wsl.biodiv

March 17, 2023

Type Package

Title A toolbox to efficiently handle Species Distribution Modelling (SDMs) in collaborative projects

Version 0.1.0

Depends R (>= 3.1.0), snow, mgcv, dismo, MASS, class, NMOF, gbm, randomForest, ROCR, cluster, spatstat, raster, terra, foreach, doParallel, usdm, glmnet, neuralnet, rgdal, lme4, MuMIn, ape

Description Package aiming at varied and flexible model fitting and tools for SDMs.
Meta information should be stored (author, date, taxon,...) efficiently.
Model evaluation and prediction should flexibly handle various fitted model objects.
Evaluation metrics for pres-only and pres-abs models should be mutually implemented.
Relationships between saved objects should be preserved.
Filtering and testing tools should help the user to find adequate predictors.
Poisson Point Process models (PPPM) implementation should remain user-friendly.
Regularization and variable selection should be implemented in this framework.

License GPL (>=3)

Encoding UTF-8

LazyData true

RoxygenNote 7.1.2

Authors Philipp Brun [cre,aut] ---------- email: philipp.brun@wsl.ch
Yohann Chauvier [cre,aut] ---------- email: yohann.chauvier@wsl.ch
Rafael Wueest [aut] ---------- email: rafael.wueest@wsl.ch
Patrice Descombes [aut] ---------- email: patrice.descombes@wsl.ch

Collate 'boycei.R'
'ceval.R'
'copy_PseuAbs.R'
'create_env_strata.R'
'df_or_rast.R'
'ecoBoyce.R'
'ecospat_d2.R'
'get_env.R'
'get_thres.R'
'glmnet_predict.R'
'hde.R'
'make_blocks.R'
'make_tiles.R'
'moran_auto.R'
'multi.R'
'multi_input_class.R'

'optme.R'
'permut_importance.R'
'plot_wsl_pseudoabsences.R'
'prd.R'
'prd_pres.R'
'preps.R'
'preva_meta.R'
'prop_sampling.R'
'pseu_targr.R'
'stratify.R'
'summary_wsl_evaluation.R'
'summary_wsl_fit.R'
'Thin_them_function.R'
'upsample_strategic.R'
'upsample_thin.R'
'wsl_ebc.R'
'wsl_evaluate.R'
'wsl_evaluate_pres.R'
'wsl_evaluation_class.R'
'wsl_fit_class.R'
'wsl_fitting.R'
'wsl_obs_filter.R'
'wsl_ppm_env.R'
'wsl_ppm_fit.R'
'wsl_ppm_qscheme.R'
'wsl_ppm_quadrature.R'
'wsl_ppm_window.R'
'wsl_predict.R'
'wsl_predict_pres.R'
'wsl_prediction_class.R'
'wsl_pseudoabsences_class.R'
'wsl_quads_class.R'
'wsl_samplePseuAbs.R'
'wsl_test_quadrature.R'
'wsl_varKeep.R'
'wsl_varPPower.R'

R topics documented:

`boycei` *Boyce index model evaluation sub-function II*

Description

Not to be called directly by the user

Usage

```
boycei(interval, obs, fit)
```

Author(s)

'boycei' from the 'ecospat' package

`ceval` *Do the actual model evaluations*

Description

Not to be called directly by the user

Usage

```
ceval(f, pa, crit, tre = numeric())
```

Author(s)

Philipp Brun

`copy_PseuAbs` *Copy pseudoabsences from wsl.pseudoabsences object for new species*

Description

Copy pseudoabsences from existing wsl.pseudoabsences object to wsl.pseudoabsences object for new species

Usage

```
copy_PseuAbs(env.stack, pres = SpatialPoints(), taxon = character(), x)
```

Arguments

`env.stack`	RasterStack with environmental layers for sampling and extraction. Need to be the same layers as in the wsl.pseudoabsences object.
`pres`	SpatialPoints object. Location of presence points. Necessary for 'geographic' sampling strategy and the best adding point if the downstream functions will be used.
`taxon`	Character; name of taxon of interest to keep track of meta information.
`x`	A wsl.pseudoabsences object

Details

if the desired pseudoabsence sampling strategy is not species-specific it may be more efficient to copy on the sampled pseudoabsences from another object.

Value

an object of class 'wsl.pseudoabsences' that can be plotted and passed on to wsl.flex

Author(s)

Philipp Brun

References

Descombes, P., Chauvier, Y., Brun, P., Righetti, D., Wüest, R. O., Karger, D. N., ... & Zimmermann, N. E. (2022). Strategies for sampling pseudo-absences for species distribution models in complex mountainous terrain. bioRxiv, 2022-03.

Examples

```
### ============================================================================
### Data preparation
### ============================================================================

# Predictors
bio=raster::getData('worldclim',var='bio',lon=16, lat=48,res=.5)
bio=bio[[c(1,4,12)]]

# install.packages("rgbif")
library(rgbif)
# extract species
spn1='Boletus aestivalis'
xt=as.vector(extent(bio))
baest <- occ_search(scientificName=spn1,
                  hasCoordinate=TRUE,
                  decimalLongitude=paste0(xt[1],",",xt[3]),
                  decimalLatitude=paste0(xt[2],",",xt[4]))

pbaest=baest$data[,c('decimalLongitude','decimalLatitude')]
baest_spp=SpatialPoints(pbaest,proj4string = crs(bio))

spn2='Boletus edulis'
bedu <- occ_search(scientificName=spn2,
                  hasCoordinate=TRUE,
                  decimalLongitude=paste0(xt[1],",",xt[3]),
                  decimalLatitude=paste0(xt[2],",",xt[4]))

pbedu=bedu$data[,c('decimalLongitude','decimalLatitude')]
bedu_spp=SpatialPoints(pbedu,proj4string = crs(bio))

### ============================================================================
### Sample pseudoabsences
### ============================================================================

# Geo.start
pseu.abs1=wsl.samplePseuAbs(type="geo.strat",
                            n=5000,
                            env.stack=bio,
                            pres=baest_spp,
                            geores_fact=3,
                            add.strat=0,
                            taxon=spn1)

plot(pseu.abs1)
```

```
pseu.abs2=copy_PseuAbs(env.stack=bio,
                       pres=bedu_spp,
                       taxon=spn2,
                       x=pseu.abs1)

plot(pseu.abs2)
```

`create_envstrat` *Create environmental strata for environmentally stratified pseudoabsence sampling.*

Description

Create environmental strata for environmentally stratified pseudoabsence sampling.

Usage

```
create_envstrat(
  env.stk,
  rAll = TRUE,
  save_it = TRUE,
  strat_dir = NA,
  poolsiz = 5 * 10^6,
  sampsiz = 4e+05,
  type
)
```

Arguments

`env.stk`	raster stack with environmental layers of interest
`rAll`	should raster be read entirely into memory? This is faster but may blow the RAMs of your machine.
`save_it`	Should a stratified sample of c. 400000 points be saved
`strat_dir`	Directory to save the stratified sample in, Not to be called directly by the user

Author(s)

Philipp Brun

df_or_rast *Predict to data.frame or raster*

Description

Not to be called directly by the user

Usage

```
df_or_rast(mod, nwdat, clust, ...)
```

Author(s)

Philipp Brun

eco.adj.D2.glm *Function originally written by A. Guisan (UNIL-ECOSPAT,Switzerland) and modified by C. Randin (UNIL-ECOSPAT,Switzerland) S-Plus Function for calculating an adjusted D2 (see Weisberg, 1980) Takes any GLM object as argument*

Description

Not to be called directly by the user

Usage

```
eco.adj.D2.glm(glm.obj)
```

Author(s)

'ecospat.adj.D2.glm' from the 'ecospat' package

ecoBoyce *Boyce index model evaluation sub-function I*

Description

Not to be called directly by the user

Usage

```
ecoBoyce(fit, obs, nclass = 0, window.w = "default", res = 100, PEplot = TRUE)
```

Author(s)

'ecospat.boyce' from the 'ecospat' package

`fitdoc` *Fit presence-absence models*

Description

Flexibly fit various types of functions but let framework take care of resampling, meta-info storage, and file saving. wsl.flex basically allows supplying any possible model, however, there may be problems with prediction/evaluation for exotic functions. Available model algorithms for fitting are: Generalized Linear Models (GLM), Generalized additive models (GAM), Maximum Entropy (MaxEnt), Artificial Neural Networks (ANN), Generalized Boosted regression Models/Boosted Regression Trees (GBM) and Random Forest (RF)

Usage

```
wsl.glm(
  x = numeric(),
  pa = numeric(),
  env_vars = data.frame(),
  taxon = character(),
  replicatetype = character(),
  reps,
  strata = NA,
  save = FALSE,
  project = NA,
  path = NA,
  step = FALSE,
  mod_tag = "",
  xy = NULL,
  ...
)

wsl.gam(
  x = numeric(),
  pa = numeric(),
  env_vars = data.frame(),
  taxon = character(),
  replicatetype = character(),
  reps,
  strata = NA,
  save = FALSE,
  project = NA,
  path = NA,
  step = FALSE,
  mod_tag = "",
  xy = NULL,
  ...
)

wsl.maxent(
  x = numeric(),
  pa = numeric(),
```

```
  env_vars = data.frame(),
  taxon = character(),
  replicatetype = character(),
  reps,
  strata = NA,
  save = FALSE,
  project = NA,
  path = NA,
  mod_tag = "",
  xy = NULL,
  ...
)

wsl.gbm(
  x = numeric(),
  pa = numeric(),
  env_vars = data.frame(),
  taxon = character(),
  replicatetype = character(),
  reps,
  strata = NA,
  save = FALSE,
  project = NA,
  path = NA,
  mod_tag = "",
  xy = NULL,
  ...
)

wsl.ann(
  x = numeric(),
  pa = numeric(),
  env_vars = data.frame(),
  taxon = character(),
  replicatetype = character(),
  reps,
  strata = NA,
  save = FALSE,
  project = NA,
  path = NA,
  mod_tag = "",
  xy = NULL,
  ...
)

wsl.flex(
  x = numeric(),
  pa = numeric(),
  env_vars = data.frame(),
  taxon = character(),
  replicatetype = character(),
  reps,
```

```
  strata = NA,
  save = FALSE,
  project = NA,
  path = NA,
  mod_args = list(),
  xy = NULL
)
```

Arguments

x	Optional. Object of class 'wsl.pseudoabsences'. If used 'pa', 'env_vars' and 'taxon' will be replaced by new set of values set in the object.
pa	Object of class 'vector' with presence/absence values
env_vars	Object of class 'data.frame' with environmental predictors
taxon	Name of the taxon for which models are fitted
replicatetype	(How) should replicates be generated? May be 'none', 'splitsample', 'cv' or 'block-cv'
reps	Number of replicates
strata	A numeric vector of the same length as observations with integers separating cross validation replicates. Only used when replicatetype='block-cv'
save	Should the model be saved in a structured way? (not implemented yet)
project	Character indicating the name of the project within which the models are run (later used to define saving directories)
path	Where to save? (not implemented yet)
step	For glms and gams only. Should the models be updated with the step function?
mod_tag	Not in wsl.flex. Descriptive label for current model
xy	Optional. XY coordinates of observations. Used for post evaluation with presence-only metric when pres_abs=TRUE in wsl.evaluate.pres(). Must be a data.frame() or matrix().
mod_args	List with elements of class 'multi.input' which specify models to be fitted in wsl.flex

Value

Object of class wsl.fit including slots for meta info, testing data for evaluation, and model objects

Author(s)

