spatially structured environments. Quat. Sci. Rev. 28, 1309-1316.

Valavi, R., Elith, J., Lahoz-Monfort, J. J., & Guillera-Arroita, G. (2019). blockCV: An R package for generating spatially or environmentally separated folds for k-fold cross-validation of species distribution models. Methods in Ecology and Evolution, 10(2), 225-232. doi:10.1111/2041-210X.13107.

See Also

spatialBlock, buffering and envBlock for blocking strategies.

---

buffering *Use distance (buffer) around records to separate train and test folds*

Description

This function generates spatially separated train and test folds by considering buffers of the specified distance around each observation point. This approach is a form of *leave-one-out* cross-validation. Each fold is generated by excluding nearby observations around each testing point within the specified distance (ideally the range of spatial autocorrelation, see spatialAutoRange). In this method, the testing set never directly abuts a training presence or absence (0s and 1s i.e. the response class). For more information see the details section.

Usage

```
buffering(
  speciesData,
  species = NULL,
  theRange,
  spDataType = "PA",
  addBG = TRUE,
  progress = TRUE
)
```

Arguments

speciesData	A simple features (sf) or SpatialPoints object containing species data (response variable).
species	Character. Indicating the name of the field in which species data (binary response i.e. 0 and 1) is stored. If speceis = NULL the presence and absence data (response variable) will be treated the same and only training and testing records will be counted. This can be used for multi-class responses such as land cover classes for remote sensing image classification, but it is not necessary. *Do not use this argument when the response variable is continuous or count data.*
theRange	Numeric value of the specified range by which the training and testing datasets are separated. This distance should be in ***metres*** no matter what the coordinate system is. The range can be explored by spatialAutoRange.
spDataType	Character input indicating the type of species data. It can take two values, **PA** for *presence-absence* data and **PB** for *presence-background* data, when species argument is not NULL. See the details section for more information on these two approaches.
addBG	Logical. Add background points to the test set when spDataType = "PB".
progress	Logical. If TRUE a progress bar will be shown.

Details

When working with presence-background (presence and pseudo-absence) data (specified by `spDataType` argument), only presence records are used for specifying the folds. Consider a target presence point. The buffer is defined around this target point, using the specified range (`theRange`). The testing fold comprises the target presence point and all background points within the buffer (this is the default. If `addBG = FALSE` the bacground points are ignored). Any non-target presence points inside the buffer are excluded. All points (presence and background) outside of buffer are used for the training set. The methods cycles through all the *presence* data, so the number of folds is equal to the number of presence points in the dataset.

For presence-absence data, folds are created based on all records, both presences and absences. As above, a target observation (presence or absence) forms a test point, all presence and absence points other than the target point within the buffer are ignored, and the training set comprises all presences and absences outside the buffer. Apart from the folds, the number of *training-presence*, *training-absence*, *testing-presence* and *testing-absence* records is stored and returned in the `records` table. If `species = NULL` (no column with 0s and 1s is defined), the procedure is like presence-absence data. All other types of data (continuous, count or multi-class responses) should be used like this.

Value

An object of class S3. A list of objects including:

- folds - a list containing the folds. Each fold has two vectors with the training (first) and testing (second) indices
- k - number of the folds
- range - the distance band to separated trainig and testing folds)
- species - the name of the species (column), if provided
- dataType - species data type
- records - a table with the number of points in each category of training and testing

See Also

`spatialAutoRange` for selecting buffer distance; `spatialBlock` and `envBlock` for alternative blocking strategies; `foldExplorer` for visualisation of the generated folds.

Examples

```
# import presence-absence species data
PA <- read.csv(system.file("extdata", "PA.csv", package = "blockCV"))
# coordinate reference system
Zone55s <- "+proj=utm +zone=55 +south +ellps=GRS80 +units=m +no_defs"
# make a sf object from data.frame
pa_data <- sf::st_as_sf(PA, coords = c("x", "y"), crs = Zone55s)

# buffering with presence-absence data
bf1 <- buffering(speciesData= pa_data,
                 species= "Species",
```

```
                  theRange= 70000,
                  spDataType = "PA",
                  progress = TRUE)


# import presence-background species data
PB <- read.csv(system.file("extdata", "PB.csv", package = "blockCV"))
# make a sf object from data.frame
pb_data <- sf::st_as_sf(PB, coords = c("x", "y"), crs = Zone55s)

# buffering with presence-background data
bf2 <- buffering(speciesData= pb_data,
                  species= "Species",
                  theRange= 70000,
                  spDataType = "PB",
                  addBG = TRUE, # add background data to testing folds
                  progress = TRUE)

# buffering with no species attribute
bf3 <- buffering(speciesData = pa_data,
                  theRange = 70000)
```

