Package ‘fuzzySim’

May 21, 2026

Type Package

Title Fuzzy Similarity in Species Distributions

Version 4.54

Date 2026-05-21

Maintainer A. Marcia Barbosa <ana.marcia.barbosa@gmail.com>

Imports graphics, methods, modEvA (> 3.9), stats, stringi, utils

Suggests aod, geodist, parallel, phylolm, raster, terra, tools

Encoding UTF-8

Description Functions to compute fuzzy versions of species occurrence patterns based on presence-absence data (including inverse distance interpolation, trend surface analysis, and prevalence-independent favourability obtained from probability of presence), as well as pair-wise fuzzy similarity (based on fuzzy logic versions of commonly used similarity indices) among those occurrence patterns. Includes also functions for model consensus and comparison (overlap and fuzzy similarity, fuzzy loss, fuzzy gain), and for data preparation, such as obtaining unique abbreviations of species names, defining the background region, cleaning and gridding (thinning) point occurrence data onto raster maps, selecting among (pseudo)absences to address survey bias, converting species lists (long format) to presence-absence tables (wide format), transposing part of a data frame, selecting relevant variables for models, assessing the false discovery rate, or analysing and dealing with multicollinearity. Initially described in Barbosa (2015) <doi:10.1111/2041-210X.12372>.

License GPL-3

URL http://fuzzysim.r-forge.r-project.org/

BugReports https://github.com/AMBarbosa/fuzzySim/issues

NeedsCompilation no

Author A. Marcia Barbosa [aut],
Alba Estrada [ctb],
Paul Melloy [ctb],
Jose Carlos Guerrero [fnd],
A. Marcia Barbosa [cre]

Depends R (>= 2.10)

Repository CRAN

Date/Publication 2026-05-21 14:00:02 UTC

Contents

`fuzzySim-package` *Fuzzy Similarity in Species Distributions*

Description

Functions to compute fuzzy versions of species occurrence patterns based on presence-absence data (including inverse distance interpolation, trend surface analysis, and prevalence-independent favourability obtained from probability of presence), as well as pair-wise fuzzy similarity (based on fuzzy logic versions of commonly used similarity indices) among those occurrence patterns. Includes also functions for model consensus and comparison (fuzzy overlap and fuzzy similarity, loss or gain), and for data preparation such as obtaining unique abbreviations of species names, cleaning species occurrence records, gridding (thinning) point occurrence data onto raster maps, converting species lists (long format) to presence-absence tables (wide format), transposing part of a data frame, selecting relevant variables for models, assessing the false discovery rate, or analysing and dealing with multicollinearity. Includes also sample datasets for providing practical examples. A step-by-step illustrated tutorial is available from the package homepage (http://fuzzysim.r-forge.r-project.org).

Details

Package:	fuzzySim
Type:	Package
Version:	4.54
Date:	2026-05-21
License:	GPL-3

Author(s)

A. Marcia Barbosa

Maintainer: A. Marcia Barbosa <ana.marcia.barbosa@gmail.com>

References

Barbosa A.M. (2015) fuzzySim: applying fuzzy logic to binary similarity indices in ecology. Methods in Ecology and Evolution, 6: 853-858.

Examples

```
data(rotifers)

head(rotifers)


# add column with species name abbreviations:

rotifers$spcode <- spCodes(rotifers$species, sep.species = "_",
nchar.gen = 1, nchar.sp = 5, nchar.ssp = 0)

head(rotifers)


# convert species list (long format) to presence-absence table
# (wide format):

rotifers.presabs <- splist2presabs(rotifers, sites.col = "TDWG4",
sp.col = "spcode", keep.n = FALSE)

head(rotifers.presabs)


# get 3rd-degree spatial trend surface for some species distributions:

data(rotif.env)

names(rotif.env)

rotifers.tsa <- multTSA(rotif.env, sp.cols = 18:20,
coord.cols = c("Longitude", "Latitude"), id.col = 1)

head(rotifers.tsa)

## Not run:

# get inverse squared distance to presence for each species:

rotifers.isqd <- distPres(rotif.env, sp.cols = 18:20,
coord.cols = c("Longitude", "Latitude"), id.col = 1, p = 2, inv = TRUE)

head(rotifers.isqd)


# get prevalence-independent environmental favourability models
# for each species:

data(rotif.env)

names(rotif.env)

rotifers.fav <- multGLM(data = rotif.env, sp.cols = 18:20,
```

```
var.cols = 5:17, id.col = 1, step = FALSE, trim = TRUE,
Favourability = TRUE)


# get matrix of fuzzy similarity between species distributions:

# either based on inverse squared distance to presence:
rot.fuz.sim.mat <- simMat(rotifers.isqd[ , -1], method = "Jaccard")

# or on environmental favourability for presence:
rot.fuz.sim.mat <- simMat(rotifers.fav$predictions[ , 5:7],
method = "Jaccard")

head(rot.fuz.sim.mat)


# transpose fuzzy rotifer distribution data to compare
# regional species composition rather than species' distributions:

names(rotifers.isqd)

rot.fuz.reg <- transpose(rotifers.fav$predictions, sp.cols = 5:7,
reg.names = 1)

head(rot.fuz.reg)


# get matrix of fuzzy similarity between (some) regions'
# species compositions:

reg.fuz.sim.mat <- simMat(rot.fuz.reg[ , 1:10], method = "Jaccard")

head(reg.fuz.sim.mat)

## End(Not run)
```

---

appendData *Append data*

Description

This function appends the rows of a dataframe 'data2' at the bottom of another dataframe 'data1', using the values in the columns with matching names, and (optionally, by default) filling missing columns with NAs.

Usage

```
appendData(data1, data2, fill = TRUE, add.source = TRUE)
```

Arguments

- `data1` — object inheriting class 'data.frame' (or that can be coerced with 'as.data.frame') to which to append data.
- `data2` — object inheriting class 'data.frame' (or that can be coerced with 'as.data.frame') to append to 'data1', with column names matching those of the corresponding columns in 'data1'. Both datasets can have more columns than those whose names match.
- `fill` — logical, whether the result should keep all columns of 'data1' that are missing in 'data2', filling them with NAs in the rows with no data. The default is TRUE. If set to FALSE, the result will keep only the columns of 'data1' that are also present in 'data2'.
- `add.source` — logical, whether the result should include an additional column saying from which input data frame ('data1' or 'data2') each row came.

Details

This function is asymmetric, i.e. `appendData(data1, data2)` may output different columns than `appendData(data2, data1)`. 'data1' dictates the columns that the result will have. Columns of 'data2' that are not matched in 'data1' are not kept in the output.

Value

This function returns a data frame with all the columns and rows of 'data1', extended with the rows of 'data2' with its values for the columns with matching names in 'data1'. By default, with 'add.source = TRUE', there is also an additional column specifying the source input object. If 'fill' is set to FALSE, the result only carries the columns with matching names in both data frames.

Author(s)

A. Marcia Barbosa

See Also

`rbindlist` in package **data.table**; `rbind.fill` in package **plyr**.

Examples

```
df1 = data.frame(A = 3:1, B = letters[1:3], C = c(1, 0, 1))
df2 = data.frame(A = 4:5, B = letters[5:4])

appendData(df1, df2)

appendData(df1, df2, fill = FALSE)

appendData(df1, df2, fill = FALSE, add.source = FALSE)
```

biasLayer *Compute a bias layer*

Description

Computes a bias layer based on a `terra::SpatVectorCollection` of lines, points and/or polygons that may indicate better surveyed areas, using either distance or rasterizing.

Usage

```
biasLayer(svc, rst = terra::rast(terra::ext(svc), crs = terra::crs(svc)),
  type = "distance", combine = "sum", ...)
```

Arguments

`svc`	An object of class `terra::SpatVectorCollection`.
`rst`	A `terra::SpatRaster` with the desired dimensions and CRS for the output. The default has the spatial extent of 'svc' and the default arguments for `terra::rast()`, but ideally the user should provide a template raster with the desired dimensions.
`type`	A character string indicating the method to use. Options are inverse "distance" (the default, with higher values indicating closer proximity to the input features, i.e. more likely survey); or "rasterize" (where only pixels overlapping the input features are given values higher than zero, indicating feasible survey).
`combine`	A character string to pass as the 'fun' argument to `terra::app()`, indicating the method for combining the resulting raster layers into one. The default is `"sum"`.
`...`	Additional arguments passed to `terra::distance()` or to `terra::rasterize()`, depending on 'type'.

Details

This function can produce a bias layer to use e.g. as the 'bias' argument in `selectAbsences`, or in any software package that can use a layer indicating survey bias (e.g. Maxent). The input points, lines and/or polygons should reflect areas or features where survey is possible (if type = "rasterize") or generally more likely (if type = "distance") for the target species. They can be e.g. roads or other access pathways; natural parks or other usually surveyed areas; occurrence records of the target and/or other species of the same taxon or guild; and/or rivers and streams, for freshwater species.

The function calculates the bias layer by either computing the inverse standardized distance, i.e. one minus `modEvA::range01(`distance), to the each of the features in 'svc'; or by rasterizing those features (depending on 'type'). It then combines the resulting raster layers into one, using the specified 'combine' method.

Value

A `terra::SpatRaster`.

Author(s)

A. Marcia Barbosa

See Also

selectAbsences

Examples

```
## Not run:
  # load mapping packages:

  library(terra)
  library(geodata)


  # get some example layers:

  lux <- vect(system.file("ex/lux.shp", package = "terra"))
  poly <- lux[2, ]
  roads <- osm("Luxembourg", var = "highways", path = tempdir())
  primary_roads <- subset(roads, roads$highway == "primary")
 asphalt_roads <- subset(roads, roads$highway == "primary" | roads$surface == "asphalt")


  # get raster template for extent and resolution:

  elev <- rast(system.file("ex/elev.tif", package = "terra"))


  # plot the layers:

  plot(elev)
  plot(poly, col = "darkgreen", add = TRUE)
  plot(asphalt_roads, lwd = 0.3, add = TRUE)
  plot(primary_roads, add = TRUE)


  # compute and plot bias layers based on these:

  bias_dist <- biasLayer(svc(poly, asphalt_roads, primary_roads),
  rst = elev, type = "distance")

  bias_rast <- biasLayer(svc(poly, primary_roads, asphalt_roads),
  rst = elev, type = "rasterize")

  plot(bias_dist, col = map.pal("plasma"), main = "bias layer: distance")
  plot(poly, density = 20, add = TRUE)
  plot(asphalt_roads, lwd = 0.5, add = TRUE)
  plot(primary_roads, lwd = 1.2, add = TRUE)

  plot(bias_rast, col = map.pal("plasma"), main = "bias layer: rasterize")
```

```
    plot(poly, density = 20, add = TRUE)
    plot(asphalt_roads, lwd = 0.5, add = TRUE)
    plot(primary_roads, lwd = 1.2, add = TRUE)

## End(Not run)
```

---

bioThreat *Biotic threat of a stronger over a weaker species based on their favourability values*

Description

This function takes two vectors of Favourability values at different localities for, respectively, a stronger and a weaker species (e.g., a superior vs. an inferior competitor, or an invasive predator vs. an unadapted native prey), and calculates the level of threat that the former may potentially pose to the latter in each locality.

Usage

```
bioThreat(strong_F, weak_F, character = FALSE, ...)
```

Arguments

`strong_F`	a numeric vector of favourability values (obtained, e.g., with functions `Fav` or `multGLM`) for the stronger species.
`weak_F`	a numeric vector of favourability values for the weaker species. Must be of the same lenght and in the same order as 'strong_F'.
`character`	logical value indicating whether the result should be returned in character rather numeric form. Defaults to FALSE.
`...`	additional arguments to pass to `favClass`, namely the breaks for separating favourability values into low, intermediate and high (see Details).

Details

Based on the notion of "favorableness" by Richerson & Lum (1980), according to which competing species may or may not be able to coexist depending on their relative environmental fitnesses, Acevedo et al. (2010, 2012) and some subsequent studies (e.g. Romero et al. 2014, Munoz et al. 2015, Chamorro et al. 2019) proposed possible biotic interaction outcomes of different combinations of favourability values for two species. Favourability has the advantage, in contrast with other types of potential distribution metrics, of being directly comparable among diferent species, independently of their relative occurrence frequencies (see `Fav`). This function builds on those proposals by including additional possible combinations of higher, intermediate or low favourability values (following Munoz & Real 2006), producing the following classification of biotic threat across a set of analysed localities:

0 ('grey'): areas where favourability is low for at least one of the species (abiotic exclusion), so biotic threat does not apply.

1 ('green'): areas where favourability is high for both species, so they should both be able to thrive and therefore co-occur (sympatric coexistence), hence biotic threat is low.

2 ('yellow'): areas where favourability is high for the weaker species and intermediate for the stronger species, so the level of threat is moderate.

3 ('orange'): areas where favourability is intermediate for both species, so the stronger one potentially prevails and the level of threat is high.

4 ('red'): areas where favourability is high for the stronger species and intermediate for the weaker species, in which case the level of threat is very high (biotic exclusion).

Value

This function returns either an integer or a character vector (following the 'character' argument, which is set to FALSE by default) of the same length as 'strong_F' and 'weak_F', classifying each locality with the level of biotic threat posed by the former on the latter (see Details).

Author(s)

A. Marcia Barbosa

References

Acevedo P., Ward A.I., Real R. & Smith G.C. (2010) Assessing biogeographical relationships of ecologically related species using favourability functions: a case study on British deer. Diversity and Distributions, 16: 515-528

Acevedo P., Jimenez-Valverde A., Melo-Ferreira J., Real R. & Alves, P.C. (2012) Parapatric species and the implications for climate change studies: a case study on hares in Europe. Global Change Biology, 18: 1509-1519

Chamorro D., Munoz A.R., Martinez-Freiria F. & Real R. (2019) Using the fuzzy logic in the distribution modelling of competitive interactions. Poster, IBS Malaga 2019 - 9th Biennial Conference of the International Biogeography Society

Munoz A.R. & Real R. (2006) Assessing the potential range expansion of the exotic monk parakeet in Spain. Diversity and Distributions, 12: 656-665

Munoz A.R., Real R. & Marquez A.L. (2015) Interacciones a escala nacional entre rapaces rupicolas en base a modelos de distribucion espacial. Los casos del buitre leonado, alimoche y aguila perdicera. Informe tecnico, Universidad de Malaga & Fundacion EDP

Richerson P.J. & Lum K. (1980) Patterns of plant species diversity in California: relation to weather and topography. American Naturalist, 116:504-536

Romero D., Baez J.C., Ferri-Yanez F., Bellido J. & Real R. (2014) Modelling favourability for invasive species encroachment to identify areas of native species vulnerability. The Scientific World Journal, 2014: 519710

See Also

sharedFav, Fav, favClass

Examples

```
data(rotif.env)
mods <- multGLM(rotif.env, sp.cols = 19:20, var.cols = 5:17)
head(mods$predictions)
favs <- mods$predictions[ , 3:4]
threat <- bioThreat(strong_F = favs[,1], weak_F = favs[,2])
threat_chr <- bioThreat(strong_F = favs[,1], weak_F = favs[,2], char = TRUE)
data.frame(favs, threat = threat, threat_col = threat_chr)
```

---

cleanCoords *Clean coordinates*

Description

This function takes a data frame with species occurrences and removes the rows whose coordinates do not pass a set of user-specified filters (see Arguments). Row names are inheritted from the input data frame, i.e. if row "2" is cleaned out, output rownames will be c("1", "3", ...).

Usage

```
cleanCoords(data, coord.cols = c("decimalLongitude", "decimalLatitude"),
uncert.col = "coordinateUncertaintyInMeters", abs.col = "occurrenceStatus",
year.col = "year", rm.dup = !is.null(coord.cols),
rm.missing.any = !is.null(coord.cols), rm.missing.both = !is.null(coord.cols),
rm.zero.any = !is.null(coord.cols), rm.zero.both = !is.null(coord.cols),
rm.equal = !is.null(coord.cols),   rm.imposs = !is.null(coord.cols),
rm.imprec.any = !is.null(coord.cols), rm.imprec.both = !is.null(coord.cols),
imprec.digits = 0, rm.uncert = !is.null(uncert.col), uncert.limit = 50000,
uncert.na.pass = TRUE, rm.abs = !is.null(abs.col), year.min = NULL,
year.na.pass = TRUE, plot = TRUE, extend = 0.1, ...)
```

Arguments

`data`	an object inheriting class 'data.frame' with the spatial coordinates to be cleaned, or a 'SpatVector' of points.
`coord.cols`	character or integer vector of length 2, with either the names or the positions of the columns that contain the spatial coordinates in 'data' - in this order, LONGitude and LATitude, or x and y. Can be NULL if 'data' is a 'SpatVector', in which case the coordinates will be extracted with terra::crds().
`uncert.col`	character or integer vector of length 1, with either the name or the position of the column that reports spatial uncertainty in 'data' (e.g., in GBIF this column is usually named "coordinateUncertaintyInMeters", the default).
`abs.col`	character or integer vector of length 1, with either the name or the position of the column that specifies whether the species is present or absent (e.g., in GBIF this column is named "occurrenceStatus", the default). If povided, rows where the value in this column is not NA and is either (case-insensitive) "absent" or "ausente" will be removed.

`year.col` character or integer vector of length 1, with either the name or the position of the column that specifies the year in which the observation was made (e.g., in GBIF this column is named "year", the default).

`rm.dup` logical, whether to remove rows with exactly the same location, i.e. the same pair of longitude-latitude coordinates. The default is TRUE if 'coord.cols' is not NULL, and FALSE otherwise.

`rm.missing.any` logical, whether to remove rows where at least one of the coordinates is NA. The default is TRUE if 'coord.cols' is not NULL, and FALSE otherwise.

`rm.missing.both`
logical, whether to remove rows where both coordinates are NA. The default is TRUE if 'coord.cols' is not NULL and FALSE otherwise, but it is not used (as it is redundant) if rm.missing.any=TRUE.

`rm.zero.any` logical, whether to remove rows where at least one of the coordinates exactly equals zero (which is likely an error). The default is TRUE if 'coord.cols' is not NULL, and FALSE otherwise.

`rm.zero.both` logical, whether to remove rows where both coordinates equal zero (which is likely an error). The default is TRUE if 'coord.cols' is not NULL and FALSE otherwise, but it is not used (as it is redundant) if rm.zero.any=TRUE.

`rm.equal` logical, whether to remove rows where latitude exactly equals longitude (which is likely an error). The default is TRUE if 'coord.cols' is not NULL, and FALSE otherwise.

`rm.imposs` logical, whether to remove rows with impossible coordinates outside planet Earth, i.e. with absolute value >180 for longitude or >90 for latitude. The default is TRUE if 'coord.cols' is not NULL, and FALSE otherwise. Note that this is only valid for unprojected angular coordinates in geographic degrees.

`rm.imprec.any` logical, whether to remove rows where at least one of the coordinates is imprecise, i.e. has no more decimal places than 'imprec.digits'. The default is TRUE if 'coord.cols' is not NULL and FALSE otherwise, but note this is normally only relevant for unprojected geographical coordinates in degrees; if your coordinates are in meters, they are usually precise enough without decimal places, so you should probably set this argument and the next to FALSE.

`rm.imprec.both` logical, whether to remove rows where both coordinates are imprecise, i.e. have no more decimal places than 'imprec.digits'. The default is TRUE if 'coord.cols' is not NULL and FALSE otherwise, but it is not used (as it is redundant) if rm.imprec.any=TRUE. See 'rm.imprec.any' above for important details.

`imprec.digits` integer, maximum number of digits to consider that a coordinate is imprecise. Used only if 'rm.imprec.any' or 'rm.imprec.both' is TRUE. The default is 0, for eliminating coordinates with no more than zero decimal places.

`rm.uncert` logical, whether to remove rows where the value in 'uncert.col' is higher than 'uncert.limit'. The default is TRUE if 'uncert.col' is not NULL, and FALSE otherwise.

`uncert.limit` lnumeric, threshold value for 'uncert.col'. If rm.uncert=TRUE and 'uncert.col' is provided, rows with values above this will be excluded. The default is 50,000, i.e. 50 km if the values in 'uncert.col' are in meters.

`uncert.na.pass`	logical, whether rows with NA in 'uncert.col' should be kept as having no uncertainty. The default is TRUE.
`rm.abs`	logical, whether to remove rows where the value in 'abs.col' is (case-insensitive) 'absent'. The default is TRUE if 'abs.col' is not NULL, and FALSE otherwise.
`year.min`	positive integer specifying the minimum (earliest) value admitted for the year column. The default is NULL (no limit).
`year.na.pass`	logical, whether rows with NA in 'year.col' should be kept as if fulfilling the year.min criterion. The default is TRUE.
`plot`	logical value specifying whether to plot the result. The default is TRUE.
`extend`	(if plot=TRUE) numeric value specifying the proportion of the input coordinates range by which to increase the extent of the output plot. The default is 0.1, i.e. 10%.
`...`	(if plot=TRUE) some additional arguments that can be passed to `plot`, e.g "main" (plot title).

Details

This function applies some basic cleaning procedures for species occurrence data, removing some of the most common errors in biodiversity databases. It is inspired by a few functions (namely 'coord_incomplete', 'coord_imprecise', 'coord_impossible', 'coord_unlikely' and 'coord_uncertain') that were present in the 'scrubr' package by Scott Chamberlain, which was archived (https://github.com/ropensci-archive/scrubr). It implements some additional cleaning procedures, such as removal of records of absence and records older than a given year. It also maps the result.

Value

This function returns a data frame of the input 'data' (or a 'SpatVector' if this matches the input) after excluding the rows that met the specified removal criteria. The row names match the original ones in 'data', at least if 'data' is of class 'data.frame'. Messages are displayed in the console saying how many rows passed each removal filter. If plot=TRUE (the default), a plot is also displayed with the selected points (blue dots) overlaid on the input points (red "x").

Author(s)

A. Marcia Barbosa

See Also

`gridRecords`

Examples

```
## Not run:
  # you can run these examples if you have the 'geodata' package installed

  # download some species occurrences from GBIF:
  occ <- geodata::sp_occurrence(genus = "Orycteropus", species = "afer",
  fixnames = FALSE)
```

```
# clean occurrences:
names(occ)
occ_clean <- cleanCoords(occ,
                  coord.cols = c("decimalLongitude", "decimalLatitude"),
                  abs.col = "occurrenceStatus",
                  uncert.col = "coordinateUncertaintyInMeters",
                  uncert.limit = 10000,  # 10 km tolerance
                  year.col = "year", year.min = 1950)

## End(Not run)
```

`corSelect` *Select among correlated variables based on a given criterion*

Description

This function computes pairwise correlations among the variables in a dataset and, among each pair of variables correlated above a given threshold (or, optionally, below a given significance value), it excludes the variable with either the highest variance inflation factor (VIF), or the weakest, least significant or least informative bivariate (individual) relationship with the response variable, according to a specified criterion.