Philipp Brun, Yohann Chauvier

Examples

```
# Take anguilla data set from dismo package
data("Anguilla_train")
vrs=c("SegSumT","USRainDays","USSlope")
env=Anguilla_train[,vrs]

### Check out wsl.glm
form.glm=as.formula(paste("Presence~",paste(paste0("poly(",vrs,",2)"),collapse="+")))
```

```
modi1=wsl.glm(pa=Anguilla_train$Angaus,
              env_vars = env,
              taxon="Angaus",
              replicatetype="cv",
              reps=5,
              project="prototest",
              mod_tag="test-glm",
              formula=form.glm,
              family="binomial",
              step=TRUE)

# Try out custom summary function
summary(modi1)

# Access glm object of first replicate
summary(modi1@fits$replicate_01$`test-glm`)

# Evaluate the model
eval1=wsl.evaluate.pa(modi1)

# Get evaluation summary
summary(eval1)

### Check out wsl.gam
form.gam=as.formula(paste("Presence~",paste(paste0("s(",vrs,")"),collapse="+")))

# Try out wsl.glm funcion
modi2=wsl.gam(pa=Anguilla_train$Angaus,
              env_vars = env,
              taxon="Angaus",
              replicatetype="splitsample",
              reps=3,
              project="prototest",
              mod_tag="test-gam",
              formula=form.gam,
              family="binomial",
              step=FALSE)

# Try out custom summary function
summary(modi2)

# Access glm object of first replicate
summary(modi2@fits$replicate_01$`test-gam`)

# Evaluate the model
eval2=wsl.evaluate.pa(modi2,crit="maxTSS")

# Get evaluation summary
summary(eval2)

### Check out wsl.gbm
form.gbm=as.formula(Presence ~ .)

# Try out wsl.glm funcion
modi3=wsl.gbm(pa=Anguilla_train$Angaus,
              env_vars = env,
```

```
                taxon="Angaus",
                replicatetype="none",
                reps=1,
                project="prototest",
                mod_tag="test-brt",
                formula= form.gbm,
                distribution = "bernoulli",
                interaction.depth = 1,
                shrinkage=.01,
                n.trees = 3500)

# Try out custom summary function
summary(modi3)

# Access glm object of first replicate
summary(modi3@fits$replicate_01$`test-brt`)

# Prepare external testing data
tste=data.frame(Presence=Anguilla_train$Angaus,CV=1,env)

# Evaluate the model
eval3=wsl.evaluate.pa(modi3,crit="maxTSS",tester=tste)

# Get evaluation summary
summary(eval3)

### Check out wsl.maxent
feat=c("linear=true","quadratic=true","hinge=true","product=true","threshold=false")

# Try out wsl.glm funcion
modi4=wsl.maxent(pa=Anguilla_train$Angaus,
                 env_vars = env,
                 taxon="Angaus",
                 replicatetype="block-cv",
                 reps=3,
                 strata=sample(1:3,nrow(env),replace=TRUE),
                 project="prototest",
                 mod_tag="test-mxe",
                 args=feat)

# Try out custom summary function
summary(modi4)

# Access glm object of first replicate
summary(modi4@fits$replicate_01$`test-mxe`)

# Define external threshold
thmxe=c(`test-mxe`=0.5)

# Evaluate the model
eval4=wsl.evaluate.pa(modi4,crit="external",thres=thmxe)

# Get evaluation summary
summary(eval4)

# Get thresholds
thr.4=get_thres(eval4,mean=FALSE)
```

```
### Check out wsl.flex
form.glm.2=as.formula(paste("Presence~",paste(vrs,collapse="+")))

modinp=list(multi("glm",list(formula=form.glm,family="binomial"),"glm-simple",step=TRUE,weight=TRUE),
   multi("gbm",list(formula=form.gbm,
   distribution = "bernoulli",
   interaction.depth = 1,
   shrinkage=.01,
   n.trees = 3500),"gbm-simple"),
  multi("gam",list(formula=form.gam,family="binomial"),"gam-simple",step=FALSE,weight=TRUE),
   multi("maxent",list(args=feat),"mxe-simple"),
   multi("randomForest",list(formula=form.gbm,ntree=500,maxnodes=NULL),"waud1"),
  multi("glm",list(formula=form.glm.2,family="binomial"),"glm-lin",step=TRUE,weight=TRUE))

# Try out wsl.glm funcion
modi5=wsl.flex(pa=Anguilla_train$Angaus,
               env_vars = env,
               taxon="Angaus",
               replicatetype="block-cv",
               reps=3,
               strata=sample(1:3,nrow(env),replace=TRUE),
               project="multitest",
               mod_args=modinp)

# Try out custom summary function
summary(modi5)

# Access glm object of first replicate
summary(modi5@fits$replicate_01$`glm-simple`)

# Evaluate the model
eval5<-wsl.evaluate.pa(modi5,crit="pp=op")

# Get evaluation summary
summary(eval5)

# Get thresholds
thr.5=get_thres(eval5, mean=FALSE)
```

`fitppm` *Fit Poisson Point Process Models (PPPM)*

Description

PPPM (Poisson Point Process Models) is a modelling approach genrally design to use observation-only data aka point occurences. It allows the user to model species observations intensity per unit area (i.e. the density of presence points over a spatial grid) as a log linear function of predictors. Quadrature points (or background points) are necessary in the model, and may be generated with the 'wsl.quadrature' function. Those points apply a spatial scaling proportional to the study area and estimate the maximised model log likelihood (see Renner 2013, Renner et al. 2015). 'wsl.ppmGlasso' applies a point process model, but also by choosing to implement regularisation and variable selection following the package "glmnet".

Usage

```
wsl.ppmGlasso(
  pres = data.frame(),
  env_vars = matrix(),
  quadPoints = wsl.quads(),
  asurface = numeric(),
  taxon = character(),
  replicatetype = character(),
  reps,
  strata = NA,
  save = FALSE,
  project = NA,
  path = NA,
  mod_tag = "",
  poly = TRUE,
  which_poly = NULL,
  lasso = TRUE,
  penalty.glmnet = NULL,
  ...
)
```

Arguments

pres	Object of class 'data.frame' or 'matrix'. Coordinates xy of Species observations.
env_vars	If 'wsl.ppmGlasso' used, must be an object of class 'matrix' or 'data.frame' with environmental predictor values. Note that categorical predictor values must be of class factor for both 'env_vars' and 'quadPoints@Qenv'.
quadPoints	If 'wsl.ppmGlasso' used, must a 'wsl.quads' object generated with 'wsl.quadrature' function and 'lasso=TRUE'.
asurface	The surface of the study area in square kilometers.
taxon	Name of the taxon for which models are fitted.
replicatetype	(How) should replicates be generated? may be 'none', 'splitsample', 'cv' or 'block-cv'.
reps	Number of replicates.
strata	A numeric vector of the same length as observations + quadrature points with integers assigning cross validation replicates. Only used when replicatetype='block-cv'. Note: the vector must first integrate CV information for observation points.
save	Should the model be saved in a structured way? (not implemented yet).
project	Character indicating the name of the project within which the models are run (later used to define saving directories).
path	Where to save? (not implemented yet).
mod_tag	Descriptive label for current model.
poly	If TRUE, PPPM fits a second order polynomial regression or a custom equation (see 'formula')
which_poly	Which predictors should be using polynomial terms? Use a binary vector that specify which variables/predictors. Length of vector must be equal to the number of input variables. '1' stands for poly=TRUE whereas '0' stands for poly=FALSE. Default is polynomial for all if poly=TRUE.

lasso	If FALSE no regularisation is applied.
penalty.glmnet	If 'lasso=TRUE', a binary vector that specify which variables/predictors used to model should be shrinked. Length of vector must be equal to the number of input variables. '1' stands for shrinkage whereas '0' stands for no shrinkage, i.e. the variable will always be included in the model.
...	If 'wsl.ppmGlasso' used with lasso = TRUE, arguments passed on to the cv.glmnet() function (package 'glmnet') use to apply a Lasso, Ridge or Elastic Net regularisation. To notice that the package's argument 'penalty.factor' is not needed here. If used, the parameter 'penalty.glmnet' must instead be filled for each 'env_vars'. If lasso = FALSE, arguments passed on to the glm("poisson") function.
formula	Optional. Equation of desired fit (in developments...).

Value

Object of class wsl.fit including slots for meta info, testing data for evaluation, and model objects.

Author(s)

Yohann Chauvier, Philipp Brun

Examples

```
# Load
data(AlpineConvention_lonlat)
data(exrst)
rst = rst[[1:6]]
data(xy_ppm)
mypoints = xy.ppm[,c("x","y")]

# Define mask
maskR = mask(rst[[1]],shp.lonlat)

# Run 'wsl.ppm.window' function
wind = wsl.ppm.window(mask = maskR,
                      val = 1,
                      owin = TRUE)

# nDefine quadrature points for 'wsl.ppmGlasso'
quadG1 = wsl.quadrature(mask = maskR,
                        area.win = wind,
                        random = FALSE,
                        lasso = TRUE,
                        env_vars = rst)

# Define your environments
envG = raster::extract(rst,mypoints)

# Spatial block cross-validation
to_b_xy = rbind(mypoints,quadG1@coords)
toSamp = c(rep(1,nrow(mypoints)),rep(0,nrow(quadG1@coords)))
block_cv_xy = make_blocks(nstrat = 5, df = to_b_xy, nclusters = 10, pres = toSamp)

# Environmental block cross-validation
```

```
to_b_env = rbind(envG,quadG1@Qenv[,-1])
block_cv_env = make_blocks(nstrat = 5, df = to_b_env, nclusters = 10, pres = toSamp)

# 'wsl.ppmGlasso' (alpha = 0.5 => Elastic net, see package 'glmnet')
   # Complex PPP lasso (poly = TRUE & lasso=TRUE)
ppm.lasso = wsl.ppmGlasso(pres = mypoints,
                        quadPoints = quadG1,
                        asurface = raster::area(shp.lonlat)/1000,
                        env_vars = envG,
                        taxon = "species_eg1",
                        replicatetype = "cv",
                        reps = 5,
                        strata = NA,
                        save=FALSE,
                        project = "lasso_eg1",
                        path = NA,
                        poly = TRUE,
                        lasso = TRUE,
                        alpha = 0.5,
                        type.measure = "mse",
                        standardize = TRUE,
                        nfolds = 5,
                        nlambda = 100)
summary(ppm.lasso)

  # Simple PPP (poly = FALSE & lasso=FALSE) + block-cross validation
ppm.simple = wsl.ppmGlasso(pres = mypoints,
                        quadPoints = quadG1,
                        asurface = raster::area(shp.lonlat)/1000,
                        env_vars = envG,
                        taxon = "species_eg2",
                        replicatetype = "block-cv",
                        reps = 5,
                        strata = block_cv_xy,
                        save = FALSE,
                        project = "lasso_eg2",
                        path = NA,
                        poly = FALSE,
                        lasso = FALSE)
```

`get_env` *Extract environments based on yearly observations (local or URL sources)*

Description

Based on local files or URL sources, this function extracts environmental values from given observations at their year of sampling. Although the function was developped mainly for CHELSA (https://chelsa-climate.org/), it can easily be used for other type of data sources if following the same file format (i.e. single raster files, informed environments and year in the URL or file names)

Usage

```
get_env(
  env_layers = NULL,
  env_id = NULL,
  year_id = NULL,
  obs = data.frame(),
  d.path = getwd()
)
```

Arguments

env_layers	Vector of file names or URL links of desired environmental layers. The given strings must have the identity of the variable and the year informed.
env_id	Vector. What are the unique names (as specified in 'env_layers') of the target variables?
year_id	Vector. What are the unique years (as specified in 'env_layers') we want the environment from?
obs	A data.frame with three columns (1-Longitude, 2-Latitude and 3-year of sampling). The observations needs to be from the same CRS as the 'env_layers'.
d.path	Character. The folder path where the environmental layers should be saved or stored.

Value

A data.frame describing for each observations their associated environmental values.

References

...

Examples

```
# Packages
library(wsl.biodiv)
data(xy_ppm)

# URL links
url.base = "https://os.zhdk.cloud.switch.ch/envicloud/chelsa/chelsa_V1/chelsa_cruts/"
url.sprint1 = sprintf(paste0("tmax/CHELSAcruts_tmax_10_%d_V.1.0.tif"),1901:1911)
url.sprint2 = sprintf(paste0("tmin/CHELSAcruts_tmin_10_%d_V.1.0.tif"),1906:1921)
URL.links = paste0(url.base,c(url.sprint1,url.sprint2))

# Dummy observations
random.year = (1901:1921)[sample(1:21,nrow(xy.ppm),replace=TRUE)]
obs = data.frame(xy.ppm[,c("x","y")],year=random.year)

# Run function
sp.env = get_env(env_layers = URL.links,
                 env_id = c("tmax","tmin"),
                 year_id = 1901:1921,
                 obs = obs,
                 d.path = getwd())
```

get_thres *Get threshold*

Description

Extracts thresholds from wsl.evaluation objects and names them so they can be fed to the wsl.predict function. At the moment only averages over replicates can be obtained.

Usage

```
get_thres(x, mean)
```

Arguments

x	An object of class wsl.evaluation
mean	Logical. If TRUE, a mean is applied to all thresholds

Value

numeric() or 'vector' with thresholds

Author(s)

Philipp Brun, Yohann Chauvier

Examples

```
# Take anguilla data set from dismo package
data("Anguilla_train")
vrs=c("SegSumT","USRainDays","USSlope")
env=Anguilla_train[,vrs]

### Check out wsl.glm
form.glm=as.formula(paste("Presence~",paste(paste0("poly(",vrs,",2)"),collapse="+")))

modi1=wsl.glm(pa=Anguilla_train$Angaus,
              env_vars = env,
              taxon="Angaus",
              replicatetype="cv",
              reps=5,
              project="prototest",
              mod_tag="test-glm",
              formula=form.glm,
              family="binomial",
              step=TRUE)

# Evaluate the model
eval1=wsl.evaluate.pa(modi1)

# Get thresholds
get_thres(eval1, mean = FALSE)
get_thres(eval1, mean = TRUE)
```

hde *Hide annoying prints*

Description

Not to be called directly by the user

Usage

```
hde(x)
```

Author(s)

Philipp Brun

make_blocks *Block-wise split data into training and testing*

Description

Creates a stratum vector based on a data.frame with n columns. If the data.frame has one column strata are created based on clusters separated by quantiles. If the data.frame has two or more columns, strata are created based on k-medoid clusters (function 'pam' from package cluster). Instead of a data.frame also the argument 'npoints' can be provided, then groups are created by random sampling. An opitimization algorithm (function 'gridSearch' from package NMOF) optimizes for equal stratum sizes.