---

envBlock *Use environmental clustering to separate train and test folds*

Description

Environmental blocking for cross-validation. This function uses clustering methods to specify sets of similar environmental conditions based on the input covariates. Species data corresponding to any of these groups or clusters are assigned to a fold. This function does the clustering in raster space and species data. Clustering is done using `kmeans` for both approaches. This function works on single or multiple raster files; multiple rasters need to be in a raster brick or stack format.

Usage

```
envBlock(
  rasterLayer,
  speciesData,
  species = NULL,
  k = 5,
  standardization = "normal",
  rasterBlock = TRUE,
  sampleNumber = 10000,
  biomod2Format = TRUE,
  numLimit = 0,
  verbose = TRUE
)
```

Arguments

rasterLayer	A raster object of covariates to identify environmental groups.
speciesData	A simple features (sf) or SpatialPoints object containing species data (response variable).
species	Character. Indicating the name of the field in which species data (binary response i.e. 0 and 1) is stored. If speceis = NULL the presence and absence data (response variable) will be treated the same and only training and testing records will be counted. This can be used for multi-class responses such as land cover classes for remote sensing image classification, but it is not necessary. *Do not use this argument when the response variable is continuous or count data*.
k	Integer value. The number of desired folds for cross-validation. The default is k = 5.
standardization	Standardize input raster layers. Three possible inputs are "normal" (the default), "standard" and "none". See details for more information.
rasterBlock	Logical. If TRUE, the clustering is done in the raster layer rather than species data. See details for more information.
sampleNumber	Integer. The number of samples from raster layers to build the clusters.
biomod2Format	Logical. Creates a matrix of folds that can be directly used in the **biomod2** package as a *DataSplitTable* for cross-validation.
numLimit	Integer value. The minimum number of points in each category of data (*train_0*, *train_1*, *test_0* and *test_1*). Shows a message if the number of points in any of the folds happens to be less than this number.
verbose	Logical. To print the report of the recods per fold.

Details

As k-means algorithms use Euclidean distance to estimate clusters, the input covariates should be quantitative variables. Since variables with wider ranges of values might dominate the clusters and bias the environmental clustering (Hastie et al., 2009), all the input rasters are first standardized within the function. This is done either by normalizing based on subtracting the mean and dividing by the standard deviation of each raster (the default) or optionally by standardizing using linear scaling to constrain all raster values between 0 and 1.

By default, the clustering is done in the raster space. In this approach the clusters will be consistent throughout the region and across species (in the same region). However, this may result in a cluster(s) that covers none of the species records (the spatial location of response samples), espcially when species data is not dispersed throughout the region or the number of clusters (k or folds) is high. In this case, the number of folds is less than specified k. If rasterBlock = FALSE, the clustering will be done in species points and the number of the folds will be the same as k.

Note that the input raster layer should cover all the species points, otherwise an error will rise. The records with no raster value should be deleted prior to the analysis or another raster layer would be provided.

Value

An object of class S3. A list of objects including:

- folds - a list containing the folds. Each fold has two vectors with the training (first) and testing (second) indices
- foldID - a vector of values indicating the number of the fold for each observation (each number corresponds to the same point in species data)
- biomodTable - a matrix with the folds to be used in **biomod2** package
- k - number of the folds
- species - the name of the species (column), if provided
- records - a table with the number of points in each category of training and testing

References

Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: Data Mining, Inference, and Prediction (2nd ed., Vol. 1). Springer series in statistics New York.

Roberts et al., (2017). Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. Ecography. 40: 913-929.

See Also

`spatialBlock` and `buffering` for alternative blocking strategies; `foldExplorer` for visualisation of the generated folds.