Usage

```
corSelect(data, sp.cols = NULL, var.cols, coeff = TRUE,
cor.thresh = ifelse(isTRUE(coeff), 0.8, 0.05),
select = ifelse(is.null(sp.cols), "VIF", "cor"), test = "Chisq",
family = "auto", use = "pairwise.complete.obs", method = "pearson",
simplif = FALSE, verbosity = 1)
```

Arguments

`data`	a data frame containing the (optionally response and) predictor variables.
`sp.cols`	optional, name or index number of the column of 'data' that contains the response (e.g. species) variable. Currently, only one 'sp.cols' can be used at a time, so an error message is returned if length(sp.cols) > 1. If left NULL, 'select' will be "VIF" by default.
`var.cols`	names or index numbers of the columns of 'data' that contain the predictor variables.
`coeff`	logical value (default TRUE) indicating whether two variables should be considered highly correlated based on the magnitude of their coefficient of correlation. If set to FALSE, this classification will be based instead on the p-value of the correlation, but mind that (with sufficient sample size) correlations can be statistically significant even if weak.

`cor.thresh` if coeff=TRUE (the default): threshold value of absolute correlation coefficient (default 0.8) above which two predictor variables are considered highly correlated. If coeff=FALSE: threshold value of p-value (default 0.05) below which two predictor variables are considered highly (or significantly) correlated.

`select` character value indicating the criterion for excluding a variable within each highly correlated pair. Can be "VIF" (the default if 'sp.cols' is NULL), "cor" (the default if 'sp.cols' is specified), "p.value"," "AIC" or "BIC" (see Details).

`test` argument to pass to the `FDR` function (which, in turn, passes it to `anova`) if test="p.value". The default is "Chisq" for back-compatibility.

`family` If 'select' is "p.value", "AIC" or "BIC", the error distribution and (optionally) the link function to use for assessing significant / informative variables (see `glm` or `family` for details). The default "auto" automatically uses "binomial" family for response variables containing only values of 0 and 1; "poisson" for positive integer responses (i.e. count data); "Gamma" for positive non-integer; and "gaussian" (i.e., linear models) otherwise.

`use` argument to pass to `cor` indicating which rows to use when there are missing values. Can be "pairwise.complete.obs" (the default here), "everything", "all.obs", "complete.obs", "na.or.complete".

`method` argument to pass to `cor` specifying the correlation coefficient to use. Can be "pearson" (the default, with a recommended minimum of 30 rows of data), "kendall", or "spearman" (with a recommended minimum of 10 rows of data).

`simplif` (if 'select' is not NULL) logical (default FALSE) indicating whether to produce a simplified output, consisting only of a character vector of the names of the selected variables.

`verbosity` integer value indicating the amount of messages to display. The default is 1, for a medium amount of messages. Use 2 for more messages.

Details

Correlations among variables are often considered problematic in multivariate models, as they inflate the variance of coefficients and thus may bias the interpretation of the effects of those variables on the response (Legendre & Legendre 2012). Note, however, that the perceived problem often stems from misconceptions about the interpretation of multiple regression models, and that removing (albeit correlated) variables usually reduces predictive power (Morrissey & Ruxton 2018, Gregorich et al. 2021, Vanhove 2021). Removing high correlations is, however, a way of reducing the number of variables to include in a model, when the potentially meaningful variables are still numerous and no better a priori selection criterion is available.

One of the strategies to reduce correlations within a dataset consists of excluding one from each pair of highly correlated variables. However, it is not always straightforward (or ecological knowledge is not alway sufficient) to choose which variable to exclude. This function selects among pairwise-correlated variables based either on their variance inflation factor (VIF: Marquardt 1970; Mansfield & Helms 1982) within the variables dataset (obtained with the `multicol` function and recalculated iteratively after each variable exclusion); or on their relationship with the response, either by simply computing the `cor`relation between each variable and the response and excluding the variable with the smallest absolute coefficient; or by building a bivariate generalized linear model (`glm`) of each variable against the response and excluding, among each of two correlated variables, the one with the largest (worst) p-value, AIC (Akaike's Information Criterion: Akaike, 1973) or BIC (Bayesian Information Criterion, also known as Schwarz criterion, SBC or SBIC: Schwarz, 1978), which is calculated with the FDR function.

If 'select' is NULL, or if 'select' is other than "VIF" but 'sp.cols' is NULL, the function returns only a table showing the pairs of variables that are correlated beyond the given threshold, without selection or exclusion. If the 'select' criterion requires assessing bivariate relationships and 'sp.cols' is provided, the function uses only the rows of the dataset where 'sp.cols' (used as the response variable) contains finite values against which the predictor variables can be modelled; rows with NA or NaN in 'sp.cols' are thus excluded from the calculation of correlations among predictor variables.

Value

If simplif=TRUE (and select is not NULL), the function returns a character vector of the names of the selected variables. Otherwise (the default), it returns a list of 7 elements (unless select=NULL, in which case it returns only the first of these elements):

`high.correlations`
: data frame showing the pairs of input variables that are correlated beyond the given threshold, their correlation coefficient and its associated p-value.

`bivariate.significance`
: data frame with the individual p-value, AIC, BIC and correlation coefficient (if one of these was the 'select' criterion and if 'sp.cols' was provided) of each of the highly correlated variables against the response variable.

`excluded.vars`
: character vector containing the names of the variables to exclude (i.e., from each highly correlated pair, the variable with the worse 'select' score.

`selected.vars`
: character vector containing the names of the variables to select (i.e., the non-correlated variables and, from each correlated pair, the variable with the better 'select' score).

`selected.var.cols`
: integer vector containing the column indices of the selected variables in 'data'.

`strongest.remaining.corr`
: numerical value indicating the strongest correlation coefficient among the selected variables.

`remaining.multicollinearity`
: data frame showing the multicollinearity among the selected variables.

Author(s)

A. Marcia Barbosa

References

Akaike H. (1973) Information theory and an extension of the maximum likelihood principle. In: Petrov B.N. & Csaki F., 2nd International Symposium on Information Theory, Tsahkadsor, Armenia, USSR, September 2-8, 1971, Budapest: Akademiai Kiado, p. 267-281.

Gregorich M., Strohmaier S., Dunkler D. & Heinze G. (2021) Regression with Highly Correlated Predictors: Variable Omission Is Not the Solution. Int. J. Environ. Res. Public Health, 18: 4259.

Legendre P. & Legendre L. (2012) Numerical ecology (3rd edition). Elsevier, Amsterdam: 990 pp.

Marquardt D.W. (1970) Generalized inverses, ridge regression, biased linear estimation, and non-linear estimation. Technometrics 12: 591-612.

Mansfield E.R. & Helms B.P. (1982) Detecting multicollinearity. The American Statistician 36: 158-160.

Morrissey M.B. & Ruxton G.D. (2018) Multiple Regression Is Not Multiple Regressions: The Meaning of Multiple Regression and the Non-Problem of Collinearity. Philosophy, Theory, and Practice in Biology, 10: 003. DOI: 10.3998/ptpbio.16039257.0010.003

Schwarz, G.E. (1978) Estimating the dimension of a model. Annals of Statistics, 6 (2): 461-464.

Vanhove J. (2021) Collinearity isn't a disease that needs curing. Meta-Phsychology 5, MP.2020.2548. DOI: 10.15626/MP.2021.2548

See Also

`multicol`, `FDR`, `cor`; and `collinear` in package **collinear**, which handles also categorical variables and allows specifying a preference order for the variables to keep.

Examples

```
data(rotif.env)

corSelect(rotif.env, var.cols = 5:17, select = NULL)

corSelect(rotif.env, var.cols = 5:17)

corSelect(rotif.env, sp.cols = 46, var.cols = 5:17)

corSelect(rotif.env, sp.cols = 46, var.cols = 5:17, simplif = TRUE)

corSelect(rotif.env, sp.cols = 46, var.cols = 5:17, cor.thresh = 0.7)

corSelect(rotif.env, sp.cols = 46, var.cols = 5:17, select = "BIC", method = "spearman")
```

---

distMat *Distance matrix for spatial coordinates*

Description

Computes a distance matrix for a given set of spatial coordinates using a specified distance measure.

Usage

```
distMat(coords, CRS = NULL, method = "auto", verbosity = 2)
```

Arguments

- `coords` — data frame (or an object that can be coerced to such) with only two columns containing the spatial coordinates (x and y, or longitude and latitude, or easting and northing, in this order!)
- `CRS` — Coordinate Reference System for 'coords' (if it is not a SpatVector with its CRS defined already), in one of the following formats: WKT/WKT2, <authority>:<code>, or PROJ-string notation (see `terra::crs()`). If provided and either the 'geodist' or the 'terra' package is installed, distances are computed with `geodist::geodist()` or with `terra::distance()`, thus accounting for the curvature of the Earth. If not provided and coordinates fall within valid latitude-longitude bounds, EPSG:4326 is assumed.
- `method` — The method to use for distance calculation. Partial and case-insensitive argument matching is used. Options are:
  - "auto" (the default): the recommended method under the circumstances (see Details).
  - "euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski": older methods, implemented in `stats::dist()`, which do not account for the curvature of the Earth
  - "geodesic", "haversine" "vincenty", "cheap": methods implemented in `geodist::geodist()` (note this requires that 'coords' are in geographic degrees)
  - "geo", "haversine", "cosine": methods implemented in `terra::distance()` (also requiring 'coords' in geographic degrees)

  If a metric has the same name in 'terra' and 'geodist' and both packages are available, the latter is used, as it is slightly faster.
- `verbosity` — integer specifying the amount of messages to display along the process. The default is 2, for the maximum amount of messages available.

Details

This function computes a matrix of pairwise distances for the input set of coordinates. It supports various distance measures by calling `stats::dist()`, `terra::distance()` or `geodist::geodist()`, depending on the specified method and on the installed R packages.

With the default "auto" option, if neither `terra` nor `geodist` are installed or if the input 'coords' are projected rather than in geographic degrees, the method is "euclidean" from `stats::dist()`. Otherwise, the method is "haversine" (relatively fast, but may be inaccurate at long distances, especially near the poles) if there are distances < 1 m or 0.00001 degrees; or "cosine" (faster but inaccurate for small distances) otherwise (unless 'terra' version is older than 1.8.7, before "cosine" was implemented). You can instead choose a specific option, e.g. "geo" or "geodesic" for the most accurate yet slowest distance method (see ?terra::distance; may crash if many points); "vincenty" (more accurate than "haversine" as it considers the Earth's ellipticity, but slower and may fail for nearly antipodal points); or "cheap" (the fastest, but inaccurate for maximum distances >100 km).

Value

A matrix of distances between the coordinates.

Author(s)

A. Marcia Barbosa

Examples

```
# generate some random geographical coordinates:

n_points <- 10

set.seed(123)
coords <- matrix(c(runif(n_points, min = -180, max = 180),
                   runif(n_points, min = -90, max = 90)),
                   ncol = 2)


# compute distance matrix (may fail with outdated PROJ version):

## Not run:
  distMat(coords, method = "euclidean")

  distMat(coords, method = "auto")

  distMat(coords, method = "auto", CRS = "EPSG:4326")

## End(Not run)
```

`distPres` *(Inverse) distance to the nearest presence*

Description

This function takes a matrix or data frame containing species presence (1) and absence (0) records and their spatial coordinates (optionally also a pre-computed pairwise distance matrix for all localities), and computes the (inverse) distance from each locality to the nearest presence locality for each species. It can be used to compute a simple spatial interpolation model of a species' distribution (e.g. Barbosa 2015, Areias-Guerreiro et al. 2016).

Usage

```
distPres(data, sp.cols, coord.cols = NULL, id.col = NULL,
dist.mat = NULL, CRS = NULL, method = "auto", suffix = "_D",
p = 1, inv = TRUE, verbosity = 2)
```

Arguments

`data`	a matrix or data frame containing, at least, two columns with spatial coordinates, and one column per species containing their presence (1) and absence (0) data, with localities in rows.

sp.cols	names or index numbers of the columns containing the species presences and absences in 'data'. It must contain only zeros (0) for absences and ones (1) for presences.
coord.cols	names or index numbers of the columns containing the spatial coordinates in 'data' (x and y, or longitude and latitude, in this order!).
id.col	optionally, the name or index number of a column (to be included in the output) containing locality identifiers in 'data'.
dist.mat	optional pre-computed pairwise distance matrix for the localities in 'data'. If provided, arguments 'CRS' and 'method' are not used.
CRS	(if 'dist.mat' is NULL) argument to pass to `distMat` specifying coordinate reference system of the 'coord.cols' in 'data'.
method	(if 'dist.mat' is NULL) argument to pass to `distMat`. The default is "auto".
suffix	character indicating the suffix to add to the distance columns in the resulting data frame. The default is `"_D"`.
p	the power to which distance should be raised. The default is 1; use 2 or higher if you want more conservative distances.
inv	logical value indicating whether distance should be inverted (i.e., converted to proximity), in which case they will be standardized to vary between 0 and 1, and then subtracted from 1. The output will also vary between 0 and 1, with higher values meaning closer to a presence, and 1 meaning presence. The default is `TRUE`, which can be used as a fuzzy version of presence-absence (for using e.g. with `fuzSim` and `simMat`).
verbosity	integer specifying the amount of messages to display along the process. The default is 2, for the maximum amount of messages available.

Value

This function returns a matrix or data frame containing the identifier column (if provided in 'id.col') and one column per species containing the distance (inverse by default, i.e. proximity) from each locality to the nearest presence of that species.

Author(s)

A. Marcia Barbosa

References

Areias-Guerreiro J., Mira A. & Barbosa A.M. (2016) How well can models predict changes in species distributions? A 13-year-old otter model revisited. Hystrix - Italian Journal of Mammalogy, in press. DOI: http://dx.doi.org/10.4404/hystrix-27.1-11867

Barbosa A.M. (2015) fuzzySim: applying fuzzy logic to binary similarity indices in ecology. Methods in Ecology and Evolution, 6: 853-858

See Also

`dist`

Examples

```
data(rotif.env)

head(rotif.env)

names(rotif.env)


## Not run:
# distance to presence (in meters):

rotifers.dist <- distPres(rotif.env, sp.cols = 18:47,
coord.cols = c("Longitude", "Latitude"), id.col = 1, p = 1,
inv = FALSE, suffix = "_D", CRS = "EPSG:4326")

head(rotifers.dist)


# inverse squared distance to presence (proximity):

rotifers.invd2 <- distPres(rotif.env, sp.cols = 18:47,
coord.cols = c("Longitude", "Latitude"), id.col = 1, p = 2,
inv = TRUE, suffix = "_iDsq", CRS = "EPSG:4326")

head(rotifers.invd2)

## End(Not run)
```

`dms2dec` *Degree-minute-second to decimal degree coordinates*

Description

This function converts degree-minute-second geographic coordinates to decimal degree (numeric) coordinates appropriate for mapping and analysis.

Usage

```
dms2dec(dms,
seps = c("\\u00ba", "\\u00b0", "\\'", "\\'", "\\\"", "\\\\?"))
```

Arguments

`dms`	character vector of geographic coordinates (latitude or longitude) in degree-minute-second-hemisphere format, e.g. 41° 34' 10.956" N (with or without spaces); or in degree-decimal minute format, e.g. 41° 34.1826' N (with or without spaces)
`seps`	character vector of possible separators in 'dms'. The default includes commonly used symbols for degrees, minutes and seconds, converted with stringi::stri_escape_unicode() for portability

Value

This function returns a numeric vector of the input coordinates after conversion to decimal degree format.

Author(s)

A. Marcia Barbosa (https://github.com/AMBarbosa) with contributions by Paul Melloy (https://github.com/PaulMelloy)

Examples

```
coords_dms <- structure(list(Longitude = c("31º40'44.12''E", "31º41'23.35''E",
"31º37'01.94''E", "30º53'07.75''E"), Latitude = c("24º54'36.44''S",
"24º05'02.09''S", "25º09'46.72''S", "24º12'09.02''S")), row.names = c(NA, 4L),
class = "data.frame")
coords_dms

lon_dec <- dms2dec(coords_dms$Longitude)
lat_dec <- dms2dec(coords_dms$Latitude)

coords_dec <- sapply(coords_dms, dms2dec)
coords_dec
```

---

entropy *(Fuzzy) entropy*

Description

This function computes fuzzy entropy (Kosko 1986, Estrada & Real 2021), or optionally Shannon's (1948) entropy.

Usage

```
entropy(data, sp.cols = 1:ncol(data), method = "fuzzy", base = exp(1),
plot = TRUE, plot.type = "lollipop", na.rm = TRUE, ...)
```

Arguments

data	a vector, matrix or data frame containing the data to analyse.
sp.cols	names or index numbers of the columns of 'data' that contain the values for which to compute entropy (if 'data' is not a vector). The default is to use all columns.
method	character value indicating the method to use. Can be "fuzzy" (the default) or "Shannon". The former requires the input to be a fuzzy system (e.g. Favourability values), while the latter requires probabilities. If method="Shannon" and the values for a vector or column do not sum up to 1, they are divided by their sum so that this additional requirement is met (Estrada & Real 2021).

`base`	base for computing the logarithm if method="Shannon". The default is the natural logarithm.
`plot`	logical value indicating whether to plot the results (if 'data' has more than one column). The default is TRUE.
`plot.type`	character value indicating the type of plot to produce (if plot=TRUE). Can be "lollipop" (the default) or "barplot".
`na.rm`	logical value indicating whether NA values should be removed before computations. The default is TRUE.
`...`	additional arguments to be passed to `barplot` or to `modEvA::lollipop`.

Details

Fuzzy entropy (Kosko 1986) applies to fuzzy systems (such as `Fav`ourability) and it can take values between zero and one. Fuzzy entropy equals one when the distribution of the values is uniform, i.e. 0.5 in all localities. The smaller the entropy, the more orderly the distribution of the values, i.e. the closer they are to 0 or 1, distinguishing (potential) presences and absences more clearly. Fuzzy entropy can reflect the overall degree of uncertainty in a species' distribution model predictions, and it is directly comparable across species and study areas (Estrada & Real 2021).

Shannon's entropy requires that the input values are probabilities and sum up to 1 (Shannon 1948). This makes sense when analysing the probability that a unique event occurs in a finite universe. However, if a species has more than one presence, the sum of probabilities in all localities equals the number of presences. To satisfy the condition that the inputs sum up to 1, this function divides each value by the sum of values when this is not the case (if method="Shannon"). Notice that this has a mathematical justification but not a biogeographical sense, and (unlike fuzzy entropy) the results are comparable only between models based on the same number of presences + absences, e.g. in a context of selection of variables for a model (Estrada & Real 2021).

Value

This function returns a numeric value of entropy for 'data' (if it is a numeric vector) or for each of 'sp.cols' in 'data' (if it is a matrix or data frame). Optionally (and by default), a plot is also produced with these values (if there is more than one column) for visual comparison.

Author(s)

A. Marcia Barbosa

References

Estrada A. & Real R. (2021) A stepwise assessment of parsimony and fuzzy entropy in species distribution modelling. Entropy, 23: 1014

Kosko B. (1986) Fuzzy entropy and conditioning. Information Sciences, 40: 165-174

Shannon C.E. (1948) A mathematical theory of communication. Bell System Technical Journal, 27: 379-423

Examples

```
data(rotif.env)

pred <- multGLM(rotif.env, sp.cols = 18:20, var.cols = 5:17)$predictions

head(pred)

entropy(pred, sp.cols = c("Abrigh_F", "Afissa_F", "Apriod_F"))

entropy(pred, sp.cols = c("Abrigh_P", "Afissa_P", "Apriod_P"), method = "Shannon")
```

`Fav` *Favourability (probability without the effect of sample prevalence)*

Description

Computes prevalence-independent favourability for a species' presence, based on a presence/(pseudo)absence model object, or on a vector of predicted probability values plus either the modelled binary response variable, or the total numbers of modelled ones and zeros, or the prevalence (proportion of ones) in the modelled binary response (i.e., in the model training data). It can also do the inverse operation, i.e. compute presence probability from a vector of favourability values and the modelled prevalence.

Usage

```
Fav(model = NULL, obs = NULL, pred = NULL, n1n0 = NULL, sample.preval = NULL,
method = "RBV", true.preval = NULL, inv = FALSE, verbosity = 2)
```

Arguments

`model`	optional binary-response presence/(pseudo)absence model object of class "glm", "gam", "gbm", "randomForest" or "bart" (the latter computed with keeptrees=TRUE), computed with weights=NULL (i.e., without weighing presences and absences differently).
`obs`	alternatively to 'model', a vector of the 1 and 0 values of the binary response variable (e.g. presence-absence of a species) in the model training data. This argument is ignored if 'model' is provided.
`pred`	alternatively to 'model', a numeric vector, RasterLayer or SpatRaster of predicted presence probability (not suitability) values, produced by a presence/(pseudo)absence modelling method (computed with weights=NULL). This argument is ignored if 'model' is provided.
`n1n0`	alternatively to 'obs' or 'sample.preval', an integer vector of length 2 providing the total numbers of modelled ones and zeros, respectively, of the binary response variable in the model training data (not necessarily the same as in the raw input data, which may e.g. have NAs or duplicates!). Ignored if 'obs' or 'model' is provided.

`sample.preval`	alternatively to 'obs' or 'n1n0', the prevalence (proportion of ones) of the binary response variable in the model training data (not necessarily the same as in the raw input data, which may e.g. have NAs or duplicates!). Ignored if 'model' is provided.
`method`	either "RBV" for the original Real, Barbosa & Vargas (2006) procedure, or "AT" if you want to try out the (misled) modification proposed by Albert & Thuiller (2008) - but see Details!
`true.preval`	the true prevalence (as opposed to sample prevalence), necessary if you want to try the "AT" method (but see Details!)
`inv`	logical value (default FALSE) indicating whether to do the inverse operation instead, i.e. compute presence probability from favourability. If TRUE, the user must supply not a 'model' object, but rather a 'pred' vector of favourability values, plus either 'obs' or 'n1n0' or 'sample.preval'.
`verbosity`	numeric value indicating the amount of messages to display; currently meaningful values are 0, 1, and 2 (the default).

Details

Presence-(pseudo)abence modelling methods, such as GLM, GAM, Random Forests, BRT/GBM or BART, are widely used for modelling species' potential distributions. These models predict presence probability, which incorporates the prevalence (proportion of presences) of the species in the modelled sample. It thus reflects the natural fact that widespread species are more likely to occur than rare species, independently of (or under equally good) environmental conditions.

The favourability function (Barbosa, 2006; Real, Barbosa & Vargas, 2006) removes this effect of prevalence from presence probability predictions. Unlike the raw probability, favourability is standardized: a value of 0.5 always means that the local environmental conditions neither increase nor decrease the species' probability of occurrence beyond its general prevalence. It is thus a consistent "neutral" baseline that can be compared and combined across species, regions, spatial resolutions, and time periods.