Usage

```
make_blocks(
  nstrat = 4,
  df = data.frame(),
  nclusters = nstrat * 5,
  npoints = NA,
  pres = numeric()
)
```

Arguments

nstrat	Number of approximately equal-sized classes to separate groups in block-cross validation
df	Object of class 'data.frame' with n columns containing critera for cluster building. Not necessary if argument npoints is supplied
nclusters	Number of clusters based on which strata should be built. Minimum is the same number as 'nstrat'. Maximum is nrow(df)/10.
npoints	Optional argument if 'df' is not supplied. For how many points should random sampling be made?

pres — Binary vector. Optional argument. If 'df' is supplied, the argument can be used to save processing time or if number of row/points > 65,536 (pam hard-limit). '1' stands for the points on which k-medoid clustering is appplied (most likely the species observations), and '0' stands for the points on which K-nearest neighbors is applied relative to the '1' (most likely the absences, background points...). If 'df' is not supplied. For which points should random sampling be made?

Value

Object of class 'vector' of length nrow(df) or 'npoints', with integers defining different strata

Author(s)

Philipp Brun

Examples

```
### Test out block generation function

data("Anguilla_train")
vrs=c("SegSumT","USRainDays","USSlope")
env=Anguilla_train[,vrs]

# No layers supplied
strt.1=make_blocks(npoints=1000)
table(strt.1)

# Stratified by 1d layer a
strt.2=make_blocks(df=env[,1,drop=F],nstrat=5,nclusters=5)
table(strt.2)

# Stratified by 1d layer b
strt.3=make_blocks(df=env[,1,drop=F],nstrat=5,nclusters=15)
table(strt.3)

# Stratified by 2d layer a
strt.4=make_blocks(df=env[,c(1,3)],nstrat=3,nclusters=3)
table(strt.4)

# Stratified by 2d layer b
strt.5=make_blocks(df=env[,c(1,3)],nstrat=5,nclusters=50)
table(strt.5)

# Stratified by 3d layer
strt.6=make_blocks(df=env[,1:3],nstrat=5,nclusters=50)
table(strt.6)

par(mfrow=c(3,2))
plot(env[,c(1,3)],col=strt.1)
plot(env[,c(1,3)],col=strt.2)
plot(env[,c(1,3)],col=strt.3)
plot(env[,c(1,3)],col=strt.4)
plot(env[,c(1,3)],col=strt.5)
plot(env[,c(1,3)],col=strt.6)
```

```
### Test out function by using the 'pres' argument

# Load
data(AlpineConvention_lonlat)
data(exrst)
rst = rst[[1:6]]
data(xy_ppm)
mypoints = xy.ppm[,c("x","y")]

# Define mask
maskR = mask(rst[[1]],shp.lonlat)

# Run 'wsl.ppm.window' function
wind = wsl.ppm.window(mask = maskR,
                      val = 1,
                      owin = TRUE)

# nDefine quadrature points for 'wsl.ppmGlasso'
quadG1 = wsl.quadrature(mask = maskR,
                        area.win = wind,
                        random = FALSE,
                        lasso = TRUE,
                        env_vars = rst)

# Define your environments
envG = raster::extract(rst,mypoints)

# Spatial block cross-validation
to_b_xy = rbind(mypoints,quadG1@coords)
toSamp = c(rep(1,nrow(mypoints)),rep(0,nrow(quadG1@coords)))
block_cv_xy = make_blocks(nstrat = 5, df = to_b_xy, nclusters = 10, pres = toSamp)

# Environmental block cross-validation
to_b_env = rbind(envG,quadG1@Qenv[,-1])
block_cv_env = make_blocks(nstrat = 5, df = to_b_env, nclusters = 10, pres = toSamp)
```

`make_tiles` *Create a specific number of tiles based on a raster extent*

Description

Based on a specific extent, one or several tiles are generated. Tiles can be smaller raster extents or geometry arguments POLYGON(). The original extent is therefore either converted into a POLYGON() argument, or divided into Ntiles of regular fragments which are converted into POLYGON() arguments and smaller SpatExtent.

Usage

```
make_tiles(geo, Ntiles, sext = TRUE)
```

Arguments

geo	Object of class 'Extent', 'SpatExtent', 'SpatialPolygon', 'SpatialPolygonDataframe', or 'SpaVector' (WGS84 or planar) to define the study's area extent. Default is NULL i.e. the whole globe.
Ntiles	Numeric. In how many tiles/fragments should geo be divided approximately?
sext	Logical. Should a list of SpatExtent also be returned for each generated POLYGON()?

Value

A list of geometry arguments POLYGON() of length Ntiles (and of SpatExtent if sext=TRUE)

References

Chauvier, Y., Thuiller, W., Brun, P., Lavergne, S., Descombes, P., Karger, D. N., ... & Zimmermann, N. E. (2021). Influence of climate, soil, and land cover on plant species distribution in the European Alps. Ecological monographs, 91(2), e01433. 10.1002/ecm.1433

Examples

```
# Load the European Alps Extent
data(geo_dat)

# Apply the function to divide the extent in ~20 fragments
mt = make_tiles(geo=shp.lonlat,Ntiles=20,sext=TRUE); mt
```

---

moranI.test *Test spatial auto-correlation in model ouptuts*

Description

A handful wrapper of moran.I() from the R package 'ape'. Based on the data coordinates & model residuals, the function run a moran test to check if there is any spatial auto-correlation ($p<0.05$) in the model.

Usage

```
moranI.test(xy, resid)
```

Arguments

xy	Object of class data.frame. Coordinates of the data points. First and secon columns must be x and y respectively.
resid	Object of class vector. Model residuals.

Value

moran.I() test outputs

Author(s)

Yohann Chauvier

Examples

```
# Load
data(AlpineConvention_lonlat)
data(exrst)
rst = rst[[1:6]]
data(xy_ppm)
mypoints = xy.ppm[,c("x","y")]

# Define mask
maskR = mask(rst[[1]],shp.lonlat)

# Run 'wsl.ppm.window' function
wind = wsl.ppm.window(mask = maskR,
                      val = 1,
                      owin = TRUE)

# nDefine quadrature points for 'wsl.ppmGlasso'
quadG1 = wsl.quadrature(mask = maskR,
                        area.win = wind,
                        random = FALSE,
                        lasso = TRUE,
                        env_vars = rst)

# Define your environments
envG = raster::extract(rst,mypoints)

  # Simple PPP (poly = FALSE & lasso=FALSE) + block-cross validation
ppm.simple = wsl.ppmGlasso(pres = mypoints,
                      quadPoints = quadG1,
                      asurface = raster::area(shp.lonlat)/1000,
                      env_vars = envG,
                      taxon = "species_eg2",
                      replicatetype = "none",
                      reps = 1,
                      lasso = FALSE)
# Test for auto-correlation
moran.I(xy = rbind(mypoints,quadG1@coords),
        resid = ppm.simple@fits[[1]][[1]]$residuals)
```

`multi` *Define settings for function that should be supplied to wsl.flex*

Description

Create a multi.input object that efficiently stores model specifications

Usage

```
multi(mod, args, tag = "", step = FALSE, weight = FALSE)
```

Arguments

mod	A character with the name of the function to be called. E.g. "gam"
args	A list with arguments to be passed to the function specified in 'mod'
tag	Character with name for model set-up
step	Should step function be applied to update model

Value

Object of class 'multi'

Author(s)

Philipp Brun

Examples

```
### Preliminary

data("Anguilla_train")
vrs=c("SegSumT","USRainDays","USSlope")

form.glm=as.formula(paste("Presence~",paste(paste0("poly(",vrs,",2)"),collapse="+")))
form.gbm=as.formula(Presence ~ .)
form.glm.2=as.formula(paste("Presence~",paste(vrs,collapse="+")))
feat=c("linear=true","quadratic=true","hinge=true","product=true","threshold=false")
form.gam=as.formula(paste("Presence~",paste(paste0("s(",vrs,")"),collapse="+")))

### Multi examples

multi("glm",list(formula=form.glm,family="binomial"),"glm-simple",step=TRUE)
multi("gbm",list(formula=form.gbm,
                 distribution = "bernoulli",
                 interaction.depth = 1,
                 shrinkage=.01,
                 n.trees = 3500),"gbm-simple")
multi("gam",list(formula=form.gam,family="binomial"),"gam-simple",step=FALSE)
multi("maxent",list(args=feat),"mxe-simple")
multi("randomForest",list(formula=form.gbm,ntree=500,maxnodes=NULL),"waud1")
multi("glm",list(formula=form.glm.2,family="binomial"),"glm-lin",step=TRUE)
```

---

multi.input-class *An S4 class to store evaluation data*

Description

An S4 class to store evaluation data

Slots

`mod` Name of model algorithm to be called (character) e.g., 'glm'

`args` A list of arguments to be supplied to the model fitting alrogrithm

`tag` The name of the model setup (character)

`step` A logical indicating whether step function should be ran (for glm and gam)

`weight` Should observations be weighted based on their prevalence?

Author(s)

Philipp Brun

---

optme — *Optimization function to create equal-sized strata in the 'make_blocks' function*

Description

Not to be called directly by the user

Usage

```
optme(x, nms, grps, tot)
```

Author(s)

Philipp Brun

---

prd.pa — *Correctly feed the predict functions depending on model type (glm, gbm, maxent...)*

Description

Not to be called directly by the user

Usage

```
prd.pa(mod, tst, clust = FALSE)
```

Author(s)

Philipp Brun

prd.pres

Correctly feed the predict functions depending on model type (ppm, glm, gbm, maxent...)

Description

Not to be called directly by the user

Usage

```
prd.pres(
  mod,
  env_vars,
  window,
  polly,
  meta,
  coefs,
  id.fact = NULL,
  valid.pres = NULL,
  env_samp = NULL
)
```

Author(s)

Yohann Chauvier

preps

Check input data, collect meta information, take care of data subsetting. Called by model fitting functions.

Description

Not to be called directly by the user

Usage

```
preps(env = parent.frame(), call)
```

Author(s)

Philipp Brun, Yohann Chauvier

preva.meta *Generate meta information for prediction and evaluation*

Description

Not to be called directly by the user

Usage

```
preva.meta(env = parent.frame(), type = character())
```

Author(s)

Philipp

prop.sampling *Sample pseudo-absences proportional to target group distribution*

Description

Uses the 'density' function from the package spatstat to create a density surface of the supplied point pattern and samples pseudo-absences from this density-distribution.

Usage

```
prop.sampling(points, nsamples = 1000, res = 1, ...)
```

Arguments

points	matrix or data.frame with column names 'x' and 'y' assumed to be on the same scale and metric distances (m, km,...)
nsamples	number of pseudoabsences to be generated
res	resolution of the denstiy grid from which pseudo-absences are drawn. default (1) corresponds to 1000 cells on the x-axis
...	arguments passed on to the 'density' function from package spatstat. Note in particluar the argument 'adjust' which controls the kernel size in the density interpolation.

Value

nsamples x 2 matrix with drawn psuedo-absences

Author(s)

Philipp Brun

Examples

```
### Load points

data(mypts)

### Test proportional sampling

pseu.1=prop.sampling(points=my.pts,nsamples=10000,adjust=1,res=1)
pseu.2=prop.sampling(points=my.pts,nsamples=10000,adjust=.5,res=1)
pseu.3=prop.sampling(points=my.pts,nsamples=10000,adjust=.2,res=1)
pseu.4=prop.sampling(points=my.pts,nsamples=10000,adjust=.1,res=1)

par(mfrow=c(2,2))

plot(pseu.1,pch=16,col="red",cex=.5)
points(my.pts)
plot(pseu.2,pch=16,col="red",cex=.5)
points(my.pts)
plot(pseu.3,pch=16,col="red",cex=.5)
points(my.pts)
plot(pseu.4,pch=16,col="red",cex=.5)
points(my.pts)
```

---

pseu.targr *Sample pseudo-absences proportional to target group distribution 2.0*

Description

This is essentially an improved version of prop.sampling. It uses the 'density' function from the package 'spatstat' to create a density surface of the supplied point pattern and samples pseudo-absences from this density-distribution. It additionally requires a raster layer (env.layer) as input, typically containing an environmental variable used for modelling to extract information about the extent of the study area and the resolution of the environmental information. Furthermore, a mask of class Spatial can be supplied to constrain the area within which pseudoabsences should be sampled. If no mask is supplied, pseudoabsences are sampled in the non-NA cells of the env.layer.