For *DataSplitTable* see `BIOMOD_cv` in **biomod2** package. for clustering.

Examples

```
# load package data
awt <- raster::brick(system.file("extdata", "awt.grd", package = "blockCV"))
# import presence-absence species data
PA <- read.csv(system.file("extdata", "PA.csv", package = "blockCV"))
# make a sf object from data.frame
pa_data <- sf::st_as_sf(PA, coords = c("x", "y"), crs = raster::crs(awt))

# environmental clustering
eb <- envBlock(rasterLayer = awt,
               speciesData = pa_data,
               species = "Species", # name of the column with response
               k = 5,
               standardization = "standard",
               rasterBlock = TRUE)
```

`foldExplorer` *Explore the generated folds*

Description

A function for visualising the generated folds on a map, and allowing interactive exploration of the data in the folds, using the **RStudio Shiny** app.

Usage

```
foldExplorer(blocks, rasterLayer, speciesData)
```

Arguments

`blocks`	An SpatialBlock, EnvironmentalBlock or BufferedBlock object.
`rasterLayer`	A raster object as background map for visualisation.
`speciesData`	A simple features (sf) or SpatialPoints object containing species data (response variable).

Value

An interactive map showing folds and the species data, that can be used to explore folds. Note that this can also be opened in a web browser window. When you return to the R console, press "Esc" to return to the prompt.

See Also

`spatialBlock`, `buffering` and `envBlock`

Examples

```
if(interactive()){

# load package data
awt <- raster::brick(system.file("extdata", "awt.grd", package = "blockCV"))
# import presence-absence species data
PA <- read.csv(system.file("extdata", "PA.csv", package = "blockCV"))
# make a sf object from data.frame
pa_data <- sf::st_as_sf(PA, coords = c("x", "y"), crs = raster::crs(awt))

# spatial blocking by specified range and random assignment
sb <- spatialBlock(speciesData = pa_data,
                   species = "Species",
                   rasterLayer = awt,
                   theRange = 70000,
                   k = 5,
                   selection = "random",
```

```
                    iteration = 100)

foldExplorer(sb, awt, pa_data)

# buffering with presence-absence data
bf <- buffering(speciesData= pa_data,
                species= "Species", # to count the number of presences and absences
                theRange= 70000,
                spDataType = "PA",
                progress = TRUE)

foldExplorer(bf, awt, pa_data)

# environmental clustering
eb <- envBlock(rasterLayer = awt,
               speciesData = pa_data,
               species = "Species",
               k = 5)

foldExplorer(eb, awt, pa_data)

}
```

`rangeExplorer` *Explore spatial block size*

Description

This function assists selection of block size. It allows the user to visualise the blocks interactively, viewing the impact of block size on number and arrangement of blocks in the landscape (and optionally on the distribution of species data in those blocks). **Slide** to the slected block size, and click **Apply Changes** to change the block size.

Usage

```
rangeExplorer(
  rasterLayer,
  speciesData = NULL,
  species = NULL,
  rangeTable = NULL,
  minRange = NULL,
  maxRange = NULL
)
```

Arguments

`rasterLayer`	A raster object as background map for visualisation.

speciesData	A simple features (sf) or SpatialPoints object containing species data (response variable). If provided, the species data will be shown on the map.
species	Character value indicating the name of the field in which the species data (response variable e.g. 0s and 1s) are stored. If provided, species presence and absence data will be shown in different colours.
rangeTable	A data.frame created by spatialAutoRange function containing spatial autocorrelation parameters of all covariates.
minRange	A numeric value to set the minimum possible range for creating spatial blocks. It is used to limit the searching domain of spatial block size.
maxRange	A numeric value to set the maximum possible range for creating spatial blocks. It is used to limit the searching domain of spatial block size.

Details

The only required argument for this function is the rasterLayer. The rest are optional. If the rangeTable is provided, the minimum, maximum and initial ranges for searching the size of spatial blocks will be selected based on the spatial autocorrelation range of covariates. It is also possible to restrict the allowable range of block sizes by using the minRange and maxRanege arguments.

Value

An interactive map with blocks (and optionally species data) superimposed. Note that this can also be opened in a web browser window. When you return to the R console, press "Esc" to return to the prompt.