Using simulated data, Albert & Thuiller (2008) proposed a modification to the favourability function which requires knowing the true prevalence of the species (not just the prevalence in the modelled sample), though this is rarely possible in real-world modelling. Besides, this suggestion was based on the misunderstanding that the favourability function was a way to obtain the probability of occurrence when prevalence differs from 50%, which is incorrect (see Acevedo & Real 2012).

To get environmental favourability, you can use the 'Fav' function with model predictions of presence probability, together with the prevalence or the presence/absence ratio in the actually modelled sample (which is not necessarily the same as in the raw presence/(pseudo)absence data! e.g. if duplicates are automatically removed, or some presences fall outside the utilized pixels, or pseudoabsences outnumber the available pixels). Input data for 'Fav' are either a model object of an implemented class, or the vector of modelled presences-absences (1-0) of your species and the corresponding presence probability values, obtained e.g. with predict(mymodel, mydata, type = "response"). Alternatively to the presences-absences, you can provide either the sample prevalence or the numbers of presences and absences in the dataset that was modelled to generate the presence probabilities. In case you want to use the "AT" method (but see Acevedo & Real 2012!), you also need to provide the true (besides the sample) prevalence of your species.

Value

If 'model' is provided or if 'pred' is a numeric vector, the function returns a numeric vector of the favourability values. If 'model' is not provided (which would override other arguments) and 'pred' is a RasterLayer or a SpatRaster, the function returns an object of the same class, containing the favourability values.

Note

This function is applicable only to presence probability values obtained without weighting presences and absences differently (i.e. with weights=NULL), thus reflecting the sample prevalence, which is generally the default in presence/absence modelling functions (like glm). Note, however, that some modelling packages may use different defaults when calling these functions, e.g. biomod2::BIOMOD_Modeling() with automatically generated pseudo-absences.

Author(s)

A. Marcia Barbosa

References

Acevedo P. & Real R. (2012) Favourability: concept, distinctive characteristics and potential usefulness. Naturwissenschaften 99: 515-522

Albert C.H. & Thuiller W. (2008) Favourability functions versus probability of presence: advantages and misuses. Ecography 31: 417-422.

Barbosa A.M.E. (2006) Modelacion de relaciones biogeograficas entre predadores, presas y parasitos: implicaciones para la conservacion de mamiferos en la Peninsula Iberica. PhD Thesis, University of Malaga (Spain).

Real R., Barbosa A.M. & Vargas J.M. (2006) Obtaining environmental favourability functions from logistic regression. Environmental and Ecological Statistics 13: 237-245.

See Also

multGLM

Examples

```
# obtain a probability model and its predictions:

data(rotif.env)

names(rotif.env)

mod <- with(rotif.env, glm(Abrigh ~ Area + Altitude +
AltitudeRange + HabitatDiversity + HumanPopulation,
family = binomial))

prob <- predict(mod, data = rotif.env, type = "response")
```

```
# obtain predicted favourability in different ways:

Fav(model = mod)

Fav(obs = rotif.env$Abrigh, pred = prob)

Fav(pred = mod$fitted.values, sample.preval = prevalence(model = mod))
```

---

favClass *Classify favourability into 3 categories (low, intermediate, high)*

Description

This function takes a vector of Favourability values and reclassifies them into 3 increasing categories: low, intermediate or high. By default, the breaks between these classes are 0.2 and 0.8 (see Details), although these can be changed by the user.

Usage

```
favClass(fav, breaks = c(0.2, 0.8), character = FALSE)
```

Arguments

`fav`	a numeric vector of favourability values (obtained, e.g., with functions `Fav` or `multGLM`).
`breaks`	a numeric vector of length 2 containing the two values which will divide `fav` into the 3 classes. Defaults to c(0.2, 0.8) following the literature (see Details).
`character`	logical value (default FALSE) indicating whether the result should be returned in character rather numeric form.

Details

Some applications of species distribution models imply setting a threshold to separate areas with high and low probability or favourability for occurrence (see, e.g., `bioThreat`). However, it makes little sense to establish as markedly different areas with, for example, 0.49 and 0.51 favourability values (Hosmer & Lemeshow, 1989). It may thus be wiser to open a gap between values considered as clearly favourable and clearly unfavourable. When this option is taken in the literature, commonly used breaks are 0.8 as a threshold to classify highly favourable values, as the odds are more than 4:1 favourable to the species; 0.2 as a threshold below which to consider highly unfavourable values, as odds are less than 1:4; and classifying the remaining values as intermediate favourability (e.g., Munoz & Real 2006, Olivero et al. 2016).

Value

This function returns either an integer or a character vector (depending on the 'character' argument, which is FALSE by default), of the same length as `fav`, reclassifying it into 3 categories: 1 ('low'), 2 ('intermediate'), or 3 ('high').

Author(s)

A. Marcia Barbosa

References

Hosmer D.W. Jr & Lemeshow S. (1989) Applied logistic regression. John Wiley & Sons, New York

Munoz A.R. & Real R. (2006) Assessing the potential range expansion of the exotic monk parakeet in Spain. Diversity and Distributions, 12: 656-665

Olivero J., Fa J.E., Real R., Farfan M.A., Marquez A.L., Vargas J.M., Gonzalez J.P., Cunningham A.A. & Nasi R. (2017) Mammalian biogeography and the Ebola virus in Africa. Mammal Review, 47: 24-37

See Also

Fav, multGLM

Examples

```
data(rotif.env)
mods <- multGLM(rotif.env, sp.cols = 20, var.cols = 5:17)
fav <- mods$predictions[ , 2]
data.frame(fav = fav, favcl_num = favClass(fav),
favcl_chr = favClass(fav, character = TRUE))
```

FDR — *False Discovery Rate*

Description

Calculate the false discovery rate (type I error) under repeated testing and determine which variables to select and to exclude from multivariate analysis.

Usage

```
FDR(data = NULL, sp.cols = NULL, var.cols = NULL, pvalues = NULL,
test = "Chisq", model.type = NULL, family = "auto", correction = "fdr",
q = 0.05, verbose = NULL, verbosity = 1, simplif = FALSE)
```

Arguments

`data`	a data frame containing the response and predictor variables (one in each column).
`sp.cols`	name or index number of the column containing the response variable (currently implemented for only one response variable at a time).
`var.cols`	names or index numbers of the columns containing the predictor variables.

`pvalues`	optionally, instead of 'data', 'sp.cols' and 'var.cols', a data frame with the names of the predictor variables in the first column andtheir bivariate p-values (obtained elsewhere) in the second column. Example: pvalues <- data.frame(var = letters[1:5], pval = c(0.02, 0.004, 0.07, 0.03, 0.05)).
`test`	(if 'pvalues' not provided) argument to pass to `anova` to obtain the p-value for each variable. Should be one of "Chisq" (currently the default, for back-compatibility), "Rao", "LRT" or "F" (the latter is not appropriate for models of family "binomial").
`model.type`	this argument (previously a character value, either "LM" or "GLM") is now deprecated and ignored with a warning if provided. This information is now included in argument 'family' – e.g., if you want linear models (LM), you can set 'family = "gaussian"'.
`family`	The error distribution and (optionally) the link function to use (see `glm` or `family` for details). The default "auto" automatically uses "binomial" family for response variables containing only values of 0 and 1; "poisson" for positive integer responses (i.e. count data); "Gamma" for positive non-integer; and "gaussian" (i.e., linear models) otherwise.
`correction`	the correction procedure to apply to the p-values; see `p.adjust.methods` for available options and `p.adjust` for more information. The default is "fdr".
`q`	the threshold value of FDR-corrected significance above which to reject variables. Defaults to 0.05.
`verbose`	deprecated argument, replaced by 'verbosity' (below).
`verbosity`	integer value indicating the amount of messages to display. The default is 1, for a medium amount of messages. Use 2 for more messages.
`simplif`	logical value indicating if simplified results should be provided (see Value).

Details

It is common in ecology to search for statistical relationships between species' occurrence and a set of predictor variables. However, when a large number of variables is analysed (compared to the number of observations), false findings may arise due to repeated testing. Garcia (2003) recommended controlling the false discovery rate (FDR; Benjamini & Hochberg 1995) in ecological studies. The `p.adjust` R function performs this and other corrections to the significance (p) values of variables under repeated testing. The 'FDR' function performs repeated regressions (either linear or logistic) or uses already-obtained p values for a set of variables; calculates the FDR with 'p.adjust'; and shows which variables should be retained for or excluded from further multivariate analysis according to their corrected p values (see, for example, Barbosa, Real & Vargas 2009).

The FDR function uses the Benjamini & Hochberg ("BH", alias "fdr") correction by default, but check the `p.adjust` documentation for other available methods, namely "BY", which allows for non-independent data. Input data may be the response variable (for example, the presence-absence or abundance of a species) and the predictors (a table with one predictor variable in each column, with the same number of rows and in the same order as the response). Alternatively, you may already have performed the univariate regressions and have a set of variables and corresponding p values which you want to correct with FDR; in this case, get a table with your variables' names in the first column and their p values in the second column, and supply it as the 'pvalues' argument (no need to provide response or predictors in this case).

Value

If simplif = TRUE, this function returns a data frame with the variables' names as row names and 4 columns containing, respectively, their individual (bivariate) coefficients against the response, their individual AIC (Akaike's Information Criterion; Akaike, 1973), BIC (Bayesian Information Criterion, also known as Schwarz criterion, SBC, SBIC; Schwarz, 1978), p-value and adjusted p-value according to the applied 'correction'. If simplif = FALSE (the default), the result is a list of two such data frames:

`exclude` with the variables to exclude.

`select` with the variables to select (under the given 'q' value).

Author(s)

A. Marcia Barbosa

References

Akaike, H. (1973) Information theory and an extension of the maximum likelihood principle. In: Petrov B.N. & Csaki F., 2nd International Symposium on Information Theory, Tsahkadsor, Armenia, USSR, September 2-8, 1971, Budapest: Akademiai Kiado, p. 267-281.

Barbosa A.M., Real R. & Vargas J.M (2009) Transferability of environmental favourability models in geographic space: The case of the Iberian desman (Galemys pyrenaicus) in Portugal and Spain. Ecological Modelling 220: 747-754

Benjamini Y. & Hochberg Y. (1995) Controlling the false discovery rate: a practical and powerful approach to multiple testing. Journal of the Royal Statistical Society, Series B 57: 289-300

Garcia L.V. (2003) Controlling the false discovery rate in ecological research. Trends in Ecology and Evolution 18: 553-554

Schwarz, G.E. (1978) Estimating the dimension of a model. Annals of Statistics, 6 (2): 461-464.

See Also

`p.adjust`

Examples

```
data(rotif.env)

names(rotif.env)

FDR(data = rotif.env, sp.cols = 18, var.cols = 5:17)

FDR(data = rotif.env, sp.cols = 18, var.cols = 5:17, simplif = TRUE)

my_pvalues <- data.frame(var = letters[1:5], pval = c(0.02, 0.004, 0.07, 0.03, 0.05))
FDR(pvalues = my_pvalues)
```

fuzSim	*Fuzzy similarity*

Description

This function calculates fuzzy similarity, based on a fuzzy version of the binary similarity index specified in `method`, between two binary (0 or 1) or fuzzy (between 0 and 1) variables.

Usage

```
fuzSim(x, y, method, na.rm = TRUE)
```

Arguments

`x`	numeric vector or SpatRaster layer of (optionally fuzzy) presence-absence data, with 1 meaning presence, 0 meaning absence, and values in between meaning fuzzy presence (or the degree to which each locality belongs to the set of species presences, or to which each species belongs to the locality; Zadeh, 1965). Fuzzy presence-absence can be obtained, for example, with functions `multGLM`, `multTSA` or `distPres` in this package, or with any other function that computes presence probability or spatial/environmental suitability for a species.
`y`	numeric vector or SpatRaster to overlap with 'x', of the same length and in the same order.
`method`	the similarity index to compute between x and y. Currently available options are "Jaccard", "Sorensen", "Simpson" and "Baroni" (see Details).
`na.rm`	logical value indicating whether NA values should be ignored. The default is TRUE.

Details

Similarity between ecological communities, beta diversity patterns, biotic regions, and distributional relationships among species are commonly determined based on pair-wise (dis)similarities in species' occurrence patterns. Some of the most commonly employed similarity indices are those of Jaccard (1901), Sorensen (1948), Simpson (1960) and Baroni-Urbani & Buser (1976), which are here implemented in their fuzzy versions (Barbosa, 2015), able to deal with both binary and fuzzy data. Jaccard's and Baroni's indices have associated tables of significant values (Baroni-Urbani & Buser 1976, Real & Vargas 1996, Real 1999).

Note that the Jaccard index's translation to fuzzy logic (where intersection = minimum and union = maximum) is equivalent to the weighted Jaccard index (Ioffe 2010) and to the overlap, coincidence and consistence indices of Real et al. (2010).

Jaccard's and Sorensen's indices have also been recommended as prevalence-independent metrics for evaluating the performance of models of species distributions and ecological niches (Leroy et al. 2018). These indices are equivalent to other previously recommended model evaluation metrics: the F-measure (which equals Sorensen's index), and the proxy of the F-measure for presence-background data, which equals 2 times Jaccard's index (Li and Guo 2013, Leroy et al. 2018).

Value

The function returns a value between 0 and 1 representing the fuzzy similarity between the provided 'x' and 'y' vectors. Note, for example, that Jaccard similarity can be converted to dissimilarity (or Jaccard distance) if subtracted from 1, while `1-Sorensen` is not a proper distance metric as it lacks the property of triangle inequality (see https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient).

Note

The formulas used in this function may look slighty different from some of their published versions (e.g. Baroni-Urbani & Buser 1976), not only because the letters are switched, but because here the A and B are the numbers of attributes present in each element, whether or not they are also present in the other one. Thus, our 'A+B' is equivalent to 'A+B+C' in formulas where A and B are the numbers of attributes present in one but not the other element, and our A+B-C is equivalent to their A+B+C. The formulas used here (adapted from Olivero et al. 1998) are faster to calculate, visibly for large datasets.

Author(s)

A. Marcia Barbosa

References

Barbosa A.M. (2015) fuzzySim: applying fuzzy logic to binary similarity indices in ecology. Methods in Ecology and Evolution, 6: 853-858.

Baroni-Urbani C. & Buser M.W. (1976) Similarity of Binary Data. Systematic Zoology, 25: 251-259

Ioffe S. (2010) Improved Consistent Sampling, Weighted Minhash and L1 Sketching. 2010 IEEE International Conference on Data Mining, Sydney, NSW, Australia, pp. 246-255, doi: 10.1109/ICDM.2010.80

Jaccard P. (1901) Etude comparative de la distribution florale dans une portion des Alpes et des Jura. Memoires de la Societe Vaudoise des Sciences Naturelles, 37: 547-579

Leroy B., Delsol R., Hugueny B., Meynard C. N., Barhoumi C., Barbet-Massin M. & Bellard C. (2018) Without quality presence-absence data, discrimination metrics such as TSS can be misleading measures of model performance. Journal of Biogeography, 45: 1994-2002

Li W. & Guo Q. (2013) How to assess the prediction accuracy of species presence-absence models without absence data? Ecography, 36: 788-799

Olivero J., Real R. & Vargas J.M. (1998) Distribution of breeding, wintering and resident waterbirds in Europe: biotic regions and the macroclimate. Ornis Fennica, 75: 153-175

Real R. (1999) Tables of significant values of Jaccard's index of similarity. Miscellania Zoologica 22: 29:40

Real R. & Vargas J.M (1996) The probabilistic basis of Jaccard's index of similarity. Systematic Biology 45: 380-385

Simpson, G.G. (1960) Notes on the measurement of faunal resemblance. Amer. J. Sci. 258A, 300-311

Sorensen T. (1948) A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons. Kongelige Danske Videnskabernes Selskab, 5(4): 1-34

Zadeh L.A. (1965) Fuzzy sets. Information and Control, 8: 338-353

See Also

simMat; modOverlap

Examples

```
data(rotif.env)

names(rotif.env)


# you can calculate similarity between binary species occurrence patterns:

fuzSim(rotif.env[, "Abrigh"], rotif.env[, "Afissa"], method = "Jaccard")
fuzSim(rotif.env[, "Abrigh"], rotif.env[, "Afissa"], method = "Sorensen")
fuzSim(rotif.env[, "Abrigh"], rotif.env[, "Afissa"], method = "Simpson")
fuzSim(rotif.env[, "Abrigh"], rotif.env[, "Afissa"], method = "Baroni")


# or you can model environmental favourability for these species
# and calculate fuzzy similarity between their environmental predictions
# which goes beyond the strict coincidence of their occurrence records:

fav <- multGLM(rotif.env, sp.cols = 18:19, var.cols = 5:17, step = TRUE,
FDR = TRUE, trim = TRUE, P = FALSE, Fav = TRUE) $ predictions

fuzSim(fav[, "Abrigh_F"], fav[, "Afissa_F"], method = "Jaccard")
fuzSim(fav[, "Abrigh_F"], fav[, "Afissa_F"], method = "Sorensen")
fuzSim(fav[, "Abrigh_F"], fav[, "Afissa_F"], method = "Simpson")
fuzSim(fav[, "Abrigh_F"], fav[, "Afissa_F"], method = "Baroni")
```

`fuzzyConsensus` *Fuzzy consensus among model predictions*

Description

This function takes a data frame or a (multilayer) SpatRaster map of favourability predictions (i.e., directly comparable predictions obtained from presence probability; see Fav) and it computes the consensus favourability, i.e., a row-wise weighted mean in which larger weights are assigned to models with higher loadings in the first axis of a principal components analysis (Baquero et al. 2021).

Usage

```
fuzzyConsensus(data, weights = "PCA1", simplif = TRUE, plot = TRUE,
biplot = FALSE, verbosity = 2, do.par = TRUE)
```

Arguments

`data`	matrix, data frame or (multilayer) 'SpatRaster' map containing the favourability values to combine.
`weights`	method for computing the weights for the weighted average of favourability values. Currently only "PCA1" is implemented.
`simplif`	logical value. If TRUE (the default), the output includes only the numeric vector of weighted mean favourability. If set to FALSE, the output will include also the complete PCA result (if weights="PCA1").
`plot`	logical value indicating whether to produce a barplot of the PCA axis loadings. The default is TRUE.
`biplot`	logical value indicating whether to produce a `biplot` of the PCA. The default is FALSE, as it makes computation slower.
`verbosity`	integer value indicating the amount of messages to display in the console. The default is to emit all messages available.
`do.par`	logical value indicating whether to override the current plotting parameters (restoring them on exit). The default is TRUE.

Details

Species distribution models are often computed using different modelling methods and/or climate scenarios. One way to summarize or combine them is to do a principal components analysis (PCA) of the different model predictions: The first axis of this PCA captures consistent spatial patterns in the predicted values across the different models (Araujo, Pearson, et al. 2005; Araujo, Whittaker, et al. 2005; Marmion et al. 2009; Thuiller 2004). However, the units of the PCA axes are difficult to interpret. Baquero et al. (2021) solved this by computing a weighted average of the favourability values (which are commensurable and therefore directly comparable across species and study areas; Real et al. 2006, Acevedo & Real 2012), using the loadings of the first PCA axis as weights. The result is therefore in the same scale as favourability, and it incorporates the degree of consensus among models, which dictates how much weight each model has in the prediction, thus avoiding disparate predictions to be blindly mixed and averaged out (Baquero et al. 2021).

Value

If simplif=TRUE (the default), the function returns a numeric vector with length equal to the number of rows in 'data' (if 'data' is a matrix or data frame), or a 'SpatRaster' layer (if 'data' is a 'SpatRaster' object), with the consensus among the input favourabilities. If simplif=FALSE, the function returns a list containing, additionally, the output of `prcomp`.

Author(s)

A. Marcia Barbosa

References

Acevedo P. & Real R. (2012) Favourability: Concept, distinctive characteristics and potential usefulness. Naturwissenschaften, 99: 515-522

Araujo M.B., Pearson R.G., Thuiller W. & Erhard M. (2005) Validation of species-climate impact models under climate change. Global Change Biology, 11: 1504-1513

Araujo M.B., Whittaker R.J., Ladle R.J. & Erhard M. (2005) Reducing uncertainty in projections of extinction risk from climate change. Global Ecology and Biogeography, 14: 529-538

Baquero R.A., Barbosa A.M., Ayllon D., Guerra C., Sanchez E., Araujo M.B. & Nicola G.G. (2021) Potential distributions of invasive vertebrates in the Iberian Peninsula under projected changes in climate extreme events. Diversity and Distributions, 27(11): 2262-2276

Marmion M., Parviainen M., Luoto M., Heikkinen R.K. & Thuiller W. (2009) Evaluation of consensus methods in predictive species distribution modelling. Diversity and Distributions 15: 59-69

Real R., Barbosa A.M. & Vargas J.M. (2006) Obtaining environmental favourability functions from logistic regression. Environmental and Ecological Statistics 13: 237-245

Thuiller W. (2004) Patterns and uncertainties of species' range shifts under climate change. Global Change Biology, 10: 2020-2027

See Also

weighted.mean

Examples

```
## Not run:
# this example requires having the 'gam' package installed

data(rotif.env)

library(gam)


# get two different model predictions for one of the species in this dataset:

names(rotif.env)
vars <- names(rotif.env)[5:17]

form_glm <- as.formula(paste("Ttetra ~", paste(vars, collapse = "+")))
mod_glm <- glm(form_glm, family = binomial, data = rotif.env)
pred_glm <- predict(mod_glm, rotif.env, type = "response")

form_gam <- as.formula(paste("Ttetra ~", paste("s(", vars, ")", collapse = "+")))
mod_gam <- gam(form_gam, family = binomial, data = rotif.env)
pred_gam <- predict(mod_gam, rotif.env, type = "response")


# convert probability predictions to favourability:

fav_glm <- Fav(pred = pred_glm, sample.preval = prevalence(model = mod_glm))
fav_gam <- Fav(pred = pred_gam, sample.preval = prevalence(model = mod_gam))
```

```
# compute the consensus favourability of these two models:

fav_consensus <- fuzzyConsensus(cbind(fav_glm, fav_gam))

cor(cbind(fav_glm, fav_gam, fav_consensus))

## End(Not run)
```

---

fuzzyOverlay   *Overlay operations based on fuzzy logic*

Description

Logical and set operations are useful for comparative distribution modelling, to assess consensus or mismatches between the predictions of different models, and to quantify differences between models obtained for different time periods. Fuzzy set theory (Zadeh 1965, Barbosa & Real 2012) allows performing such operations without converting model predictions from continuous to binary, thus avoiding the application of arbitrary thresholds and the distortion or over-simplification of those predictions. The result is a continuous numerical value quantifying the intersection, union, sum, or other operation among model predictions, whether binary or continuous.

Usage

```
fuzzyOverlay(data, overlay.cols = NULL, op = "intersection",
na.rm = FALSE, round.digits = 2)
```

Arguments

`data` matrix, data frame, or multilayer SpatRaster containing the model predictions to compare.