Usage

```
pseu.targr(
  points,
  nsamples = 1000,
  p.avoid = NULL,
  env.layer,
  mask = NULL,
  ...
)
```

Arguments

points	Matrix or data.frame with column names 'x' and 'y' assumed to be on the same scale and metric distances (m, km,...)

nsamples	Number of pseudoabsences to be generated
p.avoid	Same type and conditions as "points" argument; avoid sampling pseudo-absences in cells where input coordinates fall
env.layer	A raster layer of desired resolution and extent (and area of interest -> non_NA cells)
mask	Optional. An object of class Spatial to mask for sampling area of interest
...	Additional arugements to the density.ppp function (package 'spatstat')

Value

nsamples x 2 matrix with drawn psuedo-absences

Author(s)

Philipp Brun, Yohann Chauvier

Examples

```
# Load files

data(AlpineConvention_lonlat)
data(exrst)
data(xy_ppm)
mypoints=xy.ppm[,c("x","y")]
id.avoid=sample(1:nrow(mypoints),500,replace=TRUE)
xy.avoid=mypoints[id.avoid,]
xy.process=mypoints[-id.avoid,]

# Run pseu.targ()

dens1=pseu.targr(points=xy.process,nsamples=5000,p.avoid=xy.avoid,
                 env.layer=rst[[1]],mask=shp.lonlat,adjust=0.1)
dens2=pseu.targr(points=xy.process,nsamples=5000,p.avoid=xy.avoid,
                 env.layer=rst[[1]],mask=shp.lonlat,adjust=0.5)
dens3=pseu.targr(points=xy.process,nsamples=1000,p.avoid=NULL,
                 env.layer=rst[[1]],mask=NULL,adjust=0.1)
dens4=pseu.targr(points=xy.process,nsamples=1000,p.avoid=NULL,
                 env.layer=rst[[1]],mask=NULL,adjust=0.5)

# Plot results

par(mfrow=c(2,2))

plot(rst[[1]])
points(dens1,pch=20,cex=0.3)
plot(rst[[1]])
points(dens2,pch=20,cex=0.3)
plot(rst[[1]])
points(dens3,pch=20,cex=0.3)
plot(rst[[1]])
points(dens4,pch=20,cex=0.3)
```

stratify *Subsample based on two stratification critera*

Description

Subsample based on two stratification critera

Usage

```
stratify(spp, type, sampsiz)
```

Arguments

spp	Spatial data.frame
type	either 'env.strat' or 'env.semi.strat'
sampsiz	desired size of the subsample Not to be called directly by the user

Author(s)

Philipp Brun

References

Descombes, P., Chauvier, Y., Brun, P., Righetti, D., Wüest, R. O., Karger, D. N., ... & Zimmermann, N. E. (2022). Strategies for sampling pseudo-absences for species distribution models in complex mountainous terrain. bioRxiv, 2022-03.

thin_them *Thin a spatial points object by number of points or minimum distance*

Description

Thin a spatial points object by number of points or minimum distance

Usage

```
thin_them(spdf, lim_dist = NA, lim_n = NA)
```

Arguments

spdf	SpatialPoints or SpatialPointsDataFrame object
lim_dist	Minimum tolerated distance between the points
lim_n	Number of points to be kept

Details

iteratively the closest point pairs are compared and the point with the lower overall distinctiveness is removed

Value

thinned SpatialPoints or SpatialPointsDataFrame object

Author(s)

Philipp Brun

References

Descombes, P., Chauvier, Y., Brun, P., Righetti, D., Wüest, R. O., Karger, D. N., ... & Zimmermann, N. E. (2022). Strategies for sampling pseudo-absences for species distribution models in complex mountainous terrain. bioRxiv, 2022-03.

`upsample_strategic` *Sample a supsample from a large set of points with a minimum distance constraint*

Description

Thin a spatial points object by number of points or minimum distance

Usage

```
upsample_strategic(spdf, lim_dist, n_tot, warnig = TRUE)
```

Arguments

`spdf`	SpatialPoints or SpatialPointsDataFrame object
`lim_dist`	Minimum tolerated distance between the points
`n_tot`	Minimum tolerated distance between the points

Details

iteratively samples points and rejects them if min thist to all existing points i the sample are not respected

Value

SpatialPoints or SpatialPointsDataFrame with at least lim_dist between all points

Author(s)

Philipp brun

References

Descombes, P., Chauvier, Y., Brun, P., Righetti, D., Wüest, R. O., Karger, D. N., ... & Zimmermann, N. E. (2022). Strategies for sampling pseudo-absences for species distribution models in complex mountainous terrain. bioRxiv, 2022-03.

upsample_thin *Sample a supsample from a large set of points with a minimum distance constraint*

Description

Thin a spatial points object by number of points or minimum distance

Usage

```
upsample_thin(spdf, lim_dist, n_tot)
```

Arguments

spdf	SpatialPoints or SpatialPointsDataFrame object
lim_dist	Minimum tolerated distance between the points
n_tot	Minimum tolerated distance between the points

Details

iteratively samples points and rejects them if min thist to all existing points i the sample are not respected

Value

SpatialPoints or SpatialPointsDataFrame with at least lim_dist between all points

Author(s)

Philipp Brun

References

Descombes, P., Chauvier, Y., Brun, P., Righetti, D., Wüest, R. O., Karger, D. N., ... & Zimmermann, N. E. (2022). Strategies for sampling pseudo-absences for species distribution models in complex mountainous terrain. bioRxiv, 2022-03.

var.imp *Assess predictor importance using random permutations*

Description

This function is based on the principle of permutation importance. This metric is applicable to any type of model. The basic idea is to consider a variable important if it has a positive effect on the prediction accuracy (classification), or MSE (regression). The risk of using this metric is a potential bias towards collinear predictive variables. It is important to note that this function is fully compatible with singular SDM output of the package.

Usage

```
var.imp(model, data, nperm, type, rescale)
```

Arguments

model	A model output. e.g. GLM, GAM, GBM, random forest, glmnet outputs...
data	A data.frame of environmental values.
nperm	How many random permutations should be applied per predictor?
type	Response type. Type of desired response.
rescale	Should the results be rescaled from a scale from 1 to 100?

Value

Average model importance score of each predictor.

Author(s)

Patrice Descombes

Examples

```
# Take anguilla data set from dismo package
data("Anguilla_train")
vrs = c("SegSumT","USRainDays","USSlope")

# Apply a simple GLM and measure each predictors' contribution to the model
form.glm = as.formula(paste("Angaus~",paste(paste0("poly(",vrs,",2)"),collapse="+")))
glm.calib = glm(form.glm,data=Anguilla_train,family="binomial")
imp.test = var.imp(model = glm.calib,
                   data = Anguilla_train[,vrs],
                   nperm = 10,
                   type = "response",
                   rescale = TRUE);imp.test

# Apply a glmnet model and measure each predictors' contribution to the model
```

---

wsl.ebc *Correct environmental bias of species observations via environmental clustering*

Description

Our wsl.ebc function corrects environmental bias from an observational dataset based on environmental stratification / clustering. A map of n clusters is generated based on input raster layers (in the article, climate predictors used in PPMs are employed as inputs). Following random equal-stratified sampling design (EBCe; Hirzel & Guisan, 2002), for each cluster, the number of observations per species (relative to all others) is artificially rescaled to the total number of observations found in the cluster presenting the highest observation density. Proportional-stratified sampling design (EBCp; Hirzel & Guisan, 2002) adds a second step: for each cluster, the number of observations per species

may additionally be multiplied by the cluster's area (i.e. proportion of pixels in percentage relative to the study area), or by its logarithm (default; consensus between EBCe and EBCp). Resulting output indicates a new number of observations per cluster and species, that the function automatically sub-samples with replacement over the original observational dataset. This function may be used for presences and absences distinctively.

Usage

```
wsl.ebc(
  obs = NULL,
  ras = NULL,
  pportional = TRUE,
  plog = TRUE,
  nclust = 50,
  sp.specific = TRUE,
  sp.cor = 0.5,
  keep.bias = TRUE,
  filter = FALSE,
  path = NULL,
  ...
)
```

Arguments

obs	Object of class matrix or data frame with three columns named sp.id (character), x (numeric) and y (numeric). More than one observation per species must be referenced.
ras	Object of class RasterBrick, RasterStack, list of RasterLayer of desired resolution and extent. Used to generate the map of clusters needed to summarize the environmental space of the study area.
pportional	Logical. Should environmental stratification of observations be proportional to the clusters' areas? If TRUE, EBCp applies.
plog	Logical. Should EBCp apply with a logarithm? If TRUE, a stratification consensus between EBCp and EBCe applies.
nclust	Number of chosen clusters. Default is 50.
sp.specific	Logical. Should EBC apply only for species whose environmental bias follows the overall one (i.e. the number of original species observations per cluster is correlated with that of the full dataset)?
sp.cor	If sp.specific = TRUE, spearman's correlation tests are by default set to 0.5; i.e. species with $r < 0.5$ are excluded from the function outputs.
keep.bias	Default is TRUE. Strongly recommended to use when sp.cor = TRUE. Per species, should the number of observations of the cluster in which the species was originally sampled most often, be preserved? Said differently, for each species, should the cluster with the most original species observations be as representative as the cluster with the most corrected observations? If TRUE, after EBC applies, the number of observations per species in their densest original cluster is set to that of the densest corrected cluster.
filter	Logical. Should the observations be filtered according to 'ras' resolution?
path	Path folder where the new species observation files should be saved.
...	Additional arguments passed on to the 'clara' function (package 'cluster')

Value

The function returns in 'path', one text file of corrected observations (presences or absences) per species. If the number of new EBC observations per species is too large, sampling those randomly without replacement before model calibrations is advised.

Author(s)

Yohann Chauvier

References

Chauvier, Y., Zimmermann, N. E., Poggiato, G., Bystrova, D., Brun, P., & Thuiller, W. (2021). Novel methods to correct for observer and sampling bias in presence-only species distribution models. Global Ecology and Biogeography, 30(11), 2312-2325.

`wsl.evaluate.pa` *Evaluate presence-absence models*

Description

Assess several model skill metrics for all models in a wsl.fit object. Currently AUC, RMSE, TSS, PPV, Accuracy, and Cohen's Kappa are evaluated. Furthermore, the threshold applied is returned.

Usage

```
wsl.evaluate.pa(
  x,
  tester = data.frame(),
  window = NULL,
  thres = numeric(),
  crit = "pp=op",
  prevalence_correction = FALSE,
  pres_only = FALSE,
  log_trans = TRUE,
  bias_cov = NULL
)
```

Arguments

`x`	A wsl.fit object
`tester`	Optional. A data.frame with testing data. Only mandatory if replicatetype='none' was chosen when models were fitted. Otherwise, used when evaluation against external dataset is needed. Must be a data.frame with as columns in order: "x", "y", "Presence" ('0' and '1'), "CV" (numeric: chosen cv-folds or just set '1' if test only vs one external dataset; if replicatetype='none' '1' is mandatory) and associated environmental values (same ones as for fitted models; i.e. same columns order and names). Note that categorical predictor values must be of class factor. NB: Here, model evaluation will only be initiated for the new testing data.
`window`	Only when 'wsl.ppmO' used'. Same object of class 'owin' used for models (in developments)

thres	Vector of the same length as the number of reps in model fit object. For wsl.flex model outputs, thresholds have to be labelled with the same names provided to models.
crit	Which threshold criterion should be considered? Currently 'pp=op' (predicted prevalence = observed prevalence), 'maxTSS' (threshold yielding maximum TSS), and 'external' (thresholds manually supplied) are possible
prevalence_correction	logical. Should imbalanced presence/absence data be upsampled to prevalence 0.5 for model evaluation.
pres_only	Logical. If TRUE, evaluation metrics of presence-absence models is applied to a wsl.fit object of presence-only models i.e. generated with wsl.ppmGlasso()
log_trans	Logical. Use only if pres_only=TRUE. Should predictions be converted to logarithm before evaluation? Prevent model evaluation errors.
bias_cov	A numerical vector whose length equal the number of environmental layers/columns. Only used when a bias covariate is implemented in the model calibration i.e. to fit species obs. with a potential spatial observer bias. Default is 1 for each variable, whereas designated bias covariate(s) (i.e. 0) will be reset everywhere to zero in order to evaluate corrected predictions

Value

An object of class 'wsl.evaluation'

Author(s)

Philipp Brun, Yohann Chauvier

Examples

```
# Take anguilla data set from dismo package
data("Anguilla_train")
vrs=c("SegSumT","USRainDays","USSlope")
env=Anguilla_train[,vrs]

### Check out wsl.gam
form.gam=as.formula(paste("Presence~",paste(paste0("s(",vrs,")"),collapse="+")))

# Try out wsl.gam funcion
modi2=wsl.gam(pa=Anguilla_train$Angaus,
              env_vars = env,
              taxon="Angaus",
              replicatetype="splitsample",
              reps=3,
              project="prototest",
              mod_tag="test-gam",
              formula=form.gam,
              family="binomial",
              step=FALSE)

# Try out custom summary function
summary(modi2)

# Access gam object of first replicate
```

```
summary(modi2@fits$replicate_01$`test-gam`)

# Evaluate the model
eval2=wsl.evaluate.pa(modi2,crit="maxTSS")

# Get evaluation summary
summary(eval2)
```

`wsl.evaluate.pres` *Evaluate presence-only models*

Description

Assess model skill metrics for presence-only models in a wsl.fit object. Currently Boyce index is evaluated. Furthermore, the threshold applied is returned.

