Package 'biomod2'

October 22, 2026

Type Package

Title Ensemble Platform for Species Distribution Modeling

Version 4.3-4-8

Date 2026-10-19

BugReports https://github.com/biomodhub/biomod2/issues

URL https://biomodhub.github.io/biomod2/

Description Functions for species distribution modelling, to calibrate, evaluate, and project species-environment relationships across space and time using multiple modelling algorithms and ensemble forecasting. It accommodates diverse ecological data types (presence-only, presence-absence, counts, multi-class abundance, or relative/absolute abundance) within a unified modelling workflow which includes cross-validation schemes, pseudo-absence selection strategies, expanded model parametrization options, a dozen of algorithms, and tools for exploring and visualizing outputs.

Depends R (>= 4.1)

Imports stats, utils, methods, terra (>= 1.6-33), sp, reshape, reshape2, abind, foreach, ggplot2, gbm (>= 2.1.3), rpart, MASS, pROC (>= 1.15.0), PresenceAbsence, dplyr, rlang, scales

Suggests Hmisc, gam, mgcv, earth, maxnet, mda, nnet, randomForest, xgboost (>= 3.1.3.1), cito, torch, car, caret, dismo, ENMeval, doParallel, raster, ggpubr, testthat, knitr, markdown, rmarkdown, tidyterra, ggtext, patchwork, kableExtra, DT, R.utils, paletteer, viridis

License GPL-3

Encoding UTF-8

VignetteBuilder knitr

Collate 'biomod2-package.R' 'biomod2_globalVariables.R' 'biomod2_classes_0.R' 'biomod2_classes_1.R' 'biomod2_classes_2.R' 'biomod2_classes_3.R' 'biomod2_classes_4.R' 'biomod2_classes_5.R' 'biomod2_internal.R' 'biomod2_data.R'

'BIOMOD_EnsembleForecasting.R' 'BIOMOD_EnsembleModeling.R'
'BIOMOD_FormatingData.R' 'BIOMOD_LoadModels.R'
'BIOMOD_Modeling.R' 'BIOMOD_Projection.R' 'BIOMOD_RangeSize.R'
'BIOMOD_Report.R' 'DEPRECATED.R' 'bm_BinaryTransformation.R'
'bm_CrossValidation.R' 'bm_FindOptimStat.R' 'bm_MakeFormula.R'
'bm_ModelingOptions.R' 'bm_ModelAnalysis.R'
'bm_PlotEvalBoxplot.R' 'bm_PlotEvalMean.R' 'bm_PlotRangeSize.R'
'bm_PlotResponseCurves.R' 'bm_PlotVarImpBoxplot.R'
'bm_PseudoAbsences.R' 'bm_RunModelsLoop.R' 'bm_RangeSize.R'
'bm_SRE.R' 'bm_SampleBinaryVector.R' 'bm_SampleFactorLevels.R'
'bm_Tuning.R' 'bm_VariablesImportance.R' 'zzz.R'

LazyData true

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Maya Guéguen [aut, cre] (ORCID:
<https://orcid.org/0000-0002-1045-2997>),
Hélène Blancheteau [aut],
Rémi Lemaire-Patin [aut],
Wilfried Thuiller [aut] (ORCID:
<https://orcid.org/0000-0002-5388-5274>)

Maintainer Maya Guéguen <maya.gueguen@univ-grenoble-alpes.fr>

Repository CRAN

Date/Publication 2026-10-22 09:50:02 UTC

Contents

bioclim_current 3
bioclim_future 4
BIOMOD.ensemble.models.out 4
BIOMOD.formated.data 7
BIOMOD.formated.data.PA 11
BIOMOD.models.options 16
BIOMOD.models.out 17
BIOMOD.options.dataset 19
BIOMOD.options.default 21
BIOMOD.projection.out 22
BIOMOD.rangesize.out 25
BIOMOD.stored.data 26
biomod2_ensemble_model 27
biomod2_model 29
BIOMOD_EnsembleForecasting 30
BIOMOD_EnsembleModeling 35
BIOMOD_FormatingData 42
BIOMOD_LoadModels 49
BIOMOD_Modeling 51
BIOMOD_Projection 58