`overlay.cols` vector of the names or index numbers of the columns or layers to compare. The default is all columns or layers in `data`.

`op` character value indicating the operation to perform between the (specified) prediction columns or layers in 'data'. Options are:

- "consensus" for the arithmetic mean of predictions (or the fuzzy equivalent of the proportion of models that agree that the species can potentially occur at each site);
- "fuzzy_and" or "intersection" for fuzzy intersection (minimum value; Zadeh, 1965);
- "fuzzy_or" or "union" for fuzzy union (maximum value; Zadeh, 1965);
- "prob_and" or "prob_or" for probabilistic and/or, respectively (see Details);
- "maintenance" for the values where all predictions for the same row/pixel (rounded to the number of digits specified in 'round.digits') are the same.

If 'data' has only two columns/layers to compare, further options are:

- "xor" for exclusive 'or'
- "AnotB" for the the occurrence of the species in column/layer 1 in detriment of that in column/layer 2;
- "expansion" for the prediction increase in rows/pixels where column/layer 2 has higher values than column/layer 1;
- "contraction" for the prediction decrease in rows/pixels where column/layer 2 has lower values than column/layer 1;
- "change" for a mix of the latter two, with positive values where there is an increase and negative values where there is a decrease in favourability from columns/layers 1 to 2. For expansion, contraction and maintenance, rows/pixels where the values do not satisfy the condition (i.e. second column/layer larger, smaller, or roughly equal to the first) get a value of zero.

`na.rm` logical value indicating if NA values should be ignored. The default is FALSE, so rows/pixels with NA in any of the prediction columns/layers get NA as a result.

`round.digits` integer value indicating the number of decimal places to be used if op = "maintenance". The default is 2.

Details

If your predictions are probabilities, "prob_and" (probabilistic 'and') gives the probability of all species in 'data' occurring simultaneously by multiplying all probabilities; and "prob_or" (probabilistic 'or') gives the probability of any of them occurring at each site. These can be quite restrictive, though; probabilistic "and" can give particularly irrealistically small values.

If you have (or convert your probabilities to) favourability predictions, which can be used directly with fuzzy logic (Real et al. 2006; see `Fav` function), you can use "fuzzy_and" or "intersection" to get the favourability for all species to co-occur at each site, and "fuzzy_or" or "union" to get favourability for any of them to occur at each site (Barbosa & Real 2012).

Value

This function returns a vector with length equal to the number of rows in 'data', or (if the input is a SpatRaster) a SpatRaster layer of the same dimensions as the input's first layer, containing the row-wise or pixel-wise result of the operation performed.

Author(s)

A. Marcia Barbosa

References

Barbosa A.M. & Real R. (2012) Applying fuzzy logic to comparative distribution modelling: a case study with two sympatric amphibians. The Scientific World Journal, 2012, Article ID 428206

Real R., Barbosa A.M. & Vargas J.M. (2006) Obtaining environmental favourability functions from logistic regression. Environmental and Ecological Statistics 13: 237-245.

Zadeh, L.A. (1965) Fuzzy sets. Information and Control, 8: 338-353

See Also

fuzSim, modOverlap and fuzzyRangeChange for overall (not row-wise or pixel-wise) comparisons among model predictions.

Examples

```
data(rotif.env)

names(rotif.env)


# get model predictions for 3 of the species in rotif.env:

mods <- multGLM(rotif.env, sp.cols = 18:20, var.cols = 5:17, id.col = 1,
step = TRUE, FDR = TRUE, trim = TRUE)

preds <- mods$predictions[ , c("Abrigh_F", "Afissa_F", "Apriod_F")]


# calculate intersection and union among those predictions:

preds$intersect <- fuzzyOverlay(preds, op = "intersection")

preds$union <- fuzzyOverlay(preds, op = "union")

head(preds)


# imagine you have a model prediction for species 'Abrigh' in a future time
# (here we will create one by randomly jittering the current predictions)

preds$Abrigh_imag <- jitter(preds[ , "Abrigh_F"], amount = 0.2)
preds$Abrigh_imag[preds$Abrigh_imag < 0] <- 0
preds$Abrigh_imag[preds$Abrigh_imag > 1] <- 1


# you can calculate row-wise prediction changes from Abrigh to Abrigh_imag:

preds$Abrigh_exp <- fuzzyOverlay(preds, overlay.cols = c("Abrigh_F",
"Abrigh_imag"), op = "expansion")

preds$Abrigh_contr <- fuzzyOverlay(preds, overlay.cols = c("Abrigh_F",
"Abrigh_imag"), op = "contraction")

preds$Abrigh_chg <- fuzzyOverlay(preds, overlay.cols = c("Abrigh_F",
"Abrigh_imag"), op = "change")

preds$Abrigh_maint <- fuzzyOverlay(preds, overlay.cols = c("Abrigh_F",
"Abrigh_imag"), op = "maintenance")

head(preds)
```

`fuzzyRangeChange` *Range change based on continuous (fuzzy) values*

Description

This function quantifies overall range change (expansion, contraction, maintenance and balance) based on either presence-absence data or the continuous predictions of two models.

Usage

```
fuzzyRangeChange(pred1, pred2, number = TRUE, prop = TRUE,
na.rm = TRUE, round.digits = 2,
measures = c("Gain", "Loss", "Stable positive", "Stable negative", "Balance"),
plot = TRUE, plot.type = "waterfall", x.lab = TRUE, ...)
```

Arguments

`pred1`	numeric vector or SpatRaster layer containing the values (between 0 and 1) that will serve as reference.
`pred2`	numeric vector or SpatRaster layer containing the values (between 0 and 1) whose change will be computed. Must be of the same dimensions and in the same order as 'pred1'.
`number`	logical value (default TRUE) indicating if results should include the fuzzy number of cases.
`prop`	logical value (default TRUE) indicating if results should include the proportion of the total number of cases.
`na.rm`	logical value (default TRUE) indicating whether NA values should be ignored.
`round.digits`	argument to pass to `fuzzyOverlay`, indicating the number of decimal places to which to round 'pred' for calculating 'maintenance' or 'stability'. The default is 2.
`measures`	character vector listing the range change measures to calculate. The default includes all available measures.
`plot`	logical value (default TRUE) indicating whether to plot the results.
`plot.type`	(if plot=TRUE) character value indicating the type of plot to produce. Can be "barplot" or "waterfall" (the default, synonyms) or "lollipop".
`x.lab`	logical value indicating whether to add the x axis labels to the plot (i.e., the names below each lollipop or bar). The default is TRUE, but users may set it to FALSE and then add labels differently (e.g. with different names or rotations).
`...`	(if plot=TRUE) additional arguments to pass to `barplot` (if 'plot.type' is "barplot" or "waterfall") or to `modEvA::lollipop` (if 'plot.type' is "lollipop").

Value

This function returns a data frame with the following values in different rows (among those included in 'measures'):

- `Gain` sum of the predicted values that have increased from 'pred1' to 'pred2' (fuzzy equivalent of the number of localities that gained presence)
- `Loss` sum of the predicted values that have decreased from 'pred1' to 'pred2' (fuzzy equivalent of the number of localities that lost presence)
- `Stable positive` fuzzy equivalent of the number of (predicted) presences that have remained as such (when rounded to 'round.digits') between 'pred1' and 'pred2'
- `Stable negative` fuzzy equivalent of the number of (predicted) absences that have remained as such (when rounded to 'round.digits') between 'pred1' and 'pred2')
- `Balance` sum of the change in predicted values from 'pred1' to 'pred2' (fuzzy equivalent of the balance of gained and lost presences)

If number=TRUE (the default), there is a column named "Number" with the number of localities in each of the above categories. If prop=TRUE (the default), there is a column named "Proportion" in which this number is divided by the total number of reference values (i.e., the fuzzy range or fuzzy non-range size). If plot=TRUE (the default), a plot is also produced (by default, a barplot or waterfall plot) from the values in the last column of the output data frame.

Author(s)

A. Marcia Barbosa

See Also

`fuzSim`, `modOverlap` for other ways to compare models; `fuzzyOverlay` for row-wise or pixel-wise model comparisons

Examples

```
# get an environmental favourability model for a rotifer species:

data(rotif.env)

names(rotif.env)

fav_current <- multGLM(rotif.env, sp.cols = 18, var.cols = 5:17,
step = TRUE, FDR = TRUE, trim = TRUE, P = FALSE, Fav = TRUE) $
predictions


# imagine you have a model prediction for this species in a future time
# (here we will create one by randomly jittering the current predictions)

fav_imag <- jitter(fav_current, amount = 0.2)
fav_imag[fav_imag < 0] <- 0
```

```
fav_imag[fav_imag > 1] <- 1


# calculate range change given by current and imaginary future predictions:

fuzzyRangeChange(fav_current, fav_imag)

fuzzyRangeChange(fav_current, fav_imag, las = 2)

fuzzyRangeChange(fav_current, fav_imag, prop = FALSE)

fuzzyRangeChange(fav_current, fav_imag, ylim = c(-0.3, 0.3))

fuzzyRangeChange(fav_current, fav_imag, plot.type = "barplot")
```

---

getPreds *Get model predictions*

Description

This function allows getting the predictions of multiple models when applied to a given dataset. It can be useful if you have a list of model objects (e.g. resulting from `multGLM`) and want to apply them to a new data set containing the same variables for another region or time period. There are options to include the logit link ('Y') and/or 'Favourability' (see `Fav`).

Usage

```
getPreds(data, models, id.col = NULL, Y = FALSE, P = TRUE,
Favourability = TRUE, incl.input = FALSE, verbosity = 2)
```

Arguments

`data`	an object of class either 'data.frame' or 'RasterStack' to which to apply the 'models' (below) to get their predictions; must contain all variables (with the same names, case-sensitive) included in any of the 'models'.
`models`	an object of class 'list' containing one or more model objects, obtained e.g. with function `glm` or `multGLM`.
`id.col`	optionally, the index number of a column of 'data' containing row identifiers, to be included in the result. Ignored if incl.input = TRUE, or if 'data' is a RasterStack rather than a data frame.
`Y`	logical, whether to include the logit link (y) value in the predictions.
`P`	logical, whether to include the probability value in the predictions.
`Favourability`	logical, whether to include Favourability in the predictions (see `Fav`).
`incl.input`	logical, whether to include input columns in the output data frame (if the 'data' input is a data frame too). The default is FALSE.
`verbosity`	numeric value indicating the amount of messages to display; currently meaningful values are 0, 1, and 2 (the default).

Value

This function returns the model predictions in an object of the same class as the input 'data', i.e. either a data frame or a RasterStack.

Author(s)

A. Marcia Barbosa

See Also

multGLM, predict

Examples

```
data(rotif.env)

names(rotif.env)


# identify rotifer data in the Eastern and Western hemispheres:

unique(rotif.env$CONTINENT)

rotif.env$HEMISPHERE <- "Eastern"

rotif.env$HEMISPHERE[rotif.env$CONTINENT %in%
c("NORTHERN_AMERICA", "SOUTHERN_AMERICA")] <- "Western"

head(rotif.env)


# separate the rotifer data into hemispheres

east.hem <- rotif.env[rotif.env$HEMISPHERE == "Eastern", ]
west.hem <- rotif.env[rotif.env$HEMISPHERE == "Western", ]


# make models for 3 of the species in rotif.env based on their distribution
# in the Eastern hemisphere:

mods <- multGLM(east.hem, sp.cols = 18:20, var.cols = 5:17,
id.col = 1, step = FALSE, FDR = FALSE, trim = FALSE)


# get the models' predictions for the Western hemisphere dataset:

preds <- getPreds(west.hem, models = mods$models, P = TRUE,
Favourability = TRUE)

head(preds)
```

getRegion *Get region*

Description

This function computes a (possibly multi-part) polygon around a set of point coordinates under given criteria, useful for delineating background or (pseudo)absence regions for computing species distibution models. Some of the 'type' options, namely those involving clusters or inverse distance, address survey bias by making smaller polygons around areas with fewer and/or more isolated points, as described in Barbosa (2026).

Usage

```
getRegion(
  pres.coords,
  type = "width",
  clust_dist = 100,
  clust_type = "buffer",
  dist_mult = 1,
  width_mult = 0.5,
  weight = FALSE,
  decay = TRUE,
  CRS = NULL,
  dist_mat = NULL,
  dist_method = "auto",
  verbosity = 2,
  plot = TRUE,
  col_reg = "gold",
  prj = NULL,
  ...
)
```

Arguments

`pres.coords` a SpatVector of points, or an object inheriting class 'data.frame' with 2 columns containing, respectively, the x and y, or longitude and latitude coordinates (in this order!) of the points where species presence was recorded.

`type` character indicating which procedure to use for defining the region around 'pres.coords'. Options are:

- "width": a buffer whose radius is the minimum diameter of the 'pres.coords' spatial extent (computed with `terra::width()`);
- "mean_dist": a buffer whose radius is the mean pairwise `terra::distance()` among 'pres.coords';
- "inv_dist": a buffer whose radius is the mean pairwise `terra::distance()` among 'pres.coords', downweighted by the sum of the square-rooted distances from each point to all other points in 'pres.coords' - a rough measure

of how isolated each point is, possibly indicating opportunistic records in sparsely surveyed areas (Barbosa, 2026);

- "clust_mean_dist": a different buffer around each cluster of 'pres.coords' (clusters computed as described in 'clust_type'), sized according to the mean pairwise distance between the points in that cluster (Barbosa, 2026).
- "clust_width": a different buffer around each cluster of 'pres.coords' (clusters computed as described in 'clust_type'), sized according to the `terra::width()` of that cluster (Barbosa, 2026).

`clust_dist` if 'type' involves clusters, numeric value specifying the distance threshold (in km) within which points are clustered together. Default 100.

`clust_type` if 'type' involves clusters, character value specifying the method to compute them. Options are:

- "buffer" (now the default, more recently implemented), for aggregated `terra::buffer`s of width = 'clust_dist';
- "hclust", for clusters computed with `stats::hclust()`, method = "simple") and then `stats::cutree()` with h = clust_dist*1000 (backward-compatible, but less accurate, and much more computationally intensive if 'dist_mat' is not provided).

`dist_mult` if 'type' involves distance, multiplier for the mean distance to use for the `terra::buffer()` radius. Default 1.

`width_mult` if 'type' involves width, multiplier of the width to use for the `terra::buffer()` radius. Default 0.5.

`weight` (if 'type' includes clusters) logical value indicating whether to weigh the radius of the buffer around each cluster proportionally to the number of points that it includes. Default FALSE; if set to TRUE, clusters with fewer points (possibly indicating more sparsely surveyed areas) get proportionally smaller buffers than their mean distances or widths.

`decay` if type="inv_dist") logical value (default TRUE) indicating whether to use a distance-decay kernel (negative exponential function) to make farther points weight progressively less.

`CRS` coordinate reference system of 'pres.coords' (if it is not a SpatVector with a defined CRS already), in one of the following formats: WKT/WKT2, <authority>:<code>, or PROJ-string notation (see `terra::crs()`). If not defined and the coordinates fall within valid latitude-longitude bounds, EPSG:4326 is assumed. Note that input points in an unprojected CRS, with longitude-latitude coordinates in degrees, allow more accurate spherical distance calculations because they account for the curvature of the Earth (see e.g. `terra::distance()` and `terra::buffer()`).

`dist_mat` optional matrix of pairwise distances among 'pres.coords', to use (if 'type' or 'clust_type' implies computing distances) for efficiency instead of computing a new one. Should normally be computed with `terra::distance()`, `geodist::geodist()`, or another method that takes the Earth's curvature into account. If not provided, it is computed with `distMat`.

`dist_method` (if 'dist_mat' is NULL) argument to pass to `distMat` specifying the 'method' for distance calculation. The general default is "auto". However, if type="clust_mean_dist", to avoid different clusters getting a different automatically assigned distance method, the method is "haversine" if coordinates are longitude-latitude degrees, or "euclidean" if they are projected.

`verbosity` integer indicating the amount of messages to display along the process. The default is 2, for all available messages.

`plot` logical (default TRUE) indicating whether to plot the resulting region polygon, together with the input 'pres.coords' (black points, or points coloured according to their cluster) and a label with the number of points in each cluster (if 'type' involves clusters).

`col_reg` (if plot=TRUE) colour for the region polygon. Defaults to "gold".

`prj` (if plot=TRUE) optional CRS for projecting the region polygon and occurrence points after computing, just for plotting purposes, in one of the following formats: WKT/WKT2, <authority>:<code>, or PROJ-string notation (see `terra::crs()`). Can be useful particularly when using add=TRUE to overlay the output plot on a previous map in a different CRS.

`...` (if plot=TRUE) additional arguments to pass to `terra::plot()`.

Details

Most methods for computing species distribution models require predictor values for regions beyond those with species occurrence records, i.e. background or (pseudo)absence areas. The extent (as well as the spatial resolution) of these regions has a strong effect on model predictions. Ideally, they should include the areas that are within the reach of the species AND were reasonably surveyed (though you can further refine the latter with `selectAbsences` and an optional `biasLayer`). While sometimes we have a large enough and delimited area that we can use (e.g. when modelling a region where a national or regional distribution atlas is available), often we need to approximate the areas that appear to be both reasonably surveyed and within the species' reach.

The procedures implemented in this function, aimed at approximating accessible and (roughly) surveyed areas given a set of occurrence points, are described in Barbosa (2026). Mind that no automated procedure can properly address all possible issues related to uneven data collection, or properly conform to all possible species distribution and survey patterns. Mind also that the output region from this function does not currently consider geographical barriers, or other factors that should also be taken into account when delimiting a region for modelling.

It is thus recommended to try different values for 'type' and associated parameters; judge for yourself which one provides the most plausible approximation to the surveyed region accessible to your target species; and possibly post-process (i.e. further edit) the resulting region in light of the available knowledge of that species' distribution, survey patterns and study region.

Value

SpatVector (possibly multi-part) polygon delimiting a region around 'pres.coords'

Author(s)

A. Marcia Barbosa

References

Barbosa, A.M. (2026) Spatially adaptive background delimitation strategies for species distribution models with unevenly surveyed occurrences. Biodiversity Informatics, in press

See Also

terra::buffer(), terra::width(), terra::crop()

Examples

```
## Not run:
# you can run these examples if you have 'terra' and 'geodata' installed

# download example data:

occs <- geodata::sp_occurrence("Triturus", "pygmaeus")

occs_sv <- terra::vect(occs, geom = c("lon", "lat"), crs = "EPSG:4326")

cntry <- geodata::world(path = tempdir())


terra::plot(occs_sv)

terra::plot(cntry, lwd = 0.2, add = TRUE)


# compute regions with some different methods:

reg1 <- fuzzySim::getRegion(occs_sv)

terra::plot(cntry, lwd = 0.2, add = TRUE)


reg2 <- fuzzySim::getRegion(occs_sv, type = "inv_dist")

terra::plot(cntry, lwd = 0.2, add = TRUE)

terra::plot(reg2, lwd = 3, border = "orange", add = TRUE)


reg3 <- fuzzySim::getRegion(occs_sv, type = "clust_width", weight = TRUE,
width_mult = 0.3)

terra::plot(cntry, lwd = 0.2, add = TRUE)

terra::plot(reg3, lwd = 3, border = "orange", add = TRUE)


# note it is up to the user to pre-process the data
# (e.g. by removing duplicate or erroneous records)
# and/or post-process the region (e.g. terra::erase() areas
```

```
# that are unsurveyed or inaccessible to the target species)

## End(Not run)
```

`gridRecords` *Grid (or thin) point occurrence records to the resolution of a raster map*

Description

This function takes a (single or multi-layer) SpatRaster (or a Raster* object for backward compatibility, though this is no longer being developed) and a set of spatial coordinates of a species' presence (and optionally absence) records, and it returns a data frame of the presences and absences with their raster values in the grid of pixels (cells). This is analogous to removing duplicates and thinning points (both presences and absences) with a distance equal to the raster pixel size.

Usage

```
gridRecords(rst, pres.coords, abs.coords = NULL, absences = TRUE,
species = NULL, na.rm = TRUE, plot = FALSE, ...)
```

Arguments

`rst`	a SpatRaster object (or a much slower Raster* object, just for back-compatibility) with the desired spatial resolution and extent for the species presence-(pseudo)absence data, and the layer(s) whose values to extract for those data.
`pres.coords`	a SpatVector of points, or an object inheriting class 'data.frame' with 2 columns containing, respectively, the x and y, or longitude and latitude coordinates (in this order, and in the same coordinate reference system as 'rst'!) of the points where species presence was recorded.
`abs.coords`	(optional) same as 'pres.coords' but for points where the species was not recorded. If abs.coords=NULL and absences=TRUE (the default), all pixels that are not intersected by 'pres.coords' will be returned as absences (of occurrence records). Curently cannot be used if 'species' provided.
`absences`	logical value (default TRUE) indicating if pixels without presence points should be returned as absences (of occurrence records). FALSE can currently not be used if 'species' provided.
`species`	(optional) character vector, of the same length as 'nrow(pres.coords)', indicating the species to which each pair of coordinates corresponds. Useful for gridding records of more than one species at a time. Its unique values will be used as column names in the output. If this argument is specified, 'absences=FALSE' and 'abs.coords' cannot be used, and 'plot' is set to FALSE.
`na.rm`	logical value indicating whether pixels with NA in all of the 'rst' layers should be excluded from the output data frame. The default is TRUE.

`plot`	logical value specifying whether to plot the resulting presence and (pseudo)absence points. The default is FALSE for backward compatibility. Automatically set to FALSE if 'species' is not NULL.
`...`	(if plot=TRUE) some additional arguments that can be passed to `plot`, e.g "main" (plot title).

Details

See e.g. Baez et al. (2020), where this function was first used to get unique presences and absences from point occurrence data at the spatial resolution of marine raster variables.

Consider cleaning the coordinates beforehand, e.g. with `cleanCoords`. If your output has an overly large and/or spatially biased set of absences, it may be recommendable to use `selectAbsences` afterwards.

Value

This function returns a data frame with the following columns:

`'presence'`	integer, 1 for the cells (pixels) with one or more presence points; and (if absences=TRUE) 0 for the cells with no presence points, or (if 'abs.coords' are provided) for the cells with one or more absence points AND no presence points. If the 'species' argument is provided, instead of 'presence' you get one column named as each species.
`'x', 'y'`	centroid coordinates of each cell (pixel).
`'cell'`	the pixel identifier in 'rst'.
`one column for each layer in 'rst'`	value of each pixel for each layer.

If plot=TRUE, the fuction also plots the resulting presences (blue "plus" signs) and absences (red "minus" signs).

Note

This function requires either the **raster** or the **terra** package, depending on the class of 'rst'. It may crash for large rasters, in which case it is advisable to first divide them into tiles.