Usage

```
wsl.evaluate.pres(
  x,
  tester = data.frame(),
  env_vars,
  window = NULL,
  thres = numeric(),
  pres_abs = FALSE,
  log_trans = TRUE,
  speedup = FALSE,
  env_samp = 5000,
  bias_cov = NULL,
  ...
)
```

Arguments

x	A wsl.ppm fit object
tester	Optional. A data.frame with testing data. Only mandatory if replicatetype='none' was chosen when models were fitted. Otherwise, used when evaluation against external dataset is needed. Must be a data.frame with as columns in order: "x", "y", "Presence" ('0'/'1' or only '1'), "CV" (numeric: chosen cv-folds or just set '1' if test only vs one external dataset; if replicatetype='none' '1' is mandatory) and associated environmental values (same ones as for fitted models; i.e. same columns order and names). Note that categorical predictor values must be of class factor. NB: Here, model evaluation will only be initiated for the new testing data.
env_vars	Same spatial layers used in the fitted model or an object of class 'data.frame' or 'matrix' defining a sample of the target layers by keeping same order for columns. If spatial layers are used, object of class 'RasterStack' or RasterBrick'.
thres	Vector of the same length as the number of reps in model fit object
pres_abs	Logical. If TRUE, evaluation metrics of presence-only models is applied to a wsl.fit object of presence-absence models

log_trans	Logical. Use only if pres_abs=FALSE. Should predictions be converted to logarithm before evaluation? Prevent model evaluation errors.
speedup	If env_vars is a 'RasterStack' or RasterBrick', should the boyce evaluation be faster? If TRUE, the algortihm uses a sample of the environmental layers
env_samp	If speedup=TRUE, how many environmental cells should be sampled with replacement? Default is 50'000 samples. The sample may be smaller than requested because of NAs
bias_cov	A numerical vector whose length equal the number of environmental layers/columns. Only used when a bias covariate is implemented in the PPM calibration i.e. to fit species observations with a potential spatial observer bias. Default is 1 for each variable, whereas designated bias covariate(s) (i.e. 0) will be reset everywhere to zero in order to evaluate corrected predictions
...	Additional arguments supplied to ecospat.boyce function (package 'ecospat')

Value

an object of class 'wsl.evaluation'. If in slot "performance" NA as thresholds are found, it indicates a lack of convergence in the model tested, and so, a biased/invalid threshold

Author(s)

Yohann Chauvier, Philipp Brun

Examples

```
# Load
data(AlpineConvention_lonlat)
data(exrst)
rst = rst[[1:6]]
data(xy_ppm)
mypoints = xy.ppm[,c("x","y")]

# Define mask
maskR = mask(rst[[1]],shp.lonlat)

# Run 'wsl.ppm.window' function
wind = wsl.ppm.window(mask = maskR,
                      val = 1,
                      owin = TRUE)

# nDefine quadrature points for 'wsl.ppmGlasso'
quadG1 = wsl.quadrature(mask = maskR,
                        area.win = wind,
                        random = FALSE,
                        lasso = TRUE,
                        env_vars = rst)

# Define your environments
envG = raster::extract(rst,mypoints)

# Spatial block cross-validation
to_b_xy = rbind(mypoints,quadG1@coords)
toSamp = c(rep(1,nrow(mypoints)),rep(0,nrow(quadG1@coords)))
```

```
block_cv_xy = make_blocks(nstrat = 5, df = to_b_xy, nclusters = 10, pres = toSamp)

# 'wsl.ppmGlasso' (alpha = 0.5 => Elastic net, see package 'glmnet')
   # Complex PPP lasso (poly = TRUE & lasso=TRUE)
ppm.lasso = wsl.ppmGlasso(pres = mypoints,
                        quadPoints = quadG1,
                        asurface = raster::area(shp.lonlat)/1000,
                        env_vars = envG,
                        taxon = "species_eg1",
                        replicatetype = "cv",
                        reps = 5,
                        strata = NA,
                        save=FALSE,
                        project = "lasso_eg1",
                        path = NA,
                        poly = TRUE,
                        lasso = TRUE,
                        alpha = 0.5,
                        type.measure = "mse",
                        standardize = TRUE,
                        nfolds = 5,
                        nlambda = 100)
summary(ppm.lasso)

  # Simple PPP (poly = FALSE & lasso=FALSE) + block-cross validation
ppm.simple = wsl.ppmGlasso(pres = mypoints,
                        quadPoints = quadG1,
                        asurface = raster::area(shp.lonlat)/1000,
                        env_vars = envG,
                        taxon = "species_eg2",
                        replicatetype = "block-cv",
                        reps = 5,
                        strata = block_cv_xy,
                        save = FALSE,
                        project = "lasso_eg2",
                        path = NA,
                        poly = FALSE,
                        lasso = FALSE)
summary(ppm.simple)

### Evaluation
eval1 = wsl.evaluate.pres(x = ppm.lasso,
                          env_vars = rst)
eval2 = wsl.evaluate.pres(x = ppm.simple,
                          env_vars = rst,
                          thres = 0.001,
                          speedup = TRUE,
                          bias_cov=c(1,0,0,0,1,1))
eval3 = wsl.evaluate.pa(x = ppm.lasso,
                        crit="maxTSS",
                        pres_only = TRUE,
                        bias_cov=c(1,0,0,1,1,0))
eval4 = wsl.evaluate.pa(x = ppm.simple,
                        crit="pp=op",
                        pres_only = TRUE)
summary(eval1)
summary(eval2)
```

```
summary(eval3)
summary(eval4)

### Thresholds
get_thres(eval1, mean = FALSE)
get_thres(eval2, mean = TRUE)
get_thres(eval3, mean = TRUE)
get_thres(eval4, mean = FALSE)
```

`wsl.evaluation-class` *An S4 class to store evaluation data*

Description

An S4 class to store evaluation data

Slots

`meta` A list with meta information

`thres` A vector with externally supplied thresholds

`performance` A list with model performance estimates

Author(s)

Philipp Brun

`wsl.fit-class` *An S4 class to store fitted objects*

Description

An S4 class to store fitted objects

Slots

`meta` A list with meta information

`tesdat` A list with held out data to be used for testing

`performance` fitted model objects

`call` the function call

Author(s)

Philipp Brun, Yohann Chauvier

`wsl.obs.filter` *Filter set of observations*

Description

Filter presences / absences through a chosen resolution grid.

Usage

```
wsl.obs.filter(o.xy, a.xy = NULL, grid)
```

Arguments

`o.xy`	Object of class 'matrix' or 'data frame' with two columns named "x" and "y". Must be used alone to apply grid filetring to one set of observations (presences or absences).
`a.xy`	Object of class 'matrix' or 'data frame' with two columns named "x" and "y". NULL by default. If used, "o.xy" becomes presences and "a.xy" absences. Resolution grid filtering is applied to both observation sets, but coordinates of "a.xy" falling within the same grid cells as "o.xy" are removed.
`grid`	Object of class 'RasterLayer', 'RasterBrick' or 'RasterStack' of desired resolution and extent.

Value

Object of class 'matrix' or 'data frame' with two columns named "x" and "y" comprising the new set of observations filtered at grid resolution. If "a.xy" is used the output is a list of two data.frame: filtered presences and filtered absences

Author(s)

Yohann Chauvier

Examples

```
### Load my binary observations species data

library(raster)

data(var_select_XYtest)
data(exrst)

### wsl.obs.filter(): example for the first species

   # Loading observations: presences and absences

presences = coordinates(mySP[[1]])[myPA[[1]] %in% "1",]
absences = coordinates(mySP[[1]])[myPA[[1]] %in% "0",]

   # Loading grid
```

```
r.layer = rst[[1]]

   # To filter observations by the grid only

pres.filtered = wsl.obs.filter(presences,grid=r.layer)
abs.filtered = wsl.obs.filter(absences,grid=r.layer)

   # To filter observations by the grid & remove abs. in cells where we also find pres.

PresAbs.filtered = wsl.obs.filter(presences,absences,r.layer)

   # Count presences (same filtering)

nrow(PresAbs.filtered[[1]])
nrow(pres.filtered)

   # Count Absences (filtering plus removal of duplicated absences)

nrow(abs.filtered)
nrow(PresAbs.filtered[[2]])

   # Visual

par(mfrow=c(1,2))
plot(presences)
plot(pres.filtered)

par(mfrow=c(1,2))
plot(absences)
plot(abs.filtered)
```

`wsl.ppm.env` *Poisson Point Process Models (PPPM) 'env'*

Description

Convert envrionmental data for 'wsl.ppmO' fitting.

Usage

```
wsl.ppm.env(env_vars, mask)
```

Arguments

`env_vars`	Object of class 'RasterBrick' or 'RasterStack'. Same extent and resolution as 'mask'
`mask`	Object of class 'RasterLayer' or 'RasterBrick' or 'RasterStack'. Mask is defined according to non NAs values and the resolution of the study

Value

List of objects of class 'im'

Author(s)

Yohann Chauvier

Examples

```
### Load

data(exrst)
data(xy_ppm)
mypoints = xy.ppm[,c("x","y")]

### Define mask

maskR = mask(rst[[1]],shp.lonlat)

### Define your environments

   # For 'wsl.ppmGlasso' (observations focus)
envG = raster::extract(rst,mypoints)

   # For 'wsl.ppmO' (study area focus)
envO = wsl.ppm.env(rst,maskR)
```

---

wsl.ppm.qscheme *To prepare PPM general quadratic environment with input species points*

Description

Not to be called directly by the user

Usage

```
wsl.ppm.qscheme(data, area.win, quads)
```

Author(s)

Yohann Chauvier

---

wsl.ppm.window *Poisson Point Process Models (PPPM) 'window'*

Description

Set up a study window of class 'owin' or 'im' necessary to run 'wsl.ppmO' fit function.

Usage

```
wsl.ppm.window(mask, val = 1, owin)
```

Arguments

mask	Object of class 'RasterLayer' or 'RasterBrick' or 'RasterStack'. Mask is defined according to non NAs values. Defined study area in pixels. Same as in 'wsl.quadrature'
val	Values that are assigned to the 'owin' or 'im' mask
owin	TRUE or FALSE

Value

Object of class 'owin' or 'im'

Author(s)

Yohann Chauvier

Examples

```
### Load

data(AlpineConvention_lonlat)
data(exrst)

### Define mask

maskR = mask(rst[[1]],shp.lonlat)

### Run function

wind = wsl.ppm.window(mask = maskR,
                      val = 1,
                      owin = TRUE)
```

---

wsl.predict.pa *Make predictions*

Description

Make predictions with all models from a wsl.fit object. If thresholds are supplied binary predictions are made if convert=TRUE, otherwise continuous predictions and separate description of thresholds are returned

Usage

```
wsl.predict.pa(
  x,
  predat = data.frame(),
  thres = numeric(),
  bias_cov = NULL,
  clust = FALSE
)
```

Arguments

x	An object of class wsl.fit
predat	Data.frame or raster for which predictions should be made
thres	Optional. Object of the same length as the number of replicates, or model types if a mean is applied accross model types. Obtained with 'get_thres'
bias_cov	A numerical vector whose length equal the number of environmental layers/columns. Only used when a bias covariate is implemented in calibrations i.e. to fit species observations with a potential spatial observer bias. Default is 1 for each variable, whereas designated bias covariate(s) (i.e. 0) will be reset everywhere to zero in order to evaluate corrected predictions
clust	Logical. If raster predictions are made, should the operation be run in parallel ?