See Also

spatialBlock; spatialAutoRange for the rangeTable

Examples

```
if(interactive()){

# load package data
awt <- raster::brick(system.file("extdata", "awt.grd", package = "blockCV"))
# import presence-absence species data
PA <- read.csv(system.file("extdata", "PA.csv", package = "blockCV"))
# make a sf object from data.frame
pa_data <- sf::st_as_sf(PA, coords = c("x", "y"), crs = raster::crs(awt))

rangeExplorer(rasterLayer = awt) # the only mandatory input

# add species data to add them on the map
rangeExplorer(rasterLayer = awt,
              speciesData = pa_data,
              species = "Species",
              rangeTable = NULL,
              minRange = 30000, # limit the search domain
              maxRange = 100000)
```

```
}
```

`spatialAutoRange` *Measure spatial autocorrelation in the predictor raster files*

Description

This function provides a quantitative basis for choosing block size. The spatial autocorrelation in all continuous predictor variables available as raster layers is assessed and reported. The function estimates spatial autocorrelation ranges of all input raster layers. This is the range over which observations are independent and is determined by constructing the empirical variogram, a fundamental geostatistical tool for measuring spatial autocorrelation. The empirical variogram models the structure of spatial autocorrelation by measuring variability between all possible pairs of points (O'Sullivan and Unwin, 2010). Results are plotted. See the details section for further information.

Usage

```
spatialAutoRange(
  rasterLayer,
  sampleNumber = 5000L,
  border = NULL,
  speciesData = NULL,
  doParallel = FALSE,
  nCores = NULL,
  showPlots = TRUE,
  degMetre = 111325,
  maxpixels = 1e+05,
  plotVariograms = FALSE,
  progress = TRUE
)
```

Arguments

`rasterLayer`	A raster object of covariates to find spatial autocorrelation range.
`sampleNumber`	Integer. The number of sample points of each raster layer to fit variogram models. It is 5000 by default, however it can be increased by user to represent their region well (relevant to the extent and resolution of rasters).
`border`	A sf or SpatialPolygons object to clip the block based on it (optional).
`speciesData`	A spatial or sf object (optional). If provided, the `sampleNumber` is ignored and variograms are created based on species locations. This option is not recommended if the species data is not evenly distributed across the whole study area and/or the number of records is low.
`doParallel`	Logical. Run in parallel when more than one raster layer is available. Given multiple CPU cores, it is recommended to set it to `TRUE` when there is a large number of rasters to process.

nCores	Integer. Number of CPU cores to run in parallel. If nCores = NULL half of available cores in your machine will be used.
showPlots	Logical. Show final plot of spatial blocks and autocorrelation ranges.
degMetre	Numeric. The conversion rate of metres to degree. This is for constructing spatial blocks for visualisation. When the input map is in geographic coordinate system (decimal degrees), the block size is calculated based on deviding the calculated *range* by this value to convert to the input map's unit (by default 111325; the standard distance of a degree in metres, on the Equator).
maxpixels	Number of random pixels to select the blocks over the study area.
plotVariograms	Logical. Plot fitted variograms. This can also be done after the analysis. It is FALSE by default.
progress	Logical. Shows progress bar. It works only when doParallel = FALSE.

Details

The input raster layers should be continuous for computing the variograms and estimating the range of spatial autocorrelation. The input rasters should also have a specified coordinate reference system. However, if the reference system is not specified, the function attempts to guess it based on the extent of the map. It assumes an unprojected reference system for layers with extent lying between -180 and 180, and a projected reference system otherwise.

Variograms are calculated based on the distances between pairs of points, so unprojected rasters (in degrees) will not give an accurate result (especially over large latitudinal extents). For unprojected rasters, *the great circle distance* (rather than Euclidian distance) is used to calculate the spatial distances between pairs of points. To enable more accurate estimate, it is recommended to transform unprojected maps (geographic coordinate system / latitude-longitude) to a projected metric reference system (e.g. UTM or Lambert) where it is possible. See autofitVariogram from **automap** and variogram from **gstat** packages for further information.