Author(s)

A. Marcia Barbosa

References

Baez J.C., Barbosa A.M., Pascual P., Ramos M.L. & Abascal F. (2020) Ensemble modelling of the potential distribution of the whale shark in the Atlantic Ocean. Ecology and Evolution, 10: 175-184

See Also

`cleanCoords`, `selectAbsences`, `getRegion`

Examples

```
## Not run:

# you can run these examples if you have the 'terra' package installed

require(terra)


# import a raster map and aggregate it to a coarser resolution:

r <- terra::rast(system.file("ex/elev.tif", package = "terra"))

r <- terra::aggregate(r, 6)

plot(r)


# generate some random presence and absence points:

set.seed(321)

presences <- terra::spatSample(as.polygons(r), 100)

absences <- terra::spatSample(as.polygons(r), 70)


# add these points to the map:

points(presences, pch = 20, cex = 0.3, col = "black")

points(absences, pch = 20, cex = 0.3, col = "white")


# use 'gridRecords' on these points:

gridded_pts <- gridRecords(rst = r, pres.coords = terra::crds(presences),
abs.coords = terra::crds(absences))

head(gridded_pts)


# map the gridded points (presences black, absences white):

points(gridded_pts[ , c("x", "y")], col = gridded_pts$presence)


# you can also do it with only presence (no absence) records
# in this case, by default (with 'absences = TRUE'),
# all pixels without presence points are returned as absences:

gridded_pres <- gridRecords(rst = r, pres.coords = terra::crds(presences))
```

```
head(gridded_pres)


plot(r)

points(presences, pch = 20, cex = 0.2, col = "black")

points(gridded_pres[ , c("x", "y")], col = gridded_pres$presence)


# with only presence (no absence) records, as in this latter case,
# you can grid records for multiple species at a time
# by adding a 'species' argument

presences$species <- rep(c("species1", "species2", "species3"), each = 33)

values(presences)


plot(r, col = hcl.colors(n = 100, palette = "blues"))

plot(presences, col = as.factor(presences$species), add = TRUE)

gridded_pres_mult <- gridRecords(rst = r, pres.coords = terra::crds(presences),
species = presences$species)

head(gridded_pres_mult)


# add each each species' gridded presences to the map:

points(gridded_pres_mult[gridded_pres_mult[ , 1] == 1, c("x", "y")], col = 1, pch = 1)

points(gridded_pres_mult[gridded_pres_mult[ , 2] == 1, c("x", "y")], col = 2, pch = 2)

points(gridded_pres_mult[gridded_pres_mult[ , 3] == 1, c("x", "y")], col = 3, pch = 3)


# a large 'rst' may cause a crash, in which case you can grid in parts:

dir.create("gridRecords_tiles")  # creates a folder to receive the tile files

terra::makeTiles(r, terra::divide(r, 2),
                 filename = "gridRecords_tiles/tile_.tif")

# give a larger 'n' to divide() if your 'rst' still crashes 'gridRecords'

tiles <- terra::sprc(list.files("gridRecords_tiles", full.names = TRUE))


par(mfrow = c(2, 2))

sapply(tiles, plot)
```

```
gridded_list <- lapply(tiles, gridRecords,
                       pres.coords = terra::crds(presences),
                       plot = TRUE)

gridded_pres <- do.call(rbind, unname(gridded_list))

unlink("gridRecords_tiles", recursive = TRUE)  # deletes the tiles folder

## End(Not run)
```

---

integerCols *Classify integer columns*

Description

This function detects which numeric columns in a data frame contain only whole numbers, and converts those columns to integer class, so that they take up less space.

Usage

```
integerCols(data)
```

Arguments

data	a matrix or an object inheriting class data.frame containing possibly integer columns classified as "numeric".

Value

The function returns a data frame with the same columns as 'data', but with those that are numeric and contain only whole numbers (possibly including NA) now classified as "integer".

Author(s)

A. Marcia Barbosa

See Also

is.integer, as.integer, multConvert

Examples

```
dat <- data.frame(
  var1 = 1:10,
  var2 = as.numeric(1:10),
  var3 = as.numeric(c(1:4, NA, 6:10)),
  var4 = as.numeric(c(1:3, NaN, 5, Inf, 7, -Inf, 9:10)),
```

```
  var5 = as.character(1:10),
  var6 = seq(0.1, 1, by = 0.1),
  var7 = letters[1:10]
)  # creates a sample data frame

dat

str(dat)
# var2 classified as "numeric" but contains only whole numbers
# var3 same as var2 but containing also NA values
# var4 same as var2 but containing also NaN and infinite values
# var5 contains only whole numbers but initially classified as factor

dat <- integerCols(dat)

str(dat)
# var2 and var3 now classified as "integer"
# var4 remains as numeric because contains infinite and NaN
# (not integer) values
# var5 remains as factor
```

`modelTrim` *Trim off non-significant variables from a model*

Description

This function performs a stepwise removal of non-significant variables from a model object, following Crawley (2005, 2007).

Usage

```
modelTrim(model, method = "summary", alpha = 0.05, verbosity = 2, phy = NULL)
```

Arguments

`model`	a model object of class 'lm', 'glm' or 'phylolm'.
`method`	the method for getting the p-value of each variable. Can be either "summary" for the p-values of the coefficient estimates, or (if the model class is 'lm' or 'glm') "anova" for the p-values of the variables themselves (see Details).
`alpha`	the threshold p-value above which a variable is to be removed.
`verbosity`	integer number indicating the amount of messages to display; the default is the maximum number of messages available.
`phy`	if 'model' is of class 'phylolm', the phylogenetic tree to pass to phylolm::phylolm() when re-computing the model after the removal of each non-significant variable.

Details

Stepwise variable selection is a common procedure for simplifying models. It maximizes predictive efficiency in an objective and reproducible way, and it is useful when the individual importance of the predictors is not known a priori (Hosmer & Lemeshow, 2000). The `step` R function performs such procedure using an information criterion (AIC) to select the variables, but it often leaves variables that are not significant in the model. Such variables can be subsequently removed with a stepwise procedure (e.g. Crawley 2005, p. 208; Crawley 2007, p. 442 and 601; Barbosa & Real 2010, 2012; Estrada & Arroyo 2012). The 'modelTrim' function performs such removal automatically until all remaining variables are significant. It can also be applied to a full model (i.e., without previous use of the 'step' function), as it serves as a backward stepwise selection procedure based on the significance of the coefficients (if method = "summary", the default) or on the significance of the variables (if method = "anova", better when there are categorical variables in the model). See also `stepwise` for a more complete stepwise selection procedure based on a data frame.

Value

The updated input model object after stepwise removal of non-significant variables.

Author(s)

A. Marcia Barbosa

References

Barbosa A.M. & Real R. (2010) Favourable areas for expansion and reintroduction of Iberian lynx accounting for distribution trends and genetic diversity of the European rabbit. Wildlife Biology in Practice 6: 34-47

Barbosa A.M. & Real R. (2012) Applying fuzzy logic to comparative distribution modelling: a case study with two sympatric amphibians. The Scientific World Journal, Article ID 428206

Crawley, M.J. (2005) Statistics: An introdution using R. John Wiley & Sons, Ltd.

Crawley, M.J. (2007) The R Book. John Wiley & Sons, Ltd.

Estrada A. & Arroyo B. (2012) Occurrence vs abundance models: Differences between species with varying aggregation patterns. Biological Conservation, 152: 37-45

Hosmer D. W. & Lemeshow S. (2000) Applied Logistic Regression (2nd ed). John Wiley and Sons, New York

See Also

`step`, `stepwise`

Examples

```
# load sample data:

data(rotif.env)

names(rotif.env)
```

```
# build a stepwise model of a species' occurrence based on
# some of the variables:

mod <- with(rotif.env, step(glm(Abrigh ~ Area + Altitude + AltitudeRange +
HabitatDiversity + HumanPopulation, family = binomial)))


# examine the model:

summary(mod)  # includes non-significant variables

# use modelTrim to get rid of those:

mod <- modelTrim(mod)

summary(mod)  # only significant variables remain
```

`modOverlap` *Overall overlap between model predictions*

Description

This function calculates the degree of overlap between the predictions of two models, using niche comparison metrics such as Schoener's D, Hellinger distance and Warren's I.

Usage

```
modOverlap(pred1, pred2, na.rm = TRUE)
```

Arguments

`pred1`	numeric vector or SpatRaster layer of the predictions of a model, with values between 0 and 1.
`pred2`	numeric vector or SpatRaster layer of the predictions of another model, also with values between 0 and 1; must be of the same dimensions and in the same order as 'pred1'.
`na.rm`	logical value indicating whether NA values should be removed prior to calculation. The default is TRUE.

Details

See Warren et al. (2008).

Value

This function returns a list of 3 metrics:

`SchoenerD` Schoener's (1968) D statistic for niche overlap, varying between 0 (no overlap) and 1 (identical niches).

`WarrenI` the I index of Warren et al. (2008), based on Hellinger distance (below) but re-formulated to also vary between 0 (no overlap) and 1 (identical niches).

`HellingerDist` Hellinger distance (as in van der Vaart 1998, p. 211) between probability distributions, varying between 0 and 2.

Author(s)

A. Marcia Barbosa

References

Schoener T.W. (1968) Anolis lizards of Bimini: resource partitioning in a complex fauna. Ecology 49: 704-726

van der Vaart A.W. (1998) Asymptotic statistics. Cambridge Univ. Press, Cambridge (UK)

Warren D.L., Glor R.E. & Turelli M. (2008) Environmental niche equivalency versus conservatism: quantitative approaches to niche evolution. Evolution, 62: 2868-83 (and further ERRATUM)

See Also

`fuzSim`; `fuzzyOverlay`; `niche.overlap` in package **phyloclim**; `ecospat.niche.overlap` in package **ecospat**

Examples

```
# get an environmental favourability model for a rotifer species:

data(rotif.env)

names(rotif.env)

fav_current <- multGLM(rotif.env, sp.cols = 18, var.cols = 5:17,
step = TRUE, FDR = TRUE, trim = TRUE, P = FALSE, Fav = TRUE) $
predictions


# imagine you have a model prediction for this species in a future time
# (here we will create one by randomly jittering the current predictions)

fav_imag <- jitter(fav_current, amount = 0.2)
fav_imag[fav_imag < 0] <- 0
fav_imag[fav_imag > 1] <- 1


# calculate niche overlap between current and imaginary future predictions:
```

```
modOverlap(fav_current, fav_imag)
```

`multConvert` *Multiple conversion*

Description

This function can simultaneously convert multiple columns of a matrix or data frame, or multiple layers of a SpatRaster.

Usage

```
multConvert(data, conversion, cols = 1:ncol(data))
```

Arguments

`data`	A matrix, an object inheriting class data frame, or a SpatRaster containing the columns or layers that need to be converted
`conversion`	the conversion to apply, e.g. `as.factor`, `scale`, `log`, or a custom-made function
`cols`	the columns or layers of 'data' to convert

Details

With this function we can change the data type (e.g. convert with `as.integer`, `as.factor` or `as.character`), `scale` or `log`-transform several variables in a data frame. By default, all columns in 'data' are converted, but you can specify just some of those in the 'cols' argument. You can also specify your own function to apply, e.g. a division/multiplication of several columns by a given number (see Examples).

Value

The input data with the specified columns converted as specified in 'conversion'.

Author(s)

A. Marcia Barbosa

Examples

```
data(rotif.env)

str(rotif.env)


# convert the first 4 columns to character:

converted.rotif.env <- multConvert(data = rotif.env,
conversion = as.character, cols = 1:4)
```

```
str(converted.rotif.env)


# divide some columns by 100:

div100 <- function(x) {
  x / 100
}

rotif.env.div100 <- multConvert(data = rotif.env,
conversion = div100, cols = c(6:10, 12:17))

head(rotif.env.div100)


# scale (standardize) continuous variables:

names(rotif.env)

conts <- names(which(sapply(rotif.env[ , 1:17], is.numeric)))

rotif.env.scaled <- multConvert(data = rotif.env,
conversion = scale, cols = conts)

head(rotif.env.scaled)
```

---

multGLM *GLMs with variable selection for multiple species*

Description

This function performs selection of variables and calculates generalized linear models for a set of presence/absence records in a data frame, with a range of options for data partition, variable selection, and output form.

Usage

```
multGLM(data, sp.cols, var.cols, id.col = NULL, block.cols = NULL,
family = "binomial", test.sample = 0, FDR = FALSE, test = "Chisq",
correction = "fdr", FDR.first = TRUE, corSelect = FALSE, coeff = TRUE,
cor.thresh = ifelse(isTRUE(coeff), 0.8, 0.05), cor.method = "pearson",
step = TRUE, trace = 0, start = "null.model", direction = "both",
select = "AIC", trim = TRUE, Y.prediction = FALSE, P.prediction = TRUE,
Favourability = TRUE, group.preds = TRUE, TSA = FALSE, coord.cols = NULL,
degree = 3, verbosity = 2, test.in = "Rao", test.out = "LRT", p.in = 0.05,
p.out = 0.1, ...)
```

Arguments

`data`	a data frame in wide format (see `splist2presabs`) containing, in separate columns, your species' binary (0/1) occurrence data and the predictor variables.
`sp.cols`	names or index numbers of the columns containing the species data to be modelled.
`var.cols`	names or index numbers of the columns containing the predictor variables to be used for modelling.
`id.col`	(optional) name or index number of column containing the row identifiers (if defined, it will be included in the output 'predictions' data frame).
`block.cols`	[UNDER IMPLEMENTATION] names or index numbers of the columns containing predictor variables to force into the model, even when a selection method is applied to the remaining variables.
`family`	argument to be passed to the `glm` function; currently, only 'binomial' is implemented here.
`test.sample`	a subset of data to set aside for subsequent model testing. Can be a value between 0 and 1 for a proportion of the data to choose randomly (e.g. 0.2 for 20%); or an integer number for a particular number of cases to choose randomly among the records in 'data'; or a vector of integers for the index numbers of the particular rows to set aside; or "Huberty" for his rule of thumb based on the number of variables (Huberty 1994, Fielding & Bell 1997).
`FDR`	logical value indicating whether to do a preliminary exclusion of variables based on the false discovery rate (see `FDR`). The default is FALSE.
`test`	argument to pass to the `FDR` function (which, in turn, passes it to `anova`) if FDR=TRUE. The default is currently "Chisq" for back-compatibility.
`correction`	argument to pass to the `FDR` function if FDR=TRUE. The default is "fdr", but see `p.adjust` for other options.
`FDR.first`	logical value indicating whether FDR exclusion (if FDR=TRUE) should be applied at the beginning. The default is TRUE. If set to FALSE (and if FDR=TRUE), FDR exclusion will be applied after 'corSelect' below.
`corSelect`	logical value indicating whether to select among highly correlated variables using `corSelect`. The default is FALSE.
`coeff`	logical value to pass to `corSelect` (if corSelect=TRUE) indicating whether two variables should be considered highly correlated based on the magnitude of their coefficient (rather than p-value) of correlation. The default is TRUE.
`cor.thresh`	numerical value indicating the correlation threshold to pass to `corSelect` (if corSelect=TRUE).
`cor.method`	character value to pass to `corSelect` (if corSelect=TRUE) specifying the correlation coefficient to use. Can be "pearson" (the default), "kendall" or "spearman".
`step`	logical, whether to perform a stepwise selection of variables, using either the `step` function (if select = "AIC" or "BIC") or the `stepwise` function (if select = "p.value").
`trace`	if positive, information is printed during the stepwise selection (if step=TRUE). Larger values may give more detailed information.

start	character string specifying whether to start with the 'null.model' (so that variable selection starts forward) or with the 'full.model' (so selection starts backward). Used only if step=TRUE.
direction	if step=TRUE, argument to be passed to step or to stepwise specifying the direction of variable selection. Can be 'forward', 'backward', or 'both' (the default).
select	character string specifying the criterion for stepwise selection of variables if step=TRUE. Options are the default "AIC" (Akaike's Information Criterion; Akaike, 1973); BIC (Bayesian Information Criterion, also known as Schwarz criterion, SBC or SBIC; Schwarz, 1978); or "p.value" (Murtaugh, 2014). The first two options imply using step as the variable selection function, while the last option calls the stepwise function. If you set `select="p.value"`, we recommend also setting `trim=FALSE` to avoid mixing different significance criteria.
trim	logical value indicating whether to trim off non-significant variables from the models using modelTrim. This argument is TRUE by default (for back-compatibility), and it can be used whether or not step=TRUE – e.g. Crawley (2005, p. 208) and Crawley (2007, p. 442 and 601) recommend that step (with AIC selection) be followed by significance-based backward elimination).
Y.prediction	logical value indicating whether to include output predictions in the scale of the predictor variables (type = "link" in predict.glm).
P.prediction	logical, whether to include output predictions in the scale of the response variable, i.e. probability (type = "response" in predict.glm).
Favourability	logical, whether to apply the Favourability function to remove the effect of prevalence on predicted probability (Real et al. 2006) and include its results in the output.
group.preds	logical, whether to group together predictions of similar type ('Y', 'P' or 'F') in the output 'predictions' table (e.g. if FALSE: sp1_Y, sp1_P, sp1_F, sp2_Y, sp2_P, sp2_F; if TRUE: sp1_Y, , sp2_Y, sp1_P, sp2_P, sp1_F, sp2_F).
TSA	logical, whether to add a trend surface analysis (calculated individually for each species) as a spatial variable in each model (with type="Y" – see multTSA for more details). The default is FALSE. If TRUE, this spatial trend will be treated as any other variable, i.e. also considered by arguments 'FDR', 'corSelect', etc.
coord.cols	argument to pass to multTSA (if TSA=TRUE).
degree	argument to pass to multTSA (if TSA=TRUE).
verbosity	numeric value indicating the amount of messages to display, from less to more verbose; currently meaningful values are 0, 1, and 2 (the default).
test.in	argument to pass to stepwise if select="p.value".
test.out	argument to pass to stepwise if select="p.value".
p.in	argument to pass to stepwise if select="p.value".
p.out	argument to pass to stepwise if select="p.value".
...	(for back-compatibility) additional arguments to be passed to modelTrim (if trim=TRUE).

Details

This function automatically calculates binomial GLMs for one or more species (or other binary variables) in a data frame. The function can optionally perform stepwise variable selection using either `stepwise` or `step` (and it does so by default) instead of forcing all variables into the models, starting from either the null model (the default, so selection starts forward) or from the full model (so selection starts backward), and using AIC, BIC or statistical significance as a variable selection criterion. Instead or subsequently, it can also perform stepwise removal of non-significant variables from the models using the `modelTrim` function.

There is also an optional preliminary selection among highly correlated variables, and/or preliminary selection of variables with a significant bivariate relationship with the response, based on the false discovery rate (`FDR`). Note, however, that some variables can be significant in a multivariate model even if they would not have been selected by FDR.

`Fav`ourability can also be calculated by default, removing the effect of training prevalence from occurrence probability and thus allowing direct comparisons between different models (Real et al. 2006; Acevedo & Real 2012).

By default, all data are used in model training, but you can define an optional 'test.sample' to be reserved for model testing afterwards. You may also want to do a previous check for multicollinearity among variables, e.g. the variance inflation factor (VIF), using `multicol`.

The 'multGLM' function will create a list of the resulting models (each with the name of the corresponding species column) and a data frame with their predictions ('Y', 'P' and/or 'F', all of which are optional). If you plan on representing these predictions in a GIS format based on .dbf tables (e.g. ESRI Shapefile), remember that .dbf only allows up to 10 characters in column names; 'multGLM' predictions will add 2 characters (_Y, _P and/or _F) to each of your species column names, so better use species names/codes with up to 8 characters in the data set that you are modelling. You can create (sub)species name abbreviations with the `spCodes` function.

Value

This function returns a list with the following components:

`predictions`	a data frame with the model predictions (if either of Y.prediction, P.prediction or Favourability are TRUE).
`models`	a list of the resulting model objects.
`variables`	a list of character vectors naming the variables finally included in each model according to the specified selection criteria.

Note

With step=TRUE (the default), an error may occur if there are missing values in some of the variables that are selected (see "Warning" in `step`). If this happens, you can use something like data=na.omit(data[, c(sp.col, var.cols)]).

Thanks are due to Prof. Jose Carlos Guerrero at the University of the Republic (Uruguay), who funded the implementation of the options `select="p.value"` and `FDR.first=FALSE`.

Author(s)

A. Marcia Barbosa

References

Acevedo P. & Real R. (2012) Favourability: concept, distinctive characteristics and potential usefulness. Naturwissenschaften, 99:515-522

Akaike, H. (1973) Information theory and an extension of the maximum likelihood principle. In: Petrov B.N. & Csaki F., 2nd International Symposium on Information Theory, Tsahkadsor, Armenia, USSR, September 2-8, 1971, Budapest: Akademiai Kiado, p. 267-281.

Crawley, M.J. (2005) Statistics: An introdution using R. John Wiley & Sons, Ltd.

Crawley, M.J. (2007) The R Book. John Wiley & Sons, Ltd.

Fielding A.H. & Bell J.F. (1997) A review of methods for the assessment of prediction errors in conservation presence/absence models. Environmental Conservation 24: 38-49

Huberty C.J. (1994) Applied Discriminant Analysis. Wiley, New York, 466 pp. Schaafsma W. & van Vark G.N. (1979) Classification and discrimination problems with applications. Part IIa. Statistica Neerlandica 33: 91-126

Murtaugh P.A. (2014) In defense of P values. Ecology, 95:611-617

Real R., Barbosa A.M. & Vargas J.M. (2006) Obtaining environmental favourability functions from logistic regression. Environmental and Ecological Statistics 13: 237-245.

Schwarz, G.E. (1978) Estimating the dimension of a model. Annals of Statistics, 6 (2): 461-464.

See Also

glm, step, stepwise

Examples

```
data(rotif.env)

names(rotif.env)


# make models for 2 of the species in rotif.env:

mods <- multGLM(rotif.env, sp.cols = 46:47, var.cols = 5:17, id.col = 1,
step = TRUE, FDR = TRUE, trim = TRUE)

names(mods)
head(mods$predictions)
names(mods$models)
mods$models[[1]]
mods$models[["Ttetra"]]


# include each species' spatial trend in the models:

mods <- multGLM(rotif.env, sp.cols = 46:47, var.cols = 5:17, id.col = 1,
step = TRUE, FDR = TRUE, trim = TRUE, TSA = TRUE, coord.cols = c(11, 10))

mods$models[[1]]
mods$models[["Ttetra"]]
```

```
mods$variables
# you can then use these selected variables elsewhere
```

---

multicol *Analyse multicollinearity in a dataset, including VIF*

Description

This function analyses multicollinearity in a set of variables or in a model, including the R-squared, tolerance and variance inflation factor (VIF). It also allows selecting variables under a given VIF.