Value

Object of class wsl.prediction with slots for meta info, and model predictions

Author(s)

Philipp Brun, Yohann Chauvier

Examples

```
# Take anguilla data set from dismo package
data("Anguilla_train")
vrs=c("SegSumT","USRainDays","USSlope")
env=Anguilla_train[,vrs]

### Check out wsl.glm
form.glm=as.formula(paste("Presence~",paste(paste0("poly(",vrs,",2)"),collapse="+")))

modi1=wsl.glm(pa=Anguilla_train$Angaus,
              env_vars = env,
              taxon="Angaus",
              replicatetype="cv",
              reps=5,
              project="prototest",
              mod_tag="test-glm",
              formula=form.glm,
              family="binomial",
              step=TRUE)

# Try out custom summary function
summary(modi1)

# Access glm object of first replicate
summary(modi1@fits$replicate_01$`test-glm`)

# Evaluate the model
eval1=wsl.evaluate.pa(modi1)

# Get evaluation summary
summary(eval1)
```

```
### Check out wsl.gam
form.gam=as.formula(paste("Presence~",paste(paste0("s(",vrs,")"),collapse="+")))

# Try out wsl.glm funcion
modi2=wsl.gam(pa=Anguilla_train$Angaus,
              env_vars = env,
              taxon="Angaus",
              replicatetype="splitsample",
              reps=3,
              project="prototest",
              mod_tag="test-gam",
              formula=form.gam,
              family="binomial",
              step=FALSE)

# Try out custom summary function
summary(modi2)

# Access glm object of first replicate
summary(modi2@fits$replicate_01$`test-gam`)

# Evaluate the model
eval2=wsl.evaluate.pa(modi2,crit="maxTSS")

# Get evaluation summary
summary(eval2)

### Check out wsl.gbm
form.gbm=as.formula(Presence ~ .)

# Try out wsl.glm funcion
modi3=wsl.gbm(pa=Anguilla_train$Angaus,
              env_vars = env,
              taxon="Angaus",
              replicatetype="none",
              reps=1,
              project="prototest",
              mod_tag="test-brt",
              formula= form.gbm,
              distribution = "bernoulli",
              interaction.depth = 1,
              shrinkage=.01,
              n.trees = 3500)

# Try out custom summary function
summary(modi3)

# Access glm object of first replicate
summary(modi3@fits$replicate_01$`test-brt`)

# Prepare external testing data
tste=data.frame(Presence=Anguilla_train$Angaus,CV=1,env)

# Evaluate the model
eval3=wsl.evaluate.pa(modi3,crit="maxTSS",tester=tste)

# Get evaluation summary
```

```
summary(eval3)

### Check out wsl.maxent
feat=c("linear=true","quadratic=true","hinge=true","product=true","threshold=false")

# Try out wsl.glm funcion
modi4=wsl.maxent(pa=Anguilla_train$Angaus,
                 env_vars = env,
                 taxon="Angaus",
                 replicatetype="block-cv",
                 reps=3,
                 strata=sample(1:3,nrow(env),replace=TRUE),
                 project="prototest",
                 mod_tag="test-mxe",
                 args=feat)

# Try out custom summary function
summary(modi4)

# Access glm object of first replicate
summary(modi4@fits$replicate_01$`test-mxe`)

# Define external threshold
thmxe=c(`test-mxe`=0.5)

# Evaluate the model
eval4=wsl.evaluate.pa(modi4,crit="external",thres=thmxe)

# Get evaluation summary
summary(eval4)

# Get thresholds
thr.4=get_thres(eval4, mean=FALSE)

### Check out wsl.flex
form.glm.2=as.formula(paste("Presence~",paste(vrs,collapse="+")))

modinp=list(multi("glm",list(formula=form.glm,family="binomial"),"glm-simple",step=TRUE,weight=TRUE),
   multi("gbm",list(formula=form.gbm,
   distribution = "bernoulli",
   interaction.depth = 1,
   shrinkage=.01,
   n.trees = 3500),"gbm-simple"),
  multi("gam",list(formula=form.gam,family="binomial"),"gam-simple",step=FALSE,weight=TRUE),
   multi("maxent",list(args=feat),"mxe-simple"),
   multi("randomForest",list(formula=form.gbm,ntree=500,maxnodes=NULL),"waud1"),
  multi("glm",list(formula=form.glm.2,family="binomial"),"glm-lin",step=TRUE,weight=TRUE))

# Try out wsl.glm funcion
modi5=wsl.flex(pa=Anguilla_train$Angaus,
               env_vars = env,
               taxon="Angaus",
               replicatetype="block-cv",
               reps=3,
               strata=sample(1:3,nrow(env),replace=TRUE),
               project="multitest",
               mod_args=modinp)
```

```
# Try out custom summary function
summary(modi5)

# Access glm object of first replicate
summary(modi5@fits$replicate_01$`glm-simple`)

# Evaluate the model
eval5<-wsl.evaluate.pa(modi5,crit="pp=op")

# Get evaluation summary
summary(eval5)

# Get thresholds
thr.5=get_thres(eval5, mean=FALE)

### Make some predictions
pred4=wsl.predict.pa(modi4,predat=env)
pred5=wsl.predict.pa(modi5,predat=env,thres=thr.5)
```

`wsl.predict.pres` *Make predictions*

Description

Make predictions with all models from a wsl.fit object. If thresholds are supplied binary predictions are made, otherwise continuous predictions are returned.

Usage

```
wsl.predict.pres(
  x,
  thres = numeric(),
  predat = list(),
  window = NULL,
  log_trans = TRUE,
  raster = FALSE,
  bias_cov = NULL
)
```

Arguments

x	An object of class wsl.fit
thres	Optional. Object of the same length as the number of replicates, or model types if a mean is applied accross model types. Set a custom threshold.
predat	Same spatial layers used in the fitted model or an object of class 'data.frame' or 'matrix' defining a sample of the target layers by keeping same order for columns. If spatial layers, when wsl.ppmGlasso' used, object of class 'Raster-Stack' or RasterBrick'.
log_trans	Logical. Should the predictions be converted to logarithm before converting to binary ? Should be TRUE if log.trans was TRUE when using wsl.evaluate

raster	Logical. Should the output be a list of rasters or matrix ?
bias_cov	A numerical vector whose length equal the number of environmental layers/columns. Only used when a bias covariate is implemented in the PPM calibration i.e. to fit species observations with a potential spatial observer bias. Default is 1 for each variable, whereas designated bias covariate(s) (i.e. 0) will be reset everywhere to zero in order to evaluate corrected predictions

Value

Object of class wsl.prediction with slots for meta info, and model predictions

Author(s)

Yohann Chauvier, Philipp Brun

Examples

```
# Load
data(AlpineConvention_lonlat)
data(exrst)
rst = rst[[1:6]]
data(xy_ppm)
mypoints = xy.ppm[,c("x","y")]

# Define mask
maskR = mask(rst[[1]],shp.lonlat)

# Run 'wsl.ppm.window' function
wind = wsl.ppm.window(mask = maskR,
                      val = 1,
                      owin = TRUE)

# nDefine quadrature points for 'wsl.ppmGlasso'
quadG1 = wsl.quadrature(mask = maskR,
                        area.win = wind,
                        random = FALSE,
                        lasso = TRUE,
                        env_vars = rst)

# Define your environments
envG = raster::extract(rst,mypoints)

# Spatial block cross-validation
to_b_xy = rbind(mypoints,quadG1@coords)
toSamp = c(rep(1,nrow(mypoints)),rep(0,nrow(quadG1@coords)))
block_cv_xy = make_blocks(nstrat = 5, df = to_b_xy, nclusters = 10, pres = toSamp)

# Environmental block cross-validation
to_b_env = rbind(envG,quadG1@Qenv[,-1])
block_cv_env = make_blocks(nstrat = 5, df = to_b_env, nclusters = 10, pres = toSamp)

# 'wsl.ppmGlasso' (alpha = 0.5 => Elastic net, see package 'glmnet')
   # Complex PPP lasso (poly = TRUE & lasso=TRUE)
ppm.lasso = wsl.ppmGlasso(pres = mypoints,
                          quadPoints = quadG1,
```

```
                        asurface = raster::area(shp.lonlat)/1000,
                        env_vars = envG,
                        taxon = "species_eg1",
                        replicatetype = "cv",
                        reps = 5,
                        strata = NA,
                        save=FALSE,
                        project = "lasso_eg1",
                        path = NA,
                        poly = TRUE,
                        lasso = TRUE,
                        alpha = 0.5,
                        type.measure = "mse",
                        standardize = TRUE,
                        nfolds = 5,
                        nlambda = 100)
summary(ppm.lasso)

  # Simple PPP (poly = FALSE & lasso=FALSE) + block-cross validation
ppm.simple = wsl.ppmGlasso(pres = mypoints,
                        quadPoints = quadG1,
                        asurface = raster::area(shp.lonlat)/1000,
                        env_vars = envG,
                        taxon = "species_eg2",
                        replicatetype = "block-cv",
                        reps = 5,
                        strata = block_cv_xy,
                        save = FALSE,
                        project = "lasso_eg2",
                        path = NA,
                        poly = FALSE,
                        lasso = FALSE)
summary(ppm.simple)

### Evaluation
eval1 = wsl.evaluate.pres(x = ppm.lasso,
                          env_vars = rst)
eval2 = wsl.evaluate.pres(x = ppm.simple,
                          env_vars = rst,
                          thres = 0.001,
                          speedup = TRUE,
                          bias_cov=c(1,0,0,0,1,1))
eval3 = wsl.evaluate.pa(x = ppm.lasso,
                        crit="maxTSS",
                        pres_only = TRUE,
                        bias_cov=c(1,0,0,1,1,0))
eval4 = wsl.evaluate.pa(x = ppm.simple,
                        crit="pp=op",
                        pres_only = TRUE)
summary(eval1)
summary(eval2)
summary(eval3)
summary(eval4)

### Thresholds
get_thres(eval1, mean = FALSE)
get_thres(eval2, mean = TRUE)
```

```
get_thres(eval3, mean = TRUE)
get_thres(eval4, mean = FALSE)

### Predictions
pred1 = wsl.predict.pres(x = ppm.lasso,
                         predat = rst,
                         thres = get_thres(eval1,mean=FALSE),
                         raster = TRUE)
par(mfrow=c(2,3))
sapply(1:5,function(x) plot(pred1@predictions[[x]][[1]]))

pred2 = wsl.predict.pres(x = ppm.simple,
                         predat = rst,
                         raster = TRUE,
                         bias_cov=c(1,0,0,0,1,1))
par(mfrow=c(2,3))
sapply(1:5,function(x) plot(pred2@predictions[[x]][[1]]))

pred3 = wsl.predict.pres(x = ppm.lasso,
                         predat = rst,
                         thres = get_thres(eval1,mean=TRUE),
                         raster = TRUE,
                         bias_cov=c(1,0,0,1,1,0))
par(mfrow=c(2,3))
sapply(1:5,function(x) plot(pred3@predictions[[x]][[1]]))
pred4 = wsl.predict.pres(x = ppm.simple,
                         predat = rst,
                         raster = FALSE)
```

`wsl.predictGlasso` *SDM predict function for glmnet outputs*

Description

This function can make predictions based on any type of glmnet models, i.e. even complex model calibrated with polynomial terms. Additionally, the predict function avoids errors when pedictions are made based on predictors whose values equal all one single value (very useful to apply bias covariate correction).

Usage

```
wsl.predictGlasso(mod, predat)
```

Arguments

`mod`	A glmnet model output.
`predat`	Same spatial layers used in the fitted model or an object of class 'data.frame' or matrix' defining a sample of the target layers by keeping same order for columns. If spatial layers, when 'wsl.ppmGlasso' used, object of class 'RasterStack' or RasterBrick'.

Value

RasterLayer or Vector.

Author(s)

Yohann Chauvier

`wsl.prediction-class` *An S4 class to store prediction data*

Description

An S4 class to store prediction data

Slots

`meta` A list with meta information

`thres` A vector with externally supplied thresholds

`predictions` A list with model predictions

Author(s)

Philipp Brun

`wsl.pseudoabsences-class`
An S4 class to store pseudoabsence data

Description

An S4 class to store pseudoabsence data

Slots

`meta` A list with meta information

`type` The type of pseudoabsence sampling

`pa` a vector with presences and pseudo absences

`env_vars` data.frame with environmental predictors

`xy` a matrix with coordinates of the points

Author(s)

Philipp Brun

wsl.quadrature *Poisson Point Process Models (PPPM) 'quadrature'*

Description

Set up quadrature points (or "background points") necessary to run 'wsl.ppmO' and 'wsl.ppmGlasso' fit functions. Those points apply a spatial scaling proportional to the study area and estimate the maximised model log likelihood (see Renner 2013, Renner et al. 2015)

Usage

```
wsl.quadrature(
  mask,
  area.win,
  random = FALSE,
  nQ = 1e+05,
  lasso = TRUE,
  env_vars = NULL
)
```

Arguments

mask	Object of class 'RasterLayer' or 'RasterBrick' or 'RasterStack'. Mask is defined according to non NAs values. Defines study area in pixels used to grid sample the quadrature points. Sampling is done over raster centroids so the resolution of the mask defines the desired sampling.
area.win	Object of class 'owin'. 'owin' output of 'wsl.ppm.window'
random	Logical. Should quadrature points be generated randomly or according to a regular mask ?
nQ	To choose the number of random quadrature points in case 'random=TRUE'
lasso	Logical. Form of the output. If TRUE, 'wsl.ppmGlasso' is used
env_vars	Only when 'lasso=TRUE'. Object of class 'RasterBrick' or 'RasterStack'. Set of predictors the user wants to use to fit the model

Value

Object of class 'ppp' or 'wsl.quads'. Points associated to NAs env. values are removed

Author(s)

Yohann Chauvier

Examples

```
#### Load

data(AlpineConvention_lonlat)
data(exrst)

### Define mask
```

```
maskR = mask(rst[[1]],shp.lonlat)

### Run 'wsl.ppm.window' function

wind = wsl.ppm.window(mask = maskR,
                      val = 1,
                      owin = TRUE)

### Define quadrature points for 'wsl.ppmGlasso'

   # Grid regular
quadG1 = wsl.quadrature(mask = maskR,
                        area.win = wind,
                        random = FALSE,
                        lasso = TRUE,
                        env_vars = rst)

   # Randomly
quadG2 = wsl.quadrature(mask = maskR,
                        area.win = wind,
                        random = TRUE,
                        nQ = 100000,
                        lasso = TRUE,
                        env_vars = rst)

### Define quadrature points for 'wsl.ppmO'

   # Grid regular
quadO1 = wsl.quadrature(mask = maskR,
                        area.win = wind,
                        random = FALSE,
                        lasso = FALSE,
                        env_vars = NULL)

   # Randomly
quadO2 = wsl.quadrature(mask = maskR,
                        area.win = wind,
                        random = TRUE,
                        nQ = 100000,
                        lasso = FALSE,
                        env_vars = NULL)
```

`wsl.quads-class` *An S4 class to store quadrature objects*

Description

An S4 class to store quadrature objects

Slots

`coordinates` informations on the generated quadratures

`predictors` values extracted for each quadrature

Author(s)

Yohann Chauvier, Philipp Brun

`wsl.samplePseuAbs` *Sample pseudoabsences using various strategies*

Description

Flexlible function to sample pseudoabsences with various strategies and store the results in a 'wsl.pseudoabsences' object that can be passed on in the 'wsl.biodiv' pipeline.