BIOMOD_RangeSize . . . 62
BIOMOD_Report . . . 65
bm_BinaryTransformation . . . 69
bm_CrossValidation . . . 71
bm_FindOptimStat . . . 76
bm_MakeFormula . . . 80
bm_ModelAnalysis . . . 81
bm_ModelingOptions . . . 84
bm_PlotEvalBoxplot . . . 88
bm_PlotEvalMean . . . 90
bm_PlotRangeSize . . . 93
bm_PlotResponseCurves . . . 96
bm_PlotVarImpBoxplot . . . 100
bm_PseudoAbsences . . . 102
bm_RangeSize . . . 107
bm_RunModelsLoop . . . 111
bm_SampleBinaryVector . . . 114
bm_SampleFactorLevels . . . 115
bm_SRE . . . 117
bm_Tuning . . . 119
bm_VariablesImportance . . . 124
DataSpecies . . . 126
DataSTOC . . . 127
getters.bm . . . 128
getters.out . . . 129
load_stored_object . . . 135
ModelsTable . . . 136
ODMAP . . . 137
OptionsBigboss . . . 137
plot,BIOMOD.formated.data,missing-method . . . 139
setters . . . 141
summary,BIOMOD.formated.data-method . . . 142

Index **144**

`bioclim_current` *Bioclimatic variables for SDM based on current condition*

Description

A `SpatRaster` with 5 bioclimatic variables commonly used for SDM and describing current climate. Additional information available at worldclim.

Usage

```
bioclim_current
```

Format

A SpatRaster with 5 layers:

bio3 Isothermality

bio4 Temperature Seasonality

bio7 Temperature Annual Range

bio11 Mean Temperature of Coldest Quarter

bio12 Annual Precipitation

---

bioclim_future *Bioclimatic variables for SDM based on future condition*

Description

A SpatRaster with 5 bioclimatic variables commonly used for SDM and describing future climate based on old RCP scenarios at the horizon 2080.

Usage

```
bioclim_future
```

Format

A SpatRaster with 5 layers:

bio3 Isothermality

bio4 Temperature Seasonality

bio7 Temperature Annual Range

bio11 Mean Temperature of Coldest Quarter

bio12 Annual Precipitation

BIOMOD.ensemble.models.out
BIOMOD_EnsembleModeling() output object class

Description

Class returned by BIOMOD_EnsembleModeling, and used by BIOMOD_LoadModels, BIOMOD_PresenceOnly and BIOMOD_EnsembleForecasting

Usage

```
## S4 method for signature 'BIOMOD.ensemble.models.out'
show(object)
```

Arguments

object	a BIOMOD.ensemble.models.out object

Slots

modeling.id a character corresponding to the name (ID) of the simulation set

dir.name a character corresponding to the modeling folder

sp.name a character corresponding to the species name

expl.var.names a vector containing names of explanatory variables

data.type a character corresponding to the data type

models.out a BIOMOD.stored.models.out-class object containing informations from BIOMOD_Modeling object

em.by a character corresponding to the way kept models have been combined to build the ensemble models, must be among PA+run, PA+algo, PA, algo, all

em.computed a vector containing names of ensemble models

em.failed a vector containing names of failed ensemble models

em.models_kept a list containing single models for each ensemble model

models.evaluation a BIOMOD.stored.data.frame-class object containing models evaluation

variables.importance a BIOMOD.stored.data.frame-class object containing variables importance

models.prediction a BIOMOD.stored.data.frame-class object containing models predictions

models.prediction.eval a BIOMOD.stored.data.frame-class object containing models predictions for evaluation data

link a character containing the file name of the saved object

call a language object corresponding to the call used to obtain the object

Author(s)

Damien Georges

See Also

BIOMOD_EnsembleModeling, BIOMOD_LoadModels, BIOMOD_PresenceOnly, bm_VariablesImportance, bm_PlotEvalMean, bm_PlotEvalBoxplot, bm_PlotVarImpBoxplot, bm_PlotResponseCurves

Other Toolbox objects: BIOMOD.formated.data(), BIOMOD.formated.data.PA(), BIOMOD.models.options, BIOMOD.models.out, BIOMOD.options.dataset(), BIOMOD.options.default(), BIOMOD.projection.out, BIOMOD.rangesize.out, BIOMOD.stored.data, biomod2_ensemble_model, biomod2_model

Examples

```
showClass("BIOMOD.ensemble.models.out")

## ------------------------------------------------------------------------ #
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



## ------------------------------------------------------------------------ #
file.out <- paste0(myRespName, "/", myRespName, ".AllModels.models.out")
if (file.exists(file.out)) {
  