Usage

```
multicol(vars = NULL, model = NULL, reorder = TRUE, vif.thresh = Inf,
max.nvars = Inf, verbosity = 2, simplif = FALSE, plot = FALSE, ...)
```

Arguments

vars	A matrix, an object inheriting class data.frame, or a multi-layer SpatRaster containing the numeric variables for which to calculate multicollinearity. Note that only the 'independent' (predictor, explanatory, right hand side) variables should be entered, as the result obtained for each variable depends on all the other variables present in the analysed data set.
model	Alternatively to 'vars', a model object of class "glm" to calculate 'multicol' among the included variables.
reorder	logical, whether variables should be output in decreasing order or VIF value rather than in their input order. The default is TRUE.
vif.thresh	numeric value specifying the maximum VIF allowed in the output. The default is `Inf`, for no limit. Variables above the threshold are excluded in a stepwise manner, starting from the highest one and recomputing the VIFs every time, until no variable exceeds the 'vif.thresh' value.
max.nvars	integer value specifying the maximum number of variables allowed in the output. The default is `Inf`, for no limit. If there are more variables than 'max.nvars', variables are excluded in a stepwise manner, starting from the one with the highest VIF and recomputing the VIFs every time, until the specified number of variables is reached.
verbosity	integer specifying the amount of messages to display along the process. The default is 2, for the maximum amount of messages available.
simplif	(if 'vif.thresh' is not `Inf`) logical (default FALSE) indicating whether to produce a simplified output, consisting only of a character vector of the names of the selected variables.
plot	logical value (default TRUE) indicating whether to plot the output VIF values.
...	(if plot=TRUE) additional arguments to pass to `modEvA::lollipop`.

Details

Testing (multi)collinearity among covariates is a recommended step of data exploration before applying a statistical model (Zuur et al. 2010). You can also assess the multicollinearity among the variables already included in a model.

The multicol function calculates the degree of multicollinearity in a set of numeric variables, using three closely related measures:

- R squared: the coefficient of determination of a linear regression of each predictor variable on all other predictor variables, i.e., the amount of variation in each variable that is accounted for by other variables in the dataset;
- Tolerance = 1 - R squared, i.e., the amount of variation in each variable that is not included in the remaining variables;
- Variance Inflation Factor: VIF = 1 / (1 - R squared), which, in a linear model with these variables as predictors, reflects the degree to which the variance of an estimated regression coefficient is increased due only to the correlations among covariates (Marquardt 1970; Mansfield & Helms 1982).

The function optionally performs stepwise backward removal of variables, to keep the VIF under a given threshold, or to keep the number of variables under a given threshold (e.g., if you need a computation that is implemented for a limited number of variables, such as a Kernel density estimate with `ks::kde`). This procedure is greedy, in the sense that it makes the locally optimal choice at each step without reconsidering previous decisions. It does not evaluate whether removing a different variable at a given step might lead to a better overall subset. Exploring many (or all) possible subsets could yield improved results, but would be computationally much more demanding.

Note also that a high VIF does not necessarily imply that a particular variable should be removed. Consider removing instead other (potencially less important) variables that are correlated with it, causing the high VIF; and see also the `corSelect` function.

Value

The function returns a matrix with one row per variable, the names of the variables as row names, and 3 columns: R-squared, Tolerance, and VIF.

Author(s)

A. Marcia Barbosa

References

Marquardt D.W. (1970) Generalized inverses, ridge regression, biased linear estimation, and non-linear estimation. Technometrics 12: 591-612.

Mansfield E.R. & Helms B.P. (1982) Detecting multicollinearity. The American Statistician 36: 158-160.

Zuur A.F., Ieno E.N. & Elphick C.S. (2010) A protocol for data exploration to avoid common statistical problems. Methods in Ecology and Evolution 1: 3-14.

See Also

`vif` in package **HH**, `vif` in package **usdm**; package **collinear**

Examples

```
data(rotif.env)
names(rotif.env)


# compute multicollinearity among the predictor variables:

multicol(rotif.env[ , 5:17], reorder = FALSE)

multicol(rotif.env[ , 5:17])


# get also a plot of the results:

par(mar = c(11, 4, 2, 1))

multicol(rotif.env[ , 5:17], plot = TRUE,
ylab = "VIF", main = "VIF-selected variables", col = "orange2")


# select variables based on VIF:

multicol(rotif.env[ , 5:17], vif.thresh = 3, plot = TRUE,
ylab = "VIF", main = "VIF-selected variables", col = "orange2")


# you can also compute multicollinearity among variables included in a model:

mod <- step(glm(Abrigh ~ Area + Altitude + AltitudeRange +
HabitatDiversity + HumanPopulation + Latitude + Longitude +
Precipitation + PrecipitationSeasonality + TemperatureAnnualRange
+ Temperature + TemperatureSeasonality + UrbanArea,
data = rotif.env))

multicol(model = mod)


# more examples using R datasets:

multicol(trees)


# you'll get a warning and some NA results if any of the variables
# is not numeric:

multicol(OrchardSprays)
```

```
# so, define the subset of numeric 'vars' to calculate 'multicol' for:

multicol(OrchardSprays[ , 1:3])
```

---

multTSA *Trend Surface Analysis for multiple species*

Description

This function performs trend surface analysis for one or more species at a time. It converts categorical presence-(pseudo)absence (1-0) data into continuous surfaces denoting the spatial trend in species' occurrence patterns.

Usage

```
multTSA(data, sp.cols, coord.cols, id.col = NULL, degree = 3,
step = TRUE, criterion = "AIC", type = "P", Favourability = FALSE,
suffix = "_TS", save.models = FALSE, na.rm = TRUE, verbosity = 2, ...)
```

Arguments

`data`	a matrix or an object inheriting class data frame containing, at least, two columns with spatial coordinates, and one column per species containing their presence (1) and absence (0) data, with localities in rows. Alternatively, a terra SpatRaster map over whose pixels a spatial trend is to be computed.
`sp.cols`	names or index numbers of the columns containing the species presences and absences in 'data', containing only zeros (0) for absences and ones (1) for presences. Or, if 'data' is a SpatRaster, an object inheriting class 'data.frame' with two columns containing the point coordinates of the presences (x and y, or longitude and latitude, in this order!). For SpatRaster inputs, the function is currently only implemented for one species at a time.
`coord.cols`	names or index numbers of the columns containing the spatial coordinates in data (x and y, or longitude and latitude, in this order!). Ignored if 'data' is a SpatRaster and 'sp.cols' is a table of spatial coordinates.
`id.col`	optional (default NULL) name or index number of a column (to be included in the output) containing locality identifiers in 'data'. Ignored if 'data' is a SpatRaster.
`degree`	the degree of the spatial polynomial to use (see Details). The default is 3.
`step`	logical value indicating whether the regression of presence-absence on the spatial polynomial should do a stepwise inclusion of the polynomial terms (using the `step` function with default settings, namely backward AIC selection), rather than forcing all terms into the equation. The default is TRUE.
`criterion`	character value indicating whether the backward stepwise selection of variables (if step = TRUE) should be made according to "AIC" (the default, using the `step` function) or to "significance" (using the `modelTrim` function).

`type`	the type of trend surface to obtain. Can be either "Y" for the raw polynomial equation (i.e. in the scale of the predictors, e.g. if you want to use the spatial trend as a predictor variable in a model); "P" (the default) for the logit-transformed probability (e.g. if you want to use the output as a prediction of presence probability based on spatial trend alone); or "F" for spatial favourability, i.e., prevalence-independent probability (see `Fav`).
`Favourability`	deprecated argument; 'type' should now be used instead, although (at least for the timebeing) this will still be accepted (with Favourability=TRUE internally resulting in type="F") for back-compatibility.
`suffix`	character indicating the suffix to add to the trend surface column (or SpatRaster layer) names in the output data. The default is "_TS".
`save.models`	logical value indicating whether the models obtained from the regressions should be saved and included in the output. The default is FALSE.
`verbosity`	integer value indicating the amount of messages to display; currently meaningful values are 0, 1, and 2 (the default).
`na.rm`	logical value (default TRUE), used if 'data' is a SpatRaster, indicating whether NA input pixels should be NA in the output.
`...`	additional arguments to be passed to `modelTrim` (if step = TRUE and criterion = "significance").

Details

Trend Surface Analysis is a way to model the spatial structure in species' distributions by regressing occurrence data on the spatial coordinates x and y, for a linear trend, or on polynomial terms of these coordinates (x^2, y^2, x*y, etc.), for curvilinear trends (Legendre & Legendre, 1998; Borcard et al., 2011). Second- and third-degree polynomials are often used. 'multTSA' allows specifying the degree of the spatial polynomial to use. By default, it uses a 3rd-degree polynomial and performs stepwise AIC selection of the polynomial terms to include.

Value

If 'data' inherits class 'data.frame', the function returns a data frame containing the identifier column (if provided in 'id.col') and one column per species containing the value predicted by the trend surface analysis. If 'data' is a SpatRaster, the output is a SpatRaster of the values predicted by the trend surface analysis. If save.models = TRUE, the output is a list containing this and a list of the model objects.

Author(s)

A. Marcia Barbosa

References

Borcard D., Gillet F. & Legendre P. (2011) Numerical Ecology with R. Springer, New York.

Legendre P. & Legendre L. (1998) Numerical Ecology. Elsevier, Amsterdam.

See Also

distPres, poly, multGLM

Examples

```
data(rotif.env)

head(rotif.env)

names(rotif.env)

tsa <- multTSA(rotif.env, sp.cols = 18:20,
coord.cols = c("Longitude", "Latitude"), id.col = 1)

head(tsa)
```

---

pairwiseRangemaps   *Pairwise intersection (and union) of range maps*

Description

This function takes a set of rangemaps and returns a matrix containing the areas of their pairwise intersections; optionally, also their individual areas and/our their areas of pairwise unions.

Usage

```
pairwiseRangemaps(rangemaps, projection = NULL, diag = TRUE, unions = TRUE,
verbosity = 2, Ncpu = 1, nchunks = 1, subchunks = NULL,
filename = "rangemap_matrix.csv")
```

Arguments

rangemaps	a character vector of rangemap filenames, including the extension (e.g. ".shp" or ".gpkg"), and the folder paths if not in the woorking directory.
projection	DEPRECATED argument, previously required by function 'PBSmapping::importShapefile', which is now here replaced with 'terra::vect'. Will be ignored with a message if provided. Mind that area computations are more accurate with unprojected input maps (see ?terra::expanse).
diag	logical, whether to fill the diagonal of the resulting matrix with the area of each rangemap. The default is TRUE, and it is also automatically set to TRUE (as it is necessary) if unions = TRUE.
unions	logical, whether to fill the upper triangle of the resulting matrix with the area of union of each pair of rangemaps. The default is TRUE. It is not as computationally intensive as the intersection, as it is calculated not with spatial but with algebraic operations within the matrix (union = area1 + area2 - intersection).
verbosity	integer number indicating the amount of progress messages to display.

`Ncpu` integer indicating the number of CPUs (central processing units) to employ if parallel computing is to be used. The default is 1 CPU, which implies no parallel computing, but you may want to increase this if you have many and/or large rangemaps and your machine has more cores that can be used simultaneously. You can find out the total number of cores in you machine with the `detectCores` function of the **parallel** package; a usually wise option is to use all cores except one (i.e., Ncpu = parallel::detectCores()-1).

`nchunks` either an integer indicating the number of chunks of rows in which to divide the results matrix for calculations, or character "decreasing" to indicate that the matrix should be divided into chunks of decreasing number of rows (as intersections are calculated in the lower triangle, rows further down the matrix have an increasing number of intersections to compute). Note, however, that rangemap size, not rangemap number, is the main determinant of computation time. The default is 1 (no division of the matrix) but, if you have many rangemaps, the process can get clogged. With chunks, each set of rows of the matrix is calculated and saved to disk, and the memory is cleaned before the next chunk begins.

`subchunks` optional integer vector specifying which chunks to actually calculate. This is useful if a previous, time-consuming run of pairwiseRangemaps was interrupted (e.g. by a power outage) and you want to calculate only the remaining chunks.

`filename` optional character vector indicating the name of the file to save the resulting matrix to.

Details

This computation can be intensive and slow, especially if you have many and/or large rangemaps, due to the time needed for pairwise spatial operations between them. You can set nchunks="decreasing" for the matrix to be calculated in parts and the memory cleaned between one part and the next; and, if your computer has more than one core that you can use, you can increase 'Ncpu' to get parallel computing.

Value

This function returns a square matrix containing, in the lower triangle, the area of the pair-wise intersections among the input 'rangemaps'; in the diagonal (if diag = TRUE or union = TRUE), the area of each rangemap; and in the upper triangle (if union = TRUE), the area of the pair-wise unions among the rangemaps.

Note

This function previously used the **PBSmapping** package to import and intersect the rangemaps and to calculate areas. Now it uses the **terra** package instead. Mind that, after the implementation of spherical geometry, area computations are more accurate with unprojected input maps (see ?terra::expanse). Small differences can thus arise between the results of the previous version and the current version (from **fuzzySim** 4.9.4).

Author(s)

A. Marcia Barbosa

References

Barbosa A.M. & Estrada A. (2016) Calcular corotipos sin dividir el territorio en OGUs: una adaptacion de los indices de similitud para su utilizacion directa sobre areas de distribucion. In: Gomez Zotano J., Arias Garcia J., Olmedo Cobo J.A. & Serrano Montes J.L. (eds.), Avances en Biogeografia. Areas de Distribucion: Entre Puentes y Barreras, pp. 157-163. Editorial Universidad de Granada & Tundra Ediciones, Granada (Spain)

See Also

`rangemapSim`

---

partialResp *Partial response plot(s) for probability or favourability*

Description

This function produces partial response plot(s) for probability or favourability, for one to all variables in a 'glm' model object.

Usage

```
partialResp(model, vars = NULL, Fav = FALSE, se.mult = 1.96, plot.points = FALSE,
ylim = c(0, 1), reset.par = TRUE, line.col = "darkblue", ci.col = "#D3DDE9",
point.col = grDevices::rgb(0.7, 0.7, 0.7, alpha = 0.5), ...)
```

Arguments

`model`	a model object of class 'glm' and family 'binomial'.
`vars`	character vector of the name(s) of the variable(s) for which to compute the partial response plot. The default is `NULL`, for all variables in 'model'.
`Fav`	logical value (default `FALSE`) indicating whether to compute the response curve(s) for `Fav`ourability instead of predicted probability.
`se.mult`	numeric value indicating the multiplier for the standard error of the predictions. The default is 1.96, for the 95% confidence interval. If set to 0, no confidence intervals are plotted.
`plot.points`	logical value (default `FALSE`) indicating whether to plot the points of predicted probability (or favourability, if `Fav=TRUE`) against the values of the plotted variable.
`ylim`	either a numeric vector of length 2 indicating the minimum and maximum value for the y-axis, or character value `"auto"` for fitting the axis limits to the existing values in each plot. The default is `c(0, 1)`, for all curves to be directly comparable.
`reset.par`	logical value. If TRUE (the default), plotting `par`ameters are changed by the function and reset in the end. FALSE can be useful if the user wants to set their own parameters (like 'mfrow' or 'mar') and combine this with other plots.

`line.col` colour for the response curve line.

`ci.col` colour for the confidence interval polygon.

`point.col` (if plot.points=TRUE) colour for the points. The default uses transparency (alpha) to improve visibility of overlapping points.

`...` additional arguments that can be passed to `plot`, e.g. 'main', 'cex.axis' or 'cex.lab'.

Details

Each variable is plotted at intervals of 1/100th of its range. Confidence intervals are computed as the value plus/minus the standard error multiplied by 'se.mult' (default 1.96, for the 95% confidence interval). To avoid the confidence intervals exceeding the 0:1 interval that's possible for probability, the standard error is computed on the predictions at the scale of the predictors (i.e., computed with `type="link"`), and then back-transformed with `model$family$linkinv` (see https://fromthebottomoftheheap.net/2018/12/10/confidence-intervals-for-glms – thanks to Gavin Simpson for this post!).

Value

A partial response plot for each variable.

Author(s)

A. Marcia Barbosa

See Also

`plotmo::plotmo`, `predicts::partialResponse`

Examples

```
data(rotif.env)

form <- reformulate(names(rotif.env)[5:17], "Kcochl")

mod <- glm(form, data = rotif.env, family = binomial)

partialResp(mod)

partialResp(mod, Fav = TRUE)

partialResp(mod, Fav = TRUE, plot.points = TRUE)
```

percentTestData *Percent test data*

Description

Based on the work of Schaafsma & van Vark (1979), Huberty (1994) provided a heuristic ("rule of thumb") for determining an adequate proportion of data to set aside for testing species presence/absence models, based on the number of predictor variables that are used (Fielding & Bell 1997). The 'percentTestData' function calculates this proportion as a percentage.

Usage

```
percentTestData(nvar)
```

Arguments

nvar	the number of variables in the model.

Value

A numeric value of the percentage of data to leave out of the model for further model testing.

Author(s)

A. Marcia Barbosa

References

Huberty C.J. (1994) Applied Discriminant Analysis. Wiley, New York, 466 pp.

Schaafsma W. & van Vark G.N. (1979) Classification and discrimination problems with applications. Part IIa. Statistica Neerlandica 33: 91-126

Fielding A.H. & Bell J.F. (1997) A review of methods for the assessment of prediction errors in conservation presence/absence models. Environmental Conservation 24: 38-49

See Also

multGLM

Examples

```
# say you're building a model with 15 variables:

percentTestData(15)

# the result tells you that 21% is an appropriate percentage of data
# to set aside for testing your model, so train it with 79% of the data
```

prevalence *Prevalence*

Description

Prevalence is the proportion of presences of a species in a dataset, which is required (together with presence probability) for computing Favourability.

Usage

```
prevalence(obs, model = NULL, event = 1, na.rm = TRUE)
```

Arguments

obs	a vector or a factor of binary observations (e.g. 1 vs. 0, male vs. female, disease vs. no disease, etc.). This argument is ignored if 'model' is provided.
model	alternatively to 'obs', a binary-response model object of class "glm", "gam", "gbm", "randomForest" or "bart". If this argument is provided, 'obs' will be extracted with 'modEvA::mod2obspred'.
event	the value whose prevalence we want to calculate (e.g. 1, "present", etc.). This argument is ignored if 'model' is provided.
na.rm	logical, whether NA values should be excluded from the calculation. The default is TRUE.

Value

Numeric value of the prevalence of `event` in the `obs` vector.

Author(s)

A. Marcia Barbosa

Examples

```
# calculate prevalence from binary vectors:

(x <- rep(c(0, 1), each = 5))

(y <- c(rep(0, 3), rep(1, 7)))

(z <- c(rep(0, 7), rep(1, 3)))

prevalence(x)

prevalence(y)

prevalence(z)
```

```
(w <- c(rep("yes", 3), rep("nope", 7)))

prevalence(w, event = "yes")


# calculate prevalence from a model object:

data(rotif.env)

mod <- glm(Abrigh ~ HabitatDiversity + HumanPopulation, family = binomial, data = rotif.env)

prevalence(model = mod)

# same as:
prevalence(obs = rotif.env$Abrigh)
```

---

rangemapSim *Pairwise similarity between rangemaps*

Description

Calculate pairwise similarity among rangemaps from a matrix of their areas of intersection and union

Usage

```
rangemapSim(rangemap.matrix, total.area,
method = c("Jaccard", "Sorensen", "Simpson", "Baroni"),
diag = FALSE, upper = FALSE, verbosity = 2)
```

Arguments

rangemap.matrix	a matrix like the one produced by function pairwiseRangemaps, containing the areas of pairwise intersection among rangemaps in the lower triangle, individual rangemap areas in the diagonal, and pairwise union areas in the upper diagonal.
total.area	numeric value indicating the total size of the study area, in the same units as the areas in the rangemap.matrix. Used only if 'method' uses shared absences (as is the case of "Baroni")
method	character value indicating the similarity index to use. Currently implemented indices are "Jaccard", "Sorensen", "Simpson" and "Baroni". The default is the first one.
diag	logical value indicating if the diagonal of the resulting matrix should be filled
upper	logical value indicating if the upper triangle of the resulting matrix should be filled (symmetrical to the lower triangle)
verbosity	integer number indicating the amount of messages to display.

Details

Distributional relationships among species are commonly determined based on pair-wise (dis)similarities in species' occurrence patterns. Some of the most commonly employed similarity indices are those of Jaccard (1901), Sorensen (1948), Simpson (1960) and Baroni-Urbani & Buser (1976), which are here implemented for comparing rangemaps based on their areas of intersection and union (Barbosa & Estrada, in press).

Value

This function returns a square matrix of pairwise similarities between the rangemaps in 'rangemap.matrix', calculated with the (first) similarity index specified in 'method'.

Author(s)

A. Marcia Barbosa

References

Barbosa A.M. & Estrada A. (in press) Calcular corotipos sin dividir el territorio en OGUs: una adaptacion de los indices de similitud para su utilizacion directa sobre areas de distribucion. In: Areas de distribucion: entre puentes y barreras. Universidad de Granada, Spain.

Baroni-Urbani C. & Buser M.W. (1976) Similarity of Binary Data. Systematic Zoology, 25: 251-259

Jaccard P. (1901) Etude comparative de la distribution florale dans une portion des Alpes et des Jura. Memoires de la Societe Vaudoise des Sciences Naturelles, 37: 547-579

Simpson G.G. (1960) Notes on the measurement of faunal resemblance. Amer. J. Sci. 258A, 300-311

Sorensen T. (1948) A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons. Kongelige Danske Videnskabernes Selskab, 5(4): 1-34

See Also

`pairwiseRangemaps`; `simFromSetOps`; `simMat`

`rarity` *(Fuzzy) rarity*

Description

This function computes the index of species rarity of Real et al. (2006), using either crisp (presence/absence, i.e. ones and zeros) or fuzzy values (e.g. `Fav`ourability, between zero and one), for a single species or for several species across a study area. Rarity is like a (potential) richness index in which rarer species have higher weight.

Usage

```
rarity(data, sp.cols = 1:ncol(data), na.rm = TRUE)
```

Arguments

`data`	a numeric vector, matrix or data frame containing the presence/absence or the `Fav`ourability (fuzzy presence) values for the target species.
`sp.cols`	names or index numbers of the columns of 'data' that contain the species values for which to compute rarity. The default is to use all columns.
`na.rm`	logical value indicating whether NA values should be removed before the computation.

Details

If the input data include only one species (i.e. a numeric vector or a one-column table, with one value for each locality), rarity is 1 divided by the sum of its values. If the input includes more than one species or column, rarity is the sum of the product of each (fuzzy) presence value by the rarity of the corresponding species, so that rarer species have higher weight in the resulting sum (Real et al. 2006). See also Estrada et al. (2011) for a more complex version of fuzzy rarity.

Value

If 'data' is a vector or a one-column table, or if 'sp.cols' is of length 1, this function returns a single value of rarity for the underlying species, which is simply 1 divided by the sum of its values. If 'data' and 'sp.cols' refer to more than 1 column, the function returns the total combined rarity value of all corresponding species for each row in 'data' (see Examples).