Usage

```
wsl.samplePseuAbs(
  n = 10000,
  env.stack,
  type = "geographic",
  add.strat = 0,
  pres = numeric(),
  taxon = character(),
  geodist_fact = 1,
  geores_fact = 20,
  template_dir = NA,
  geo_nrep = 7,
  target.group_dir = NA,
  env.strat_path = NA,
  rAll = TRUE,
  force_spat_thin = "no",
  limdist = NA,
  set_max_npres_to_nabs = TRUE
)
```

Arguments

`n`	number of pseudoabsence points desired. Default is 10000.
`env.stack`	RasterStack with environmental layers for sampling and extraction
`type`	Desired sampling strategy. Options are 'geographic', 'density', 'random', 'target.group', 'geo.strat', 'env.strat' and 'env.semi.strat' (see details). Default is 'geographic'
`add.strat`	Fraction between 0 and 1; should strategy be complemented by a fraction of environmental strata.
`pres`	SpatialPoints object. Location of presence points. Necessary for 'geographic' sampling strategy and the best adding point if the downstream functions will be used.
`taxon`	Character; name of taxon of interest to keep track of meta information.
`geodist_fact`	Factor to adjust spatial autocorrelation lengths: for 'geographic' pseudoabsence point patterns, values below 1 increase autocorrelation length; values above 1 decrease it; for 'density' sampling it is the other way around.

`georesfact` Aggregation factor 'geographic' template. Larger values save calculation time, but decrease resolution of sampling points.

`templatedir` Directory where template raster for 'geographic' sampling should be saved in/loaded from. If NA, nothing will be saved; if provided, template will be saved in/loaded from directory depending on whether a file already exists.

`geonrep` number of replicates of geographic models. More will create a smoother pattern but increase computation time.

`target.groupdir` Directory where xy files of target group taxa are stored. Must be supplied if sampling stragegy is 'target.group', must contain a column names 'x' and 'y' with coordinates in the same projection as other spatial data

`env.stratpath` Directory where sample of environmental strata for 'env.strat' sampling should be saved in/loaded from. If NA, nothing will be saved; if provided, environmental strata will be saved in/loaded from directory depending on whether a file already exists.

`rAll` should all data be read into memory for computation of environmental strata? this is faster but you may run into memory issues for large rasters.

`forcespatthin` Should minimum distance be enforced between points? Options are 'no', 'presences', 'absences', and 'both'. By default thinning is defined for pseudoabsences from 'geographic', 'density', 'random', and 'geo.strat' methods with minimum distance according to the resolution of the template raster. 'presences' takes the minimum distance criterion from the template raster over to the 'presence' points; 'absences' takes the criterion over to 'env.strat', 'env.semi.strat', and 'target.group'; 'both' does it for both.

`limdist` The minimum distance accepted for spatial thinning. Units should be km if the spatial data is projected, otherwise the units of the coordinate reference system used. If no value is supplied, the maximum distance between two cells of the template raster will be taken.

`setmaxnpresto_nabs` logical. Should the maximum number of presences be equal to the number of pseudoabsences defined.

Details

'geographic' samples pseudoabsences with a sampling probability. inversely proportional to the geographic distance to presence observations. 'density' samples pseudoabsences proportional to the density of presence observations. 'random' samples pseudoabsences randomly with a sampling probability proportional the area of the cells. 'target.group' samples pseudoabsences from the presences of the taxa of the target group, attempting to correct for sampling bias. It depends on a directory with taxa defined by the user as target group. 'geo.strat' samples pseudoabsences geographically stratified either on a plane, or on a sphere depending on the projection of the supplied env.stack. 'env.strat' samples pseudoabsences environmentally stratified. Points are sampled form all realized combinations of environmental conditions occuring in the environmental stack that have a minimal occurrence frequency. By default environmental strata are calculated based on all raster layers supplied. If a directory is supplied as 'env.strat_path', a large sample of stratified points will be saved to speed up computations for follow-up species. If environmental strata based on different predictors than supplied are preferred 'env.strat_path'can be an .RData file from a previous sampling of strata form different environmental predictors. 'env.semi.strat' is similar to 'env.strat' but samples environmental strata proportional to the logarithm of the area they cover.

Value

an object of class 'wsl.pseudoabsences' that can be plotted and passed on to wsl.flex

Author(s)

Philipp Brun

References

Descombes, P., Chauvier, Y., Brun, P., Righetti, D., Wüest, R. O., Karger, D. N., ... & Zimmermann, N. E. (2022). Strategies for sampling pseudo-absences for species distribution models in complex mountainous terrain. bioRxiv, 2022-03.

Examples

```
### ============================================================================
### Data preparation
### ============================================================================

# Predictors
bio=getData('worldclim',var='bio',lon=16, lat=48,res=.5)
bio=bio[[c(1,4,12)]]

# install.packages("rgbif")
library(rgbif)
# extract species
spn='Boletus aestivalis'
xt=as.vector(extent(bio))
baest <- occ_search(scientificName=spn,
                  hasCoordinate=TRUE,
                  decimalLongitude=paste0(xt[1],",",xt[3]),
                  decimalLatitude=paste0(xt[2],",",xt[4]))

pbaest=baest$data[,c('decimalLongitude','decimalLatitude')]
baest_spp=SpatialPoints(pbaest,proj4string = crs(bio))

# extract target group
targr <- occ_search(familyKey = 8789,
                   hasCoordinate=TRUE,
                   limit = 10000,
                   decimalLongitude=paste0(xt[1],",",xt[3]),
                   decimalLatitude=paste0(xt[2],",",xt[4]))

ptargr=as.matrix(targr$data[,c('decimalLongitude','decimalLatitude')])
colnames(ptargr)=c("x","y")

# create temporary directory for target.group info
tdir=paste0(tempdir(),"/trgr")
dir.create(tdir)
write.table(ptargr,file=paste0(tdir,"/targetxy.txt"),row.names = F)

# create temporary directory for template raster and env strata
strdir=paste0(tempdir(),"/str")
dir.create(strdir)

# Note that for these should not be temporary files for a real analysis.
```

```
### ============================================================================
### Sample pseudoabsences
### ============================================================================

# Geograhpic method with 20% env strata
pseu.abs1=wsl.samplePseuAbs(type="geographic",
                            n=5000,
                            env.stack=bio,
                            pres=baest_spp,
                            add.strat=0.2,
                            template_dir=strdir,
                            env.strat_path=strdir,
                            geodist_fact=1,
                            geores_fact=3,
                            geo_nrep=7,
                            taxon=spn)

plot(pseu.abs1)

# Only geographic with longer autocorrelation length
pseu.abs2=wsl.samplePseuAbs(type="geographic",
                             n=5000,
                             env.stack=bio,
                             pres=baest_spp,
                             add.strat=0,
                             template_dir=strdir,
                             env.strat_path=strdir,
                             geodist_fact=.5,
                             geores_fact=3,
                             geo_nrep=7,
                             taxon=spn)

plot(pseu.abs2)

# Random and thin presences
pseu.abs3=wsl.samplePseuAbs(type="random",
                            n=5000,
                            env.stack=bio,
                            template_dir=strdir,
                            pres=baest_spp,
                            geores_fact=3,
                            add.strat=0,
                            taxon=spn,
                            force_spat_thin="presences")

plot(pseu.abs3)

# Geo.start
pseu.abs4=wsl.samplePseuAbs(type="geo.strat",
                            n=5000,
                            env.stack=bio,
                            template_dir=strdir,
                            pres=baest_spp,
                            geores_fact=3,
                            add.strat=0,
                            taxon=spn)
```

```
plot(pseu.abs4)

# Target group with 20% env strat
pseu.abs5=wsl.samplePseuAbs(type="target.group",
                          n=5000,
                          env.stack=bio,
                          template_dir=strdir,
                          target.group_dir=tdir,
                          env.strat_path=strdir,
                          geores_fact=3,
                          pres=baest_spp,
                          add.strat=0.2,
                          taxon=spn,
                          force_spat_thin="both")

plot(pseu.abs5)

# Environmental semi-stratified
pseu_abs6=wsl.samplePseuAbs(n = 5000,
                            env.stack=bio,
                            type = "env.semi.strat",
                            add.strat = 0,
                            pres = baest_spp,
                            taxon = spn,
                            template_dir=strdir,
                            env.strat_path=strdir)

plot(pseu_abs6)

# Environmental semi-stratified with min dist
pseu_abs7=wsl.samplePseuAbs(n = 5000,
                            env.stack=bio,
                            type = "env.semi.strat",
                            add.strat = 0,
                            geores_fact=3,
                            pres = baest_spp,
                            taxon = spn,
                            template_dir=strdir,
                            env.strat_path=strdir)

plot(pseu_abs7)

# Density dependent
pseu_abs8=wsl.samplePseuAbs(n = 5000,
                            env.stack=bio,
                            type = "density",
                            add.strat = 0,
                            pres = baest_spp,
                            taxon = spn,
                            geores_fact=3,
                            template_dir=strdir,
                            env.strat_path=strdir)

plot(pseu_abs8)

### ============================================================================
```

```
### Fit SDMs
### ============================================================================

# Define model settings
vrs=names(bio)
form.glm=as.formula(paste("Presence~",paste(paste0("poly(",vrs,",2)"),collapse="+")))
form.gam=as.formula(paste("Presence~",paste(paste0("s(",vrs,")"),collapse="+")))
form.tree=as.formula(Presence ~ .)

modinp=list(multi("glm",list(formula=form.glm,family="binomial"),"glm-simple",step=FALSE),
           multi("gbm",list(formula=form.tree,
                distribution = "bernoulli",
                interaction.depth = 1,
                shrinkage=.01,
                n.trees = 3500),"gbm-simple"),
          multi("gam",list(formula=form.gam,family="binomial"),"gam-simple",step=FALSE),
          multi("randomForest",list(formula=form.tree,ntree=500,maxnodes=NULL),"waud1"))

# Fit models using wsl.flex function
library(gam)
library(gbm)
library(randomForest)
modi5=wsl.flex(x=pseu.abs5,
               replicatetype="block-cv",
               reps=3,
               strata=sample(1:3,nrow(pseu.abs5@env_vars),replace=TRUE),
               project="multitest",
               mod_args=modinp)

### ============================================================================
### Evaluate
### ============================================================================

# Evaluate the models with wsl.evaluate function
eval5<-wsl.evaluate(modi5,crit="maxTSS",prevalence_correction = TRUE)
summary(eval5)

### ============================================================================
### Predict
### ============================================================================

modi_pred=wsl.flex(x=pseu.abs5,
                   replicatetype="none",
                   reps=1,
                   project="test_pred",
                   mod_args=modinp[c(1:2)])

# Get thresholds
thr.5=get_thres(eval5)[1:2]

### Make some predictions
pred5=wsl.predict(modi_pred,predat=bio,thres = thr.5)

par(mfrow=c(1,2))
plot(pred5@predictions[[1]]$`glm-simple`,main="GLM")
plot(pred5@predictions[[1]]$`gbm-simple`,main="GBM")
```

`wsl.test.quads` *Poisson Point Process Models (PPPM) 'quadrature' for presence-absence evaluation metrics*

Description

Sample independent test quadrature points over the study area used for evaluating presence-only models with presence-absence metrics (i.e. AUC, TSS, Kappa, PPV...). Presence-only models can be evaluated with other metrics than Boyce index by using random sampling points as absences. But these absences validation data need to be generated independenly over the study area i.e. by avoiding resampling quadrature points already used in the models.