Value

An object of class S3. A list object including:

- range - the suggested range, which is the median of all calculated ranges
- rangeTable - a table of input covariates names and their autocorrelation range
- plots - the output plot (the plot is shown by default)
- sampleNumber
- variograms - fitted variograms for all layers

References

O'Sullivan, D., Unwin, D.J., (2010). Geographic Information Analysis, 2nd ed. John Wiley & Sons.

Roberts et al., (2017). Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. Ecography. 40: 913-929.

Examples

```
# load the example raster data
awt <- raster::brick(system.file("extdata", "awt.grd", package = "blockCV"))

# run the model in parallel
range1 <- spatialAutoRange(rasterLayer = awt,
                           sampleNumber = 5000, # number of cells to be used
                           doParallel = TRUE,
                           nCores = 2, # if NULL, it uses half of the CPU cores
                           plotVariograms = FALSE,
                           showPlots = TRUE)


# run the model with no parallel
range3 <- spatialAutoRange(rasterLayer = awt,
                           sampleNumber = 5000,
                           doParallel = FALSE,
                           progress = TRUE)

# show the result
summary(range1)
```

---

spatialBlock *Use spatial blocks to separate train and test folds*

Description

This function creates spatially separated folds based on a pre-specified distance. It assigns blocks to the training and testing folds **randomly**, **systematically** or in a **checkerboard pattern**. The distance (theRange) should be in **metres**, regardless of the unit of the reference system of the input data (for more information see the details section). By default, the function creates blocks according to the extent and shape of the study area, assuming that the user has considered the landscape for the given species and case study. Alternatively, blocks can solely be created based on species spatial data. Blocks can also be offset so the origin is not at the outer corner of the rasters. Instead of providing a distance, the blocks can also be created by specifying a number of rows and/or columns and divide the study area into vertical or horizontal bins, as presented in Wenger & Olden (2012) and Bahn & McGill (2012). Finally, the blocks can be specified by a user-defined spatial polygon layer.

Usage

```
spatialBlock(
  speciesData,
  species = NULL,
  blocks = NULL,
```

```
  rasterLayer = NULL,
  theRange = NULL,
  rows = NULL,
  cols = NULL,
  k = 5L,
  selection = "random",
  iteration = 100L,
  numLimit = 0L,
  maskBySpecies = TRUE,
  degMetre = 111325,
  border = NULL,
  showBlocks = TRUE,
  biomod2Format = TRUE,
  xOffset = 0,
  yOffset = 0,
  progress = TRUE,
  verbose = TRUE
)
```

Arguments

speciesData	A simple features (sf) or SpatialPoints object containing species data (response variable).
species	Character (optional). Indicating the name of the column in which species data (response variable e.g. 0s and 1s) is stored. This argument is used *to make folds with evenly distributed records*. **This option only works by random fold selection and with binary or multi-class responses** e.g. species presence-absence/background or land cover classes for remote sensing image classification. If speceis = NULL the response classes will be treated the same and only training and testing records will be counted and balanced.
blocks	A sf or SpatialPolygons object to be used as the blocks (optional). This can be a user defined polygon and it must cover all the species points.
rasterLayer	A raster object for visualisation (optional). If provided, this will be used to specify the blocks covering the area.
theRange	Numeric value of the specified range by which blocks are created and training/testing data are separated. This distance should be in **metres**. The range could be explored by spatialAutoRange() and rangeExplorer() functions.
rows	Integer value by which the area is divided into latitudinal bins.
cols	Integer value by which the area is divided into longitudinal bins.
k	Integer value. The number of desired folds for cross-validation. The default is k = 5.
selection	Type of assignment of blocks into folds. Can be **random** (default), **systematic** or **checkerboard**. The checkerboard does not work with user-defined spatial blocks.
iteration	Integer value. The number of attempts to create folds that fulfil the set requirement for minimum number of points in each trainig and testing fold (for each