Author(s)

A. Marcia Barbosa

References

Real R., Estrada A., Barbosa A.M. & Vargas J.M. (2006) Aplicacion de la logica difusa al concepto de rareza para su uso en Gap Analysis: el caso de los mamiferos terrestres en Andalucia. Serie Geografica 13: 99-116

Estrada A., Real R. & Vargas J.M. (2011) Assessing coincidence between priority conservation areas for vertebrate groups in a Mediterranean hotspot. Biological Conservation, 144: 1120-1129

See Also

`vulnerability`

Examples

```
data(rotif.env)

head(rotif.env)
```

```
rarity(rotif.env[ , 18])

rarity(rotif.env, sp.cols = "Abrigh")

rarity(rotif.env, sp.cols = 18:47)
# yields one value of combined rarity for each row in 'data'


# fuzzy rarity (from favourability values):

pred <- multGLM(rotif.env, sp.cols = 18:20, var.cols = 5:17)$predictions

head(pred)

rarity(pred, sp.cols = "Abrigh_F")

rarity(pred, sp.cols = c("Abrigh_F", "Afissa_F", "Apriod_F"))
# yields one value of combined rarity for each row in 'data'
```

`rotif.env` *Rotifers and environmental variables on TDWG level 4 regions of the world*

Description

These data were extracted from a database of monogonont rotifer species presence records on the geographical units used by the Biodiversity Information Standards (formerly Taxonomic Database Working Group, TDWG: https://www.tdwg.org) and a few environmental (including human and spatial) variables on the same spatial units. The original data were compiled and published by Fontaneto et al. (2012) in long (narrow, stacked) format. Here they are presented in wide or unstacked format (presence-absence table, obtained with the `splist2presabs` function), reduced to the species recorded in at least 100 (roughly one third) different TDWG level 4 units, and with abbreviations of the species' names (obtained with the `spCodes` function). Mind that this is not a complete picture of these species' distributions, due to insufficient sampling in many regions.

Usage

```
data(rotif.env)
```

Format

A data frame with 291 observations on the following 47 variables.

`TDWG4` a factor with 291 levels indicating the abbreviation code of each TDWG4 region

`LEVEL_NAME` a factor with 291 levels indicating the name of each TDWG4 region

`REGION_NAME` a factor with 47 levels indicating the name of the main geographical region to which each TDWG4 level belongs

`CONTINENT` a factor with 9 levels indicating the continent to which each TDWG4 level belongs

`Area` a numeric vector

`Altitude` a numeric vector

`AltitudeRange` a numeric vector

`HabitatDiversity` a numeric vector

`HumanPopulation` a numeric vector

`Latitude` a numeric vector

`Longitude` a numeric vector

`Precipitation` a numeric vector

`PrecipitationSeasonality` a numeric vector

`TemperatureAnnualRange` a numeric vector

`Temperature` a numeric vector

`TemperatureSeasonality` a numeric vector

`UrbanArea` a numeric vector

`Abrigh` a numeric vector

`Afissa` a numeric vector

`Apriod` a numeric vector

`Bangul` a numeric vector

`Bcalyc` a numeric vector

`Bplica` a numeric vector

`Bquadr` a numeric vector

`Burceo` a numeric vector

`Cgibba` a numeric vector

`Edilat` a numeric vector

`Flongi` a numeric vector

`Kcochl` a numeric vector

`Kquadr` a numeric vector

`Ktropi` a numeric vector

`Lbulla` a numeric vector

`Lclost` a numeric vector

`Lhamat` a numeric vector

`Lluna` a numeric vector

`Llunar` a numeric vector

`Lovali` a numeric vector

`Lpatel` a numeric vector

`Lquadr` a numeric vector

`Mventr` a numeric vector

`Ppatul` a numeric vector

`Pquadr` a numeric vector

`Pvulga` a numeric vector

`Specti` a numeric vector

`Tpatin` a numeric vector

`Tsimil` a numeric vector

`Ttetra` a numeric vector

Source

Fontaneto D., Barbosa A.M., Segers H. & Pautasso M. (2012) The 'rotiferologist' effect and other global correlates of species richness in monogonont rotifers. Ecography, 35: 174-182.

Examples

```
data(rotif.env)

head(rotif.env)
```

---

rotifers *Rotifer species on TDWG level 4 regions of the world*

Description

These data were extracted from a database of monogonont rotifer species records on the geographical units used by the Biodiversity Information Standards (formerly Taxonomic Database Working Group, TDWG: https://www.tdwg.org). The original data were compiled and published by Fontaneto et al. (2012) for all TDWG levels. Here they are reduced to the TDWG - level 4 units and to the species recorded in at least 100 (roughly one third) of these units. Mind that this is not a complete picture of these species' distributions, due to insufficient sampling in many regions.

Usage

```
data("rotifers")
```

Format

A data frame with 3865 observations on the following 2 variables.

`TDWG4` a factor with 274 levels corresponding to the code names of the TDWG level 4 regions in which the records were taken

`species` a factor with 30 levels corresponding to the names of the (sub)species recorded in at least 100 different TDWG level 4 regions

Source

Fontaneto D., Barbosa A.M., Segers H. & Pautasso M. (2012) The 'rotiferologist' effect and other global correlates of species richness in monogonont rotifers. Ecography, 35: 174-182.

Examples

```
data(rotifers)

head(rotifers, 10)
```

`selectAbsences` *Select (spatially biased) absence rows.*

Description

This function takes a matrix or (optionally spatial) data frame containing species presence (1) and (pseudo)absence (0) data (e.g., the output of [`gridRecords`), and it selects among the absence rows to stay within a given number or ratio of absences, and/or within and/or beyond a given distance to the presences. Optionally, to incorporate survey bias, absences can be selected with higher probability towards the vicinity of presences; or with probability weights according to a user-provided bias raster.

Usage

```
selectAbsences(data, sp.cols, coord.cols = NULL, CRS = NULL, min.dist = NULL,
max.dist = NULL, n = NULL, mult.p = NULL, bias = FALSE, bunch = FALSE,
dist.mat = NULL, seed = NULL, plot = !is.null(coord.cols), df = TRUE,
verbosity = 2)
```

Arguments

`data`	a 'data.frame' or an object that can be coerced to such (e.g. a 'matrix', 'tibble', 'SpatVector' or 'sf' object) containing a column with the species' presence (1) and absence (0) records, with localities in rows; and (if distance or spatial bias are required) two columns with the spatial coordinates, x and y.
`sp.cols`	names or index numbers of the columns containing the species presences (1) and absences (0) in 'data'.
`coord.cols`	names or index numbers of the columns containing the spatial coordinates in 'data' (x and y, or longitude and latitude, in this order!). Needed if distance or spatial bias are required.
`CRS`	coordinate reference system of the 'coord.cols' in 'data', in one of the following formats: WKT/WKT2, <authority>:<code>, or PROJ-string notation (see `terra::crs()`). Ignored if 'dist.mat' is provided. Otherwise, if 'CRS' is provided and the 'terra' package is installed, distances are computed with `terra::distance()`, thus accounting for the curvature of the Earth.
`min.dist`	(optional) numeric value specifying the minimum distance (in the same units as 'coord.cols') at which selected absences should be from the presences.
`max.dist`	(optional) numeric value specifying the maximum distance (in the same units as 'coord.cols') at which selected absences should be from the presences.

n	(optional) integer value specifying the number of absence rows to select. Can also be specified as a ratio – see 'mult.p' below.
mult.p	(optional) numeric value specifying how many times the number of presences to use as 'n' (e.g. 10 times as many absences as presences). Ignored if 'n' is not NULL.
bias	either a logical value TRUE to make the selection of absences biased towards the vicinity of presences (which requires specifying 'coord.cols'; may take time and memory for large datasets if 'dist.mat' is not provided); or a SpatRaster layer (quantifying e.g. survey effort, accessibility or human presence; see Details and biasLayer) with higher pixel values where the selection of absences should be more likely. Note that this layer should have (approximately) the same spatial resolution as 'data'. The default is FALSE, for no bias.
dist.mat	optional argument to pass to distPres. If not provided, a distance matrix will be computed with stats::dist()) or with terra::distance()).
bunch	[PENDING IMPLEMENTATION] logical value specifying if the selected absences should concentrate around presences in proportion to their local density, as in Vollering et al. (2019). The default is FALSE.
seed	(optional) integer value to pass to set.seed specifying the random seed to use for sampling among the absences.
plot	logical value specifying whether to plot the result. The default is TRUE if 'coord.cols' are provided.
df	logical value specifying whether to return a dataframe with the input 'data' after removal of the non-selected absences. The default is TRUE. If set to FALSE, the output is a logical vector specifying if each row of 'data' was selected or not.
verbosity	numeric value indicating the amount of messages to display. Choose 0 for no messages.

Details

Species occurrence data typically incorporate two probability distributions: the actual probability of the species being present, and the probability of it being recorded if it was present (Merow et al. 2013). Thus, any covariation between recording probability and the predictor variables can bias the predictions of species distribution models (Yackulic et al. 2013).

Methods to correct for this bias include the selection of (pseudo)absences preferably towards the vicinity of presence records, in order to reproduce the survey bias. This function implements this strategy in several (alternative or complementary) ways: 1) selecting absences within and/or beyond a given distance from presences; 2) biasing the random selection of absences, making it more likely towards the vicinity of presences (providing the 'prob' argument in sample with the result of distPres); or [PENDING IMPLEMENTATION!] 3) bunching up the absences preferably around the areas with higher densities of presences (Vollering et al. 2019).

More recent versions allow using instead a raster map of weights (bias layer), with higher pixel values where absence selection should be proportionaly more likely, and zero or NA where pseudoabsence points should not be placed. This layer should normally reflect (a proxy for) likely survey effort, such as proximity to roads or populated areas, human footprint (see e.g. geodata::footprint()) or travel time (e.g. geodata::travel_time()). You can also create your own e.g. with biasLayer. Users should provide the bias layer at aproximately the same spatial resolution as the 'coord.cols' in the input 'data'.

Value

This function returns the 'data' input after removal of the non-selected absences, or (if df=FALSE) a logical vector specifying if each row of 'data' was selected (TRUE) or not (FALSE). If plot=TRUE and provided 'coord.cols', it also plots the presences (blue "plus" signs), the selected absences (red "minus" signs) and the excluded absences (orange dots).

Author(s)

A. Marcia Barbosa

References

Vollering J., Halvorsen R., Auestad I. & Rydgren K. (2019) Bunching up the background betters bias in species distribution models. Ecography, 42: 1717-1727

See Also

gridRecords, sample, biasLayer

Examples

```
data(rotif.env)

head(rotif.env)

names(rotif.env)

table(rotif.env$Burceo)

# select among the absences using different criteria:

abselect <- selectAbsences(data = rotif.env, sp.cols = "Burceo",
coord.cols = c("Longitude", "Latitude"), n = 150, seed = 123)

abselect <- selectAbsences(data = rotif.env, sp.cols = "Burceo",
coord.cols = c("Longitude", "Latitude"), mult.p = 1.5, seed = 123)

## Not run:
abselect <- selectAbsences(data = rotif.env, sp.cols = "Burceo",
coord.cols = c("Longitude", "Latitude"), max.dist = 18)

abselect <- selectAbsences(data = rotif.env, sp.cols = "Burceo",
coord.cols = c("Longitude", "Latitude"), max.dist = 18, min.dist = 5,
n = sum(rotif.env$Burceo), bias = TRUE)


# you can run the next example if you have the 'geodata' package installed:

human_footprint <- geodata::footprint(year = 2009, path = tempfile())

# see also the biasLayer() function for creating your own bias layer
```

```
# or see below for creating a bias layer based on target-group background


abselect <- fuzzySim::selectAbsences(data = rotif.env, sp.cols = "Burceo",
coord.cols = c("Longitude", "Latitude"), n = sum(rotif.env$Burceo),
bias = human_footprint)

terra::plot(human_footprint, alpha = 0.3, add = TRUE)


# you can run the next example if you have 'terra' and 'spatstat' installed:

# coordinates of records (surveyed sites) for the whole taxonomic group:
recs <- rotif.env[rowSums(rotif.env[ , 18:47] > 0), c("Longitude", "Latitude")]

rng <- terra::ext()

win <- spatstat.geom::owin(xrange = rng[1:2], yrange = rng[3:4])

ppp <- spatstat.geom::ppp(x = recs$Longitude, y = recs$Latitude, window = win)

kernel_density <- terra::rast(spatstat.explore::density.ppp(ppp))

terra::plot(kernel_density)

points(recs, col = "red", cex = 0.2)

abselect <- fuzzySim::selectAbsences(data = rotif.env, sp.cols = "Burceo",
coord.cols = c("Longitude", "Latitude"), n = sum(rotif.env$Burceo),
bias = kernel_density)

terra::plot(kernel_density, alpha = 0.3, add = TRUE)

## End(Not run)
```

---

sharedFav *Shared favourability for two competing species*

Description

This function implements the graphical analyses of Acevedo et al. (2010, 2012) on biogeographical interactions. It takes two vectors of favourability values at different localities for, respectively, a stronger and a weaker competing species (or two equally strong competitors), and plots their favourableness or shared favourability to assess potential competitive interactions.

Usage

```
sharedFav(strong_F, weak_F, conf = 0.95, bin_interval = "0.1", ...)
```

Arguments

`strong_F`	a numeric vector of favourability values (obtained, e.g., with functions `Fav` or `multGLM`) for the stronger species.
`weak_F`	a numeric vector of favourability values for the weaker species. Must be of the same length and in the same order as strong_F.
`conf`	confidence level for the confidence intervals in the plot. The default is 0.95. Set it to NA for no confidence intervals (see "Note" below).
`bin_interval`	character value specifying the method for grouping the favourability values into bins for plotting and comparing mean favourability for each species. Currently implemented options are "0.1" (the default, dividing the values at 0.1 intervals as per Acevedo et al. 2010, 2012) and "quantiles" (as the former method may produce an error if there are bins too small to allow computing confidence intervals). See "Note" below.
`...`	some additional arguments can be passed to `barplot`, such as "main" (for the plot title) or "las" (for the orientation of the axis labels).

Details

This function implements the biogeographic analyses of Acevedo et al. (2010, 2012), assessing the trends of environmental favourability across a range of favourability intersection values between two interacting species. It first calculates the fuzzy intersection (minimum value) between the two species' favourability values at each locality (i.e., favourability for the occurrence of both species simultaneously); it groups these values into 10 bins; and calculates the mean favourability (and its confidence interval) for each of the two species within each interval.

According to the notion of "favorableness" by Richerson & Lum (1980), competing species may or may not be able to coexist depending on their relative environmental fitnesses; competition between species increases and competitive exclusion decreases as their favourability intersection increases (Acevedo et al. 2010, 2012). The shaded area in the shared favourability plot, where at least one of the species is at intermediate favourability, is the area where competitive interactions may limit species occurrence. Outside this shaded area, where favourability is either very low for at least one of the species (left) or very high for both species (right side of the plot), competition is not limiting (see also `bioThreat` for details).

Value

This function returns the numeric value of the fuzzy overlap index (FOvI; Dubois & Prade 1980, Acevedo et al. 2010, 2012), a data frame with the bin values and the shared favourability plot, with circles and a continuous line representing favourability for the stronger species, and squares and a dashed line representing favourability for the weaker species. The height of the bars at the bottom represents the proportional sample size of each bin.

Note

This function may generate an error if one or more bins don't have enough values for the confidence interval to be computed. If this occurs, you can try a different 'bin_interval' (e.g. "quantiles") or set the 'conf' argument to NA (in which case confidence intervals will not be computed). Either will affect only the plot, not the overall fuzzy overlap value.

Author(s)

A. Marcia Barbosa

References

Acevedo P., Ward A.I., Real R. & Smith G.C. (2010) Assessing biogeographical relationships of ecologically related species using favourability functions: a case study on British deer. Diversity and Distributions, 16: 515-528

Acevedo P., Jimenez-Valverde A., Melo-Ferreira J., Real R. & Alves, P.C. (2012) Parapatric species and the implications for climate change studies: a case study on hares in Europe. Global Change Biology, 18: 1509-1519

Dubois D. & Prade H. (1980) Fuzzy sets and systems: theory and applications. Academic Press, New York

Richerson P.J. & Lum K. (1980) Patterns of plant species and diversity in California: relation to weather and topography. American Naturalist, 116: 504-536

See Also

bioThreat, Fav

Examples

```
# get favourability model predictions for two species:
data(rotif.env)
mods <- multGLM(rotif.env, sp.cols = 19:20, var.cols = 5:17)
head(mods$predictions)
favs <- mods$predictions[ , 3:4]

# get shared favourability:
sharedFav(strong_F = favs[,1], weak_F = favs[,2], main = "Shared favourability")

sharedFav(strong_F = favs[,1], weak_F = favs[,2], bin_interval = "quantiles",
main = "Shared favourability", las = 2)
```

---

simFromSetOps — *Calculate similarity from set operations*

Description

This function calculates pair-wise similarity based on the results of set operations (intersection, union) among the subjects.

Usage

```
simFromSetOps(size1, size2, intersection, union, total.size = NULL,
method = c("Jaccard", "Sorensen", "Simpson", "Baroni"),
verbosity = 1)
```

Arguments

size1	size of subject 1 (e.g., area of the distribution range of a species, or its number of presences within a grid). Not needed if method = "Jaccard".
size2	the same for subject 2.
intersection	size of the intersection among subjects 1 and 2 (area of the intersection among their distribution ranges, or number of grid cells in which they co-occur).
union	size of the union of subjects 1 and 2.
total.size	total size of the study area. Needed only when calculating a similarity index that takes shared absences into account (i.e., method = "Baroni").
method	the similarity index to use. Currently implemented options are "Jaccard", "Sorensen", "Simpson" or "Baroni".
verbosity	integer indicating whether to display messages.

Details

Similarities among ecological communities, beta diversity patterns, biotic regions, and distributional relationships among species are commonly determined based on pair-wise (dis)similarities in species' occurrence patterns. This function implements some of the most commonly employed similarity indices, namely those of Jaccard (1901), Sorensen (1948), Simpson (1960) and Baroni-Urbani & Buser (1976), based on the amount of occupied and overlap area between two species.

Value

The numeric value of similarity among subjects 1 and 2.

Author(s)

A. Marcia Barbosa

References

Baroni-Urbani C. & Buser M.W. (1976) Similarity of Binary Data. Systematic Zoology, 25: 251-259

Jaccard P. (1901) Etude comparative de la distribution florale dans une portion des Alpes et des Jura. Memoires de la Societe Vaudoise des Sciences Naturelles, 37: 547-579

Simpson, G.G. (1960) Notes on the measurement of faunal resemblance. Amer. J. Sci. 258A, 300-311

Sorensen T. (1948) A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons. Kongelige Danske Videnskabernes Selskab, 5(4): 1-34

See Also

fuzSim, simMat

Examples

```
# take two species which occur in 22 and 35 area units, respectively
# and which overlap in 8 of those units:

sp1 <- 22
sp2 <- 35
int <- 8
uni <- sp1 + sp2 - int


# calculate similarity between their distributions based on
# different indices:

simFromSetOps(intersection = int, union = uni, method = "Jaccard")

simFromSetOps(sp1, sp2, int, uni, method = "Sorensen")

simFromSetOps(sp1, sp2, int, uni, method = "Simpson")


# if you want Baroni-Urbani & Buser's index
# you need to provide also the total size of your study area:

simFromSetOps(sp1, sp2, int, uni, total = 100, method = "Baroni")
```

---

simMat *Pair-wise (fuzzy) similarity matrix*

Description

simMat takes multiple species occurrence data or regional species composition, either categorical (0 or 1) or fuzzy (between 0 and 1), and uses the fuzSim function to compute a square matrix of pair-wise similarities between them, using a fuzzy logic version (Barbosa, 2015) of the specified similarity index.

Usage

```
simMat(data, method, diag = TRUE, upper = TRUE, verbosity = 2, plot = FALSE, ...)
```

Arguments

data a matrix, data frame, or multilayer SpatRaster containing (optionally fuzzy) species presence-absence data (in wide format, i.e. one column or layer per species), with 1 meaning presence, 0 meaning absence, and values in between for fuzzy presence (or the degree to which each locality belongs to the set of species presences; see Zadeh, 1965). Fuzzy presence-absence can be obtained, for example, with multGLM, distPres or multTSA. These data can also be transposed for comparing regional species compositions.

method	the similarity index whose fuzzy version to use. See fuzSim for available options.
diag	logical value indicating whether the diagonal of the matrix should be filled (with ones). Defaults to TRUE.
upper	logical value indicating whether the upper triangle of the matrix (symmetric to the lower triangle) should be filled. Defaults to TRUE.
verbosity	integer value indicating the amount of messages to display; currently meaningful values are 0, 1, and 2 (the default).
plot	logical argument indicating whether to also plot the matrix as an image. The default is FALSE (for back-compatibility).
...	some additional arguments can be passed to image (and through to plot) if plot=TRUE, such as 'col', 'main', 'font.main' or 'cex.main' (not 'axes', 'xlab' or 'ylab', which are already defined by simMat).

Details

The fuzzy versions of species occurrence data and of binary similarity indices introduce tolerance for small spatial differences in species' occurrence localities, allow for uncertainty about species occurrence, and may compensate for under-sampling and geo-referencing errors (Barbosa, 2015).

Value

This function returns a square matrix of pair-wise similarities among the species distributions (columns) in `data`. Similarity is calculated with the fuzzy version of the index specified in `method`, which yields traditional binary similarity if the data are binary (0 or 1), or fuzzy similarity if the data are fuzzy (between 0 and 1) (Barbosa, 2015).

Author(s)

A. Marcia Barbosa

References

Barbosa A.M. (2015) fuzzySim: applying fuzzy logic to binary similarity indices in ecology. Methods in Ecology and Evolution, 6: 853-858.