Usage

```
wsl.test.quads(mask, quadrature, env_vars, nQ = 10000, replace = TRUE)
```

Arguments

`mask`	Object of class 'RasterLayer' or 'RasterBrick' or 'RasterStack' of. Mask is defined according to non NAs values. Defines study area in pixels used to grid sample the quadrature points. Sampling is done over raster centroids so the resolution of the mask defines the desired sampling. Same as in 'wsl.quadrature'
`quadrature`	Object of class 'wsl.quads' or 'ppp'. May be one object or a list of object. Same as in wsl.ppm.fit
`env_vars`	An object of class 'RasterStack' or 'RasterBrick'. Same spatial layers used in the models. Use to extract env values from test quadrature points
`nQ`	Mumber of random quadrature points to sample
`replace`	Logical. If TRUE, quadrature sampling is done with replacements

Value

Object of class 'wsl.quads' or 'list'. Points associated to NAs env. values are removed

Author(s)

Yohann Chauvier

Examples

```
#### Load

data(AlpineConvention_lonlat)
data(exrst)

### Define mask

maskR = mask(rst[[1]],shp.lonlat)

### Define quadrature points for 'wsl.ppmGlasso'
```

```
quadG2 = wsl.quadrature(mask = maskR,
                        area.win = wind,
                        random = TRUE,
                        nQ = 10000,
                        lasso = TRUE,
                        env_vars = rst)

### Define quadrature points for 'wsl.ppmO'

quadO2 = wsl.quadrature(mask = maskR,
                        area.win = wind,
                        random = TRUE,
                        nQ = 10000,
                        lasso = FALSE,
                        env_vars = NULL)

### Define quadrature test points for 'wsl.ppmGlasso'

test.quadG2=wsl.quads.test(mask=maskR,
                           quadrature=quadG2,
                           nQ=1000,
                           replace=TRUE)


### Define quadrature test points for 'wsl.ppmO'

test.quadG2=wsl.quads.test(mask=maskR,
                           quadrature=quadO2,
                           nQ=1000,
                           replace=TRUE)
```

`wsl.varKeep` *Pre-select set of predictors*

Description

Rank spatial predictors "ras" in descending order following their mean predictive power found with wsl.varPPower(), and apply descending successive correlations tests to obtain a pre-selection of predictors. Additionally a Variance Inflation Factor Test (VIF) may also be applied to predictors which succeeded the descending tests to identify the ones that show high mutlicollinearity.

Usage

```
wsl.varKeep(
  PPower.object,
  ras,
  corTEST = 0.7,
  vifTEST = FALSE,
  corVIF = 0.7,
  ...
)
```

Arguments

PPower.object	'wsl.varPPower' object
ras	Object of class 'RasterLayer' or 'list' of 'RasterLayer'. Same as in wsl.varPPower(). Layers must be of same resolution i.e. excatly the same number of cells
corTEST	Numeric. Threshold of accepted correlation applied in the descending tests
vifTEST	Logical. If TRUE, a VIF test is applied to the set of predictors that succeeded descending correlation tests
corVIF	Numeric. Threshold of accepted correlation within VIF if "vifTEST=TRUE"
...	Arguments passed on to the cor() function used for "corTEST"

Value

Object of class list with three elements: 1) ranking of non correlated predictors, 2) ranking of leftovers correlated predictors, 3) vif test results

Author(s)

Yohann Chauvier

Examples

```
### Data Preparation

# Load environmental rasters and assign random raster Class
data(exrst)
rCLASS = c(rep("Pollution",4),rep("Climate",4),rep("LandCover",4))

# Create a list of rasters out of the rasterBRICK
rasterL = unstack(rst)

# Load my binary observations species data
data(var_select_XYtest.RData)

### wsl.varPPower(): example with a data.frame & rasters not in classes

PPower.DF = wsl.varPPower(points=sp.DF[,c("x","y")],
                          val = sp.DF$myPA,
                          species = sp.DF$spCODES,
                          ras = rasterL,
                          rasCLASS = NULL,
                          mlinear = FALSE,
                          glmMODE = "binomial",
                          weight = sp.DF$myWEIGHT,
                          poly = TRUE,
                          polyBRUT = TRUE,
                          parallel = FALSE,
                          cores = NULL)


### wsl.varPPower(): example with SpatialPointsDataFrame & rasters in classes

PPower.DF = wsl.varPPower(points=sp.DF[,c("x","y")],
                          val = sp.DF$myPA,
                          species = sp.DF$spCODES,
```

```
                            ras = rasterL,
                            rasCLASS = NULL,
                            mlinear = FALSE,
                            glmMODE = "binomial",
                            weight = sp.DF$myWEIGHT,
                            poly = TRUE,
                            polyBRUT = TRUE,
                            parallel = FALSE,
                            cores = NULL)

### wsl.varPPower(): example with SpatialPointsDataFrame & rasters in classes

# Checking length of every species input
c(length(mySP),length(myPA),length(spCODES),length(myWEIGHT))

# Running the function
PPower.SPDF = wsl.varPPower(points = mySP,
                            val = myPA,
                            species = spCODES,
                            ras = rasterL,
                            rasCLASS = rCLASS,
                            mlinear = FALSE,
                            glmMODE = "binomial",
                            weight = myWEIGHT,
                            poly = TRUE,
                            polyBRUT = TRUE,
                            parallel = FALSE,
                            cores = NULL)


 ### wsl.varKeep() with a descending correlation of 0.7 with a VIF test at 0.7

# Example with no raster classes
KeepVar = wsl.varKeep(PPower.object = PPower.DF,
                      ras = rasterL,
                      corTEST = 0.7,
                      vifTEST = TRUE,
                      corVIF = 0.7,
                      use="complete.obs")

#Example with raster classes
KeepVar = wsl.varKeep(PPower.object = PPower.SPDF,
                      ras = rasterL,
                      corTEST = 0.7,
                      vifTEST = TRUE,
                      corVIF = 0.7,
                      use = "complete.obs")
```

`wsl.varPPower` *Predictive power assessments of predictors*

Description

Evaluate the predictive power of single spatial predictors for fitting spatial observations or continous values. Predictors may be classify in classes to separate the ouputs, and data to explain

(Y input) may be binary (e.g. presences/absences), discrete (e.g. diversity count) or continuous. Fix and random effects may be used with generalized linear model (GLM) and simple linear model (LM). Therefore, the function currently implement D2.adj::glm(), R2.adj::lm(), pseudo-R2::glmer() and pseudo-R2::lmer(). D2 is calculated via the ecospat package and pseudo-R2 with the MuMIn package for mixed models. Quadratic terms may also be used for more flexible fits, and a parallel argument is available if processing the data is too time consuming.

Usage

```
wsl.varPPower(
  points,
  val,
  species = NULL,
  ras,
  rasCLASS = NULL,
  mlinear = FALSE,
  mixEffect = FALSE,
  mixCat = NULL,
  glmMODE = "binomial",
  weight = 1,
  poly = FALSE,
  polyBRUT = FALSE,
  parallel = FALSE,
  parINFOS = NULL,
  cores = detectCores()/2,
  ...
)
```

Arguments

points	Numeric. Object of class 'matrix' or 'data frame' with two columns named "x" and "y", or 'SpatialPoints', or 'list' of 'SpatialPoints'. "points" should be of same length as "val", "species" and "weight"
val	Numeric. Object of class 'vector' if "points" is of class 'matrix' or 'data.frame', otherwise object of class 'list'. Binary observations if "glmMODE="binomial", discrete if "glmMODE="poisson", continious if "mlinear=TRUE". Must be of same length as "points", "species" and "weight"
species	Character. Object of class 'vector' specifying the ID of your observations. No mandatory use except when "points" is a 'list' of 'SpatialPoints'. If used, must be of same length as "points", "val" and "weight"
ras	Object of class 'RasterLayer' or 'list' of 'RasterLayer'. Layers must be of exact same resolution and extent i.e. exactly the same number of cells
rasCLASS	Object of class 'vector' to associate IDs to the raster(s) if "ras" is a 'list' of 'RasterLayer'
mlinear	'TRUE' run lm(), 'FALSE' run glm()
mixEffect	Default is FALSE. If 'TRUE' run mLinear with random effect (i.e. mix LM or GLM)
mixCat	Character. Object of class 'vector' if "points" is of class 'matrix' or 'data.frame', otherwise object of class 'list'. Use to assign categories to your observations. Those are used for potential random effects. Must be of same length as "points", "val" and "species"

glmMODE	glm() link function if "mlinear=FALSE" (defaut::"binomial" or "poisson")
weight	Numeric. Object of class 'vector' if "points" is of class 'matrix' or 'data.frame', otherwise object of class 'list'. Used for weighting your observations when running glm() or lm(). Must be of same length as "points", "val" and "species"
poly	If TRUE, glm() or lm() uses a polynomial quadratic transformation on "ras"
polyBRUT	If TRUE, force "poly" parameter in cases NAs are found in "ras"
parallel	If TRUE, parallelisation is active
parINFOS	Path. Create a txt file to write parallelisation infos if "parallel=TRUE".
cores	Number of cores if "parallel=TRUE"
...	Arguments passed on to the lm() or glm() function

Value

'wsl.varPPower' object with n slots corresponding to n "rasCLASS". In each slots: matrix of nrow::c("species" + mean + standard deviation) & ncol::"ras". Where NAs occur, models could not correctly converged.

Author(s)

Yohann Chauvier

Examples

```
### Data Preparation

rm(list = setdiff(ls(), lsf.str()))

# Load environmental rasters and assign random raster Class
data(exrst)
rCLASS = c(rep("Pollution",4),rep("Climate",4),rep("LandCover",4))

# Create a list of rasters out of the rasterBRICK
rasterL = unstack(rst)

# Load my binary observations species data
data(var_select_XYtest)

# Create a category vector
mixV = sample(LETTERS[1:3], 35743, replace=TRUE)

### wsl.varPPower(): example with a data.frame & rasters not in classes

PPower.DF = wsl.varPPower(points=sp.DF[,c("x","y")],
                      val = sp.DF$myPA,
                      species = sp.DF$spCODES,
                      ras = rasterL,
                      rasCLASS = NULL,
                      mlinear = FALSE,
                      glmMODE = "binomial",
                      weight = sp.DF$myWEIGHT,
                      poly = TRUE,
                      polyBRUT = TRUE,
                      parallel = FALSE,
                      cores = NULL)
```

```
### wsl.varPPower(): Same example including random effect
PPower.DF = wsl.varPPower(points = sp.DF[,c("x","y")],
                      val = sp.DF$myPA,
                      species = sp.DF$spCODES,
                      ras = rasterL,
                      rasCLASS = NULL,
                      mlinear = FALSE,
                      mixEffect = TRUE,
                      mixCat = mixV,
                      glmMODE = "binomial",
                      weight = sp.DF$myWEIGHT,
                      poly = TRUE,
                      polyBRUT = FALSE,
                      parallel = FALSE,
                      cores = NULL)

### wsl.varPPower(): example with a data.frame & rasters in classes
PPower.DF = wsl.varPPower(points=sp.DF[,c("x","y")],
                        val = sp.DF$myPA,
                        species = sp.DF$spCODES,
                        ras = rasterL,
                        rasCLASS = rCLASS,
                        mlinear = FALSE,
                        glmMODE = "binomial",
                        weight = sp.DF$myWEIGHT,
                        poly = TRUE,
                        polyBRUT = TRUE,
                        parallel = FALSE,
                        cores = NULL)

### wsl.varPPower(): Same example including random effect
PPower.DF = wsl.varPPower(points=sp.DF[,c("x","y")],
                        val = sp.DF$myPA,
                        species = sp.DF$spCODES,
                        ras = rasterL,
                        rasCLASS = rCLASS,
                        mlinear = FALSE,
                        mixEffect = TRUE,
                        mixCat = mixV,
                        glmMODE = "binomial",
                        weight = sp.DF$myWEIGHT,
                        poly = TRUE,
                        polyBRUT = TRUE,
                        parallel = FALSE,
                        cores = NULL)

### wsl.varPPower(): example with SpatialPointsDataFrame & rasters in classes

# Checking length of every species input
c(length(mySP),length(myPA),length(spCODES),length(myWEIGHT))

# Generate new random effect classes
mixL = lapply(sapply(mySP,length),function(x) sample(LETTERS[1:3],x, replace=TRUE))

# Running the function
PPower.SPDF = wsl.varPPower(points = mySP[[1]],
```

```
val = myPA[[1]],
species = NULL,
ras = rasterL,
rasCLASS = rCLASS,
mlinear = FALSE,
mixEffect=TRUE,
mixCat=mixL[[1]],
glmMODE = "binomial",
weight = myWEIGHT[[1]],
poly = TRUE,
polyBRUT = TRUE,
parallel = FALSE,
cores = NULL)
```

Index