	response class e.g. *train_0*, *train_1*, *test_0* and *test_1*), as specified by `species` and `numLimit` arguments.
`numLimit`	Integer value. The minimum number of points in each training and testing folds. If `numLimit = 0`, the most evenly dispersed number of records is chosen (given the number of iteration). This option no longer accepts NULL as input. If it is set to NULL, 0 is used instead.
`maskBySpecies`	Since version 1.1, this option is always set to `TRUE`.
`degMetre`	Integer. The conversion rate of metres to degree. See the details section for more information.
`border`	A sf or SpatialPolygons object to clip the block based on it (optional).
`showBlocks`	Logical. If TRUE the final blocks with fold numbers will be created with ggplot and plotted. A raster layer could be specified in `rasterlayer` argument to be as background.
`biomod2Format`	Logical. Creates a matrix of folds that can be directly used in the **biomod2** package as a *DataSplitTable* for cross-validation.
`xOffset`	Numeric value between **0** and **1** for shifting the blocks horizontally. The value is the proportion of block size.
`yOffset`	Numeric value between **0** and **1** for shifting the blocks vertically. The value is the proportion of block size.
`progress`	Logical. If TRUE shows a progress bar when `numLimit = NULL` in random fold selection.
`verbose`	Logical. To print the report of the recods per fold.

Details

To keep the consistency, all the functions use **metres** as their unit. In this function, when the input map has geographic coordinate system (decimal degrees), the block size is calculated based on deviding `theRange` by 111325 (the standard distance of a degree in metres, on the Equator) to change the unit to degree. This value is optional and can be changed by user via `degMetre` argument.

The `xOffset` and `yOffset` can be used to change the spatial position of the blocks. It can also be used to assess the sensitivity of analysis results to shifting in the blocking arrangements. These options are available when `theRange` is defined. By default the region is located in the middle of the blocks and by setting the offsets, the blocks will shift.

Roberts et. al. (2017) suggest that blocks should be substantially bigger than the range of spatial autocorrelation (in model residual) to obtain realistic error estimates, while a buffer with the size of the spatial autocorrelation range would result in a good estimation of error. This is because of the so-called edge effect (O'Sullivan & Unwin, 2014), whereby points located on the edges of the blocks of opposite sets are not separated spatially. Blocking with a buffering strategy overcomes this issue (see `buffering`).

Value

An object of class S3. A list of objects including:

- folds - a list containing the folds. Each fold has two vectors with the training (first) and testing (second) indices
- foldID - a vector of values indicating the number of the fold for each observation (each number corresponds to the same point in species data)
- biomodTable - a matrix with the folds to be used in **biomod2** package
- k - number of the folds
- blocks - SpatialPolygon of the blocks
- range - the distance band of separating trainig and testing folds, if provided
- species - the name of the species (column), if provided
- plots - ggplot object
- records - a table with the number of points in each category of training and testing

References

Bahn, V., & McGill, B. J. (2012). Testing the predictive performance of distribution models. Oikos, 122(3), 321-331.

O'Sullivan, D., Unwin, D.J., (2010). Geographic Information Analysis, 2nd ed. John Wiley & Sons.

Roberts et al., (2017). Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. Ecography. 40: 913-929.

Wenger, S.J., Olden, J.D., (2012). Assessing transferability of ecological models: an underappreciated aspect of statistical validation. Methods Ecol. Evol. 3, 260-267.

See Also

`spatialAutoRange` and `rangeExplorer` for selecting block size; `buffering` and `envBlock` for alternative blocking strategies; `foldExplorer` for visualisation of the generated folds.

For *DataSplitTable* see `BIOMOD_cv` in **biomod2** package

Examples

```
# load package data
library(sf)

awt <- raster::brick(system.file("extdata", "awt.grd", package = "blockCV"))
# import presence-absence species data
PA <- read.csv(system.file("extdata", "PA.csv", package = "blockCV"))
# make a sf object from data.frame
pa_data <- sf::st_as_sf(PA, coords = c("x", "y"), crs = raster::crs(awt))

# spatial blocking by specified range and random assignment
sb1 <- spatialBlock(speciesData = pa_data,
                    species = "Species",
                    theRange = 70000,
                    k = 5,
```

```
                    selection = "random",
                    iteration = 100,
                    numLimit = NULL,
                    biomod2Format = TRUE,
                    xOffset = 0.3, # shift the blocks horizontally
                    yOffset = 0)

# spatial blocking by row/column and systematic fold assignment
sb2 <- spatialBlock(speciesData = pa_data,
                    species = "Species",
                    rasterLayer = awt,
                    rows = 5,
                    cols = 8,
                    k = 5,
                    selection = "systematic",
                    biomod2Format = TRUE)
```

Index