See Also

fuzSim

Examples

```
# load and look at the rotif.env presence-absence data:

data(rotif.env)

head(rotif.env)

names(rotif.env)
```

```
# build a matrix of similarity among these binary data
# using e.g. Jaccard's index:

bin.sim.mat <- simMat(rotif.env[ , 18:47], method = "Jaccard")

head(bin.sim.mat)


## Not run:
# calculate a fuzzy version of the presence-absence data
# based on inverse distance to presences:

rotifers.invd <- distPres(rotif.env, sp.cols = 18:47,
coord.cols = c("Longitude", "Latitude"), id.col = 1, suffix = ".d",
p = 1, inv = TRUE)

head(rotifers.invd)


# build a matrix of fuzzy similarity among these fuzzy
# distribution data, using the fuzzy version of Jaccard's index:

fuz.sim.mat <- simMat(rotifers.invd[ , -1], method = "Jaccard")

head(fuz.sim.mat)


# plot the similarity matrices as colours:

image(x = 1:ncol(bin.sim.mat), y = 1:nrow(bin.sim.mat),
z = bin.sim.mat, col = rev(heat.colors(256)), xlab = "", ylab = "",
axes = FALSE, main = "Binary similarity")
axis(side = 1, at = 1:ncol(bin.sim.mat), tick = FALSE,
labels = colnames(bin.sim.mat), las = 2)
axis(side = 2, at = 1:nrow(bin.sim.mat), tick = FALSE,
labels = rownames(bin.sim.mat), las = 2)

image(x = 1:ncol(fuz.sim.mat), y = 1:nrow(fuz.sim.mat),
z = fuz.sim.mat, col = rev(heat.colors(256)), xlab = "", ylab = "",
axes = FALSE, main = "Fuzzy similarity")
axis(side = 1, at = 1:ncol(fuz.sim.mat), tick = FALSE,
labels = colnames(fuz.sim.mat), las = 2, cex = 0.5)
axis(side = 2, at = 1:nrow(fuz.sim.mat), tick = FALSE,
labels = rownames(fuz.sim.mat), las = 2)


# plot a UPGMA dendrogram from each similarity matrix:

plot(hclust(as.dist(1 - bin.sim.mat), method = "average"),
main = "Binary cluster dendrogram")
```

```
plot(hclust(as.dist(1 - fuz.sim.mat), method = "average"),
main = "Fuzzy cluster dendrogram")

## End(Not run)

# you can get fuzzy chorotypes from these similarity matrices
# (or fuzzy biotic regions if you transpose 'data'),
# so that localities are in columns and species in rows)
# using the RMACOQUI package (Olivero et al. 2011)
```

---

spCodes *Obtain unique abbreviations of species names*

Description

This function takes a vector of species names and converts them to abbreviated species codes containing the specified numbers of characters from the genus, the specific and optionally also the subspecific name. Separators can be specified by the user. The function checks that the resulting codes are unique.

Usage

```
spCodes(species, nchar.gen = 3, nchar.sp = 3, nchar.ssp = 0,
sep.species = " ", sep.spcode = "", verbosity = 2)
```

Arguments

`species`	a character vector containig the species names to be abbreviated.
`nchar.gen`	the number of characters from the genus name to be included in the resulting species code.
`nchar.sp`	the number of characters from the specific name to be included in the resulting species code.
`nchar.ssp`	optionally, the number of characters from the subspecific name to be included in the resulting species code. Set it to 0 if you have subspecific names in 'species' but do not want them included in the resulting species codes.
`sep.species`	the character that separates genus, specific and subspecific names in 'species'. The default is a white space.
`sep.spcode`	the character you want separating genus and species abbreviations in the resulting species codes. The default is an empty character (no separator).
`verbosity`	integer value indicating the amount of messages to display. Defaults to 2, for showing all messages.

Value

This function returns a character vector containing the species codes resulting from the abbreviation. If the numbers of characters specified do not make for unique codes, an error message is displayed showing which 'species' names caused it, so that you can try again with different 'nchar.gen', 'nchar.sp' and/or 'nchar.ssp'.

Author(s)

A. Marcia Barbosa

See Also

substr, strsplit

Examples

```
data(rotifers)

head(rotifers)

## add a column to 'rotifers' with shorter versions of the species names:

## Not run:
rotifers$spcode <- spCodes(rotifers$species, sep.species = "_",
nchar.gen = 1, nchar.sp = 4, nchar.ssp = 0, sep.spcode = ".")

# this produces an error due to resulting species codes not being unique

## End(Not run)

rotifers$spcode <- spCodes(rotifers$species, sep.species = "_",
nchar.gen = 1, nchar.sp = 5, nchar.ssp = 0, sep.spcode = ".")

# with a larger number of characters from the specific name,
# resulting codes are now unique

## check out the result:
head(rotifers)
```

---

splist2presabs *Convert a species list to a presence-absence table*

Description

This function takes a locality+species dataset in long (stacked) format, i.e., a matrix or data frame containing localities in one column and their recorded species in another column, and converts them to a presence-absence table (wide format) suitable for mapping and for computing distributional similarities (see e.g. simMat). Try out the Examples below for an illustration).

Usage

```
splist2presabs(data, sites.col, sp.col, keep.n = FALSE)
```

Arguments

`data`	a matrix or data frame with localities in one column and species in another column. Type or paste 'data(rotifers); head(rotifers)' (without the quote marks) in the R console for an example.
`sites.col`	the name or index number of the column containing the localities in 'data'.
`sp.col`	the name or index number of the column containing the species names or codes in 'data'.
`keep.n`	logical value indicating whether to get in the resulting table the number of times each species appears in each locality; if FALSE (the default), only presence (1) or absence (0) is recorded.

Value

A data frame containing the localities in the first column and then one column per species indicating their presence or absence (or their number of records if keep.n = TRUE). Type 'data(rotif.env); head(rotif.env[,18:47])' (without the quote marks) in the R console for an example.

Author(s)

A. Marcia Barbosa

See Also

`table`

Examples

```
data(rotifers)

head(rotifers)

rotifers.presabs <- splist2presabs(rotifers, sites.col = "TDWG4",
sp.col = "species", keep.n = FALSE)

head(rotifers.presabs)
```

`stepByStep` *Compare model predictions along a stepwise variable selection process*

Description

This function builds (or takes) a generalized linear model with stepwise inclusion of variables, using either AIC, BIC or p.value as the selection criterion; and it returns the values predicted at each step (i.e., as each variable is added or dropped), as well as their correlation with the final model predictions.

Usage

```
stepByStep(data, sp.col, var.cols, family = binomial(link = "logit"),
Favourability = FALSE, trace = 0, direction = "both", select = "AIC",
k = 2, test.in = "Rao", test.out = "LRT", p.in = 0.05, p.out = 0.1,
cor.method = "pearson")
```

Arguments

`data` a data frame (or another object that can be coerced with "as.data.frame", e.g. a matrix, a tibble, a SpatVector) containing the response and predictor variables to model. Alternatively, a model object of class 'glm', from which the names, values and order of the variables will be taken – arguments 'sp.col', 'var.cols', 'family', 'trace', 'direction', 'select', 'k', 'test.in', 'test.out', 'p.in' and 'p.out' will then be ignored.

`sp.col` (if 'data' is not a model object) the name or index number of the column of 'data' that contains the response variable.

`var.cols` (if 'data' is not a model object) the names or index numbers of the columns of 'data' that contain the predictor variables.

`family` (if 'data' is not a model object) argument to pass to `glm` indicating the family (and error distribution) to use in modelling. The default is binomial distribution with logit link (for binary response variables).

`Favourability` logical, whether to apply the `Fav`ourability function to remove the effect of prevalence from predicted probability (Real et al. 2006). Applicable only to binomial GLMs. Defaults to FALSE.

`trace` (if 'data' is not a model object) argument to pass to `step` (if select="AIC" or "BIC") or to `stepwise` (if select="p.value"). If positive, information is printed during the stepwise procedure. Larger values may give more detailed information. The default is 0 (silent).

`direction` (if 'data' is not a model object) argument to pass to `step` (if select="AIC" or "BIC") or to `stepwise` (if select="p.value"). Can be "forward" or "both". The default is the latter, to match related functions like `step`, `stepwise` and `multGLM`. (Note that older versions of this function had "forward" as the default.)

`select` (if 'data' is not a model object) character string specifying the criterion for stepwise selection of variables if step=TRUE. Options are the default "AIC" (Akaike's Information Criterion; Akaike, 1973); BIC (Bayesian Information Criterion, also known as Schwarz criterion, SBC or SBIC; Schwarz, 1978); or "p.value" (Murtaugh, 2014). The first two options imply using `step` as the variable selection function, while the last option calls the `stepwise` function.

`k` (if 'data' is not a model object and select="AIC") argument passed to the `step` function indicating the multiple of the number of degrees of freedom used for the penalty. The default is 2, which yields the original AIC. You can use larger values for a more stringent selection– e.g., for a critical p-value of 0.05, use k = qchisq(0.05, 1, lower.tail = F). If select="BIC", k is accordingly changed to log(n), being 'n' the number of complete rows of the response + variables dataframe (after removing missing values).

`test.in`	(if 'data' is not a model object and select="p.value") argument passed to `add1` specifying the statistical test whose 'p.in' a variable must pass to enter the model. Can be "Rao" (the default), "LRT", "Chisq" or "F".
`test.out`	(if 'data' is not a model object and select="p.value") argument passed to `drop1` specifying the statistical test whose 'p.out' a variable must exceed to be expelled from the model (if it does not simultaneously pass the 'test.in' when direction="both"). Can be "LRT" (the default), "Rao", "Chisq" or "F".
`p.in`	(if 'data' is not a model object and select="p.value") threshold p-value for a variable to enter the model. Defaults to 0.05.
`p.out`	(if 'data' is not a model object and select="p.value") threshold p-value for a variable to leave the model. Defaults to 0.1.
`cor.method`	character string to pass to `cor` indicating which coefficient to use for correlating predictions at each step with those of the final model. Can be "pearson" (the default), "kendall", or "spearman".

Details

Stepwise variable selection often includes more variables than would a model selected after examining all possible combinations of the variables (e.g. with package **MuMIn** or **glmulti**). The 'stepByStep' function can be useful to assess if a stepwise model with just the first few variables could already provide predictions very close to the final ones (see e.g. Fig. 3 in Munoz et al., 2005). It can also be useful to see which variables determine the more general trends in the model predictions, and which variables just provide additional (local) nuances.

Value

This function returns a list of the following components:

`predictions`	a data frame with the model's fitted values at each step of the variable selection.
`correlations`	a numeric vector of the correlation between the predictions at each step and those of the final model.
`variables`	a character vector of the variables in the final model, named with the step at which each was included.
`model`	the resulting model object.

Author(s)

A. Marcia Barbosa, with contribution by Alba Estrada

References

Akaike, H. (1973) Information theory and an extension of the maximum likelihood principle. In: Petrov B.N. & Csaki F., 2nd International Symposium on Information Theory, Tsahkadsor, Armenia, USSR, September 2-8, 1971, Budapest: Akademiai Kiado, p. 267-281.

Munoz, A.R., Real R., Barbosa A.M. & Vargas J.M. (2005) Modelling the distribution of Bonelli's Eagle in Spain: Implications for conservation planning. Diversity and Distributions 11: 477-486

Murtaugh P.A. (2014) In defense of P values. Ecology, 95:611-617

Real R., Barbosa A.M. & Vargas J.M. (2006) Obtaining environmental favourability functions from logistic regression. Environmental and Ecological Statistics 13: 237-245.

Schwarz, G.E. (1978) Estimating the dimension of a model. Annals of Statistics, 6 (2): 461-464.

See Also

step, glm, modelTrim

Examples

```
data(rotif.env)

stepByStep(data = rotif.env, sp.col = 21, var.cols = 5:17)

stepByStep(data = rotif.env, sp.col = 21, var.cols = 5:17, select = "p.value")


# with a model object:

form <- reformulate(names(rotif.env)[5:17], names(rotif.env)[21])
mod <- step(glm(form, data = rotif.env))

stepByStep(data = mod)
```

---

stepwise *Stepwise regression*

Description

This function runs a stepwise regression, selecting and/or excluding variables based on the significance (p-value) of the statistical tests implemented in the add1 and drop1 functions of R.

Usage

```
stepwise(data, sp.col, var.cols, id.col = NULL, family = binomial(link="logit"),
direction = "both", test.in = "Rao", test.out = "LRT", p.in = 0.05, p.out = 0.1,
trace = 1, simplif = TRUE, preds = FALSE, Favourability = FALSE, Wald = FALSE)
```

Arguments

data	a data frame (or an object that can be coerced with 'as.data.frame') containing your target and predictor variables.
sp.col	name or index number of the column of 'data' that contains the response variable.
var.cols	names or index numbers of the columns of 'data' that contain the predictor variables.
id.col	(optional) name or index number of column containing the row identifiers (if defined, it will be included in the output 'predictions' data frame).

family	argument to be passed to glm indicating the error distribution (and optionally the link function) to be used in the model. The default is binomial distribution with logit link (i.e. logistic regression, for binary response variables), and it is the only one that has been tested so far. If you try other options, please carefully check your results and let me know if you find a bug.
direction	the mode of stepwise search. Can be either "forward", "backward", or "both" (the default).
test.in	argument to pass to add1 specifying the statistical test whose 'p.in' a variable must pass to enter the model. Can be "Rao" (the default), "LRT", "Chisq" or "F".
test.out	argument to pass to drop1 specifying the statistical test whose 'p.out' a variable must exceed to be expelled from the model (if it does not simultaneously pass the 'test.in' when direction="both"). Can be "LRT" (the default), "Rao", "Chisq" or "F".
p.in	threshold p-value (default 0.05) for a variable to enter the model.
p.out	threshold p-value (default 0.1) for a variable to leave the model.
trace	if positive, information is printed to the console at each step. The default is 1, for naming each variable that was added or removed. With trace=2, the summary of the model at each step is also printed.
simplif	logical (default TRUE), whether to return a simple output containing only the model object. With FALSE, the output is a list with, additionally, a data frame showing the variable included or excluded at each step.
preds	(if simplif=FALSE) logical, whether to return also the predictions produced by the model at each step.
Favourability	(if simplif=FALSE and preds=TRUE) logical, whether to convert the predictions with the Fav function.
Wald	(if trace > 1) logical (default FALSE), whether to print the Wald test statistics using summaryWald, rather than the z test normally returned by summary. Requires the **aod** package.

Details

Stepwise variable selection is a way of selecting a subset of significant variables to get a simple and easily interpretable model. It is more computationally efficient than best subset selection. This function uses the R functions add1 for selecting and drop1 for excluding variables. The default parameters mimic the "Forward Selection (Conditional)" stepwise procedure implemented in the IBM SPSS software. This is a widely used (e.g. Munoz et al. 2005, Olivero et al. 2017, 2020, Garcia-Carrasco et al. 2021) but also widely criticized (e.g. Harrell 2001; Whittingham et al. 2006; Flom & Cassell, 2007; Smith 2018) method for variable selection, though its AIC-based counterpart (implemented in the step R function) is equally flawed (e.g. Murtaugh 2014; Coelho et al. 2019).

Value

If simplif=TRUE (the default), this function returns the model object obtained after the variable selection procedure. If simplif=FALSE, it returns a list with the following components:

`model`	the model object obtained after the variable selection procedure.
`steps`	a data frame where each row shows the variable included or excluded at each step.
`predictions`	(if preds=TRUE) a data frame where each column contains the predictions of the model obtained at each step. These predictions are probabilities by default, or favourabilities if Favourability=TRUE.

Author(s)

A. Marcia Barbosa

References

Coelho M.T.P., Diniz-Filho J.A. & Rangel T.F. (2019) A parsimonious view of the parsimony principle in ecology and evolution. Ecography, 42:968-976

Flom P.L. & Cassell D.L. (2007) Stopping stepwise: Why stepwise and similar selection methods are bad, and what you should use. NESUG 2007

Garcia-Carrasco J.M., Munoz A.R., Olivero J., Segura M. & Real R. (2021) Predicting the spatio-temporal spread of West Nile virus in Europe. PLoS Neglected Tropical Diseases 15(1):e0009022

Harrell F.E. (2001) Regression modeling strategies: With applications to linear models, logistic regression, and survival analysis. Springer-Verlag, New York

Munoz, A.R., Real R., Barbosa A.M. & Vargas J.M. (2005) Modelling the distribution of Bonelli's Eagle in Spain: Implications for conservation planning. Diversity and Distributions 11: 477-486

Murtaugh P.A. (2014) In defense of P values. Ecology, 95:611-617

Olivero J., Fa J.E., Real R., Marquez A.L., Farfan M.A., Vargas J.M, Gaveau D., Salim M.A., Park D., Suter J., King S., Leendertz S.A., Sheil D. & Nasi R. (2017) Recent loss of closed forests is associated with Ebola virus disease outbreaks. Scientific Reports 7: 14291

Olivero J., Fa J.E., Farfan M.A., Marquez A.L., Real R., Juste F.J., Leendertz S.A. & Nasi R. (2020) Human activities link fruit bat presence to Ebola virus disease outbreaks. Mammal Review 50:1-10

Smith G. (2018) Step away from stepwise. Journal of Big Data 32 (https://doi.org/10.1186/s40537-018-0143-6)

Whittingham M.J., Stephens P.A., Bradbury R.B. & Freckleton R.P. (2006) Why do we still use stepwise modelling in ecology and behaviour? Journal of Animal Ecology, 75:1182-1189

See Also

`step`, `stepByStep`, `modelTrim`

Examples

```
data(rotif.env)

stepwise(data = rotif.env, sp.col = 21, var.cols = 5:17)

sw <- stepwise(data = rotif.env, sp.col = 21, var.cols = 5:17, simplif = FALSE)
sw
```

summaryWald *Model summary with Wald (instead of z) test statistics*

Description

This function produces a summary of a generalized linear model, with the Wald test (instead of the z test) and associated statistics.

Usage

```
summaryWald(model, interceptLast = TRUE)
```

Arguments

`model`	a model object of class "glm".
`interceptLast`	logical, whether to place the intercept in the last (rasther than the first) row of the output. Defaults to TRUE.

Details

This function requires the **aod** package, whose `wald.test` function is used for computing the Wald test.

Value

This function returns a data frame with the model summary statistics.

Author(s)

A. Marcia Barbosa

See Also

`summary`

Examples

```
# load sample data:

data(rotif.env)
names(rotif.env)


# build a model of a species' occurrence based on
# some of the variables:

model <- glm(Abrigh ~ Area + Altitude + AltitudeRange + HabitatDiversity +
HumanPopulation, family = binomial, data = rotif.env)
```

```
# get the Wald-based model summary:

summaryWald(model)
```

`timer` *Timer*

Description

Reporting of time elapsed since a given start time, or during the running of an expression.

Usage

```
timer(..., digits = 1)
```

Arguments

`...`	A date-time object of class `POSIXct`, e.g. as given by `Sys.time`; or an expression to be timed.
`digits`	integer value specifying the number of decimal places to `round` the output to.

Value

The function returns a message informing of the time elapsed since the input timestamp (if it is a date-time object of class 'POSIXct'), or during the running of the input expression.

Author(s)

A. Marcia Barbosa

See Also

`Sys.time`, `proc.time`, `difftime`

Examples

```
# get starting time:
start <- Sys.time()

# do some random analysis:
x <- sort(rnorm(1e7))

# see how long it took:
timer(start)


# time an expression directly:
timer(x <- sort(rnorm(1e6)))
timer(x <- sort(rnorm(1e6)), digits = 2)
```

transpose *Transpose (part of) a matrix or dataframe*

Description

This function transposes (a specified part of) a matrix or data frame, optionally using one of its columns as column names for the transposed result. It can be useful for turning a species presence-absence table into a regional species composition table.

Usage

```
transpose(data, sp.cols = 1:ncol(data), reg.names = NULL)
```

Arguments

`data`	a matrix or data frame containing the species occurrence data to transpose.
`sp.cols`	names or index numbers of the columns containing the species occurrences in 'data' which are meant to be transposed.
`reg.names`	name or index number of the column in 'data' containing the region names, to be used as column names in the transposed result.

Value

This function returns the transposed 'sp.cols' of 'data', with the column specified in 'reg.names' as column names.

Author(s)

A. Marcia Barbosa

See Also

`t`

Examples

```
data(rotif.env)

head(rotif.env)

names(rotif.env)

rotif.reg <- transpose(rotif.env, sp.cols = 18:47, reg.names = 1)

head(rotif.reg)
```

`triMatInd` *Triangular matrix indices*

Description

This function outputs the indices of one triangle (the lower one by default) of an input square matrix. It is used by `simMat` and, for large matrices, makes it faster than e.g. with `lower.tri` or `upper.tri`.

Usage

```
triMatInd(mat, lower = TRUE, list = FALSE)
```

Arguments

`mat`	a square matrix.
`lower`	logical indicating whether the indices should correspond to the lower triangle. The default is TRUE; FALSE produces the upper triangle indices.
`list`	logical indicating whether the results should be output as a list instead of a matrix. The default is FALSE.

Value

The indices (row, column) of the elements of the matrix that belong to the requested triangle.

Author(s)

A. Marcia Barbosa

References

http://stackoverflow.com/questions/20898684/how-to-efficiently-generate-lower-triangle-indices-of-a-symmetric-matrix

See Also

`lower.tri`, `upper.tri`

Examples

```
mat <- matrix(nrow = 4, ncol = 4)
mat
triMatInd(mat)
triMatInd(mat, list = TRUE)
```

`vulnerability` *(Fuzzy) vulnerability*

Description

This function computes the index of species vulnerability of Estrada et al. (2011), using either crisp (presence/absence, i.e. ones and zeros) or fuzzy (Favourability, between zero and one) values, taking into account the conservation status of each species. Vulnerability is like a (potential) richness index in which more vulnerable species (i.e., those with a more threatened conservation status) have higher weight.

Usage

```
vulnerability(data, sp.cols = 1:ncol(data), categories, na.rm = TRUE)
```

Arguments

`data`	a numeric vector, matrix or data frame containing the presence/absence (ones and zeros) or the Favourability (fuzzy presence, between zero and one) values for the target species.
`sp.cols`	names or index numbers of the columns of 'data' that contain the species values for which to compute vulnerability. The default is to use all columns.
`categories`	numeric vector of the same length as 'sp.cols' (or of length 1 if 'data' is a vector) indicating the IUCN Red List category of each species. This vector should be provided in the same order as the columns in data[, sp.cols]. See Details.
`na.rm`	logical value indicating whether NA values should be removed before the computation.

Details

The numeric values for the 'categories' argument are suggested by Estrada et al. (2011) to be as follows for each species, according to its IUCN Red List category (available at https://www.iucnredlist.org):

Critically endangered (CR): 16

Endangered (EN): 8

Vulnerable (VU): 4

Near Threatened (NT): 2

Least Concern (LC): 1

Data Deficient (DD): 1

Not evaluated (NE): 0

These values follow an exponential scale, because a critically endangered species is generally considered more important than two endangered species, an endangered species more important than two vulnerable species, and so on (Estrada et al. 2011).

Value

This function returns a numeric vulnerability value for each value or each row in 'data'.

Author(s)

A. Marcia Barbosa

References

Estrada A., Real R. & Vargas J.M. (2011) Assessing coincidence between priority conservation areas for vertebrate groups in a Mediterranean hotspot. Biological Conservation, 144: 1120-1129

See Also

rarity

Examples

```
data(rotif.env)

# note the 'categories' below are made up, as rotifers are not on yet redlisted
# see Details above for how to get actual values for your species

vulnerability(rotif.env[ , 18], categories = 8)

vulnerability(rotif.env, sp.cols = "Abrigh", categories = 8)

vulnerability(rotif.env, sp.cols = c("Apriod", "Burceo", "Kcochl"), categories = c(8, 16, 2))


# fuzzy vulnerability (from favourability values):

pred <- multGLM(rotif.env, sp.cols = c("Apriod", "Burceo", "Kcochl"), var.cols = 5:17)$predictions

head(pred)

vulnerability(pred, sp.cols = "Apriod_F", categories = 8)

vulnerability(pred, sp.cols = c("Apriod_F", "Burceo_F", "Kcochl_F"), categories = c(8, 16, 2))
```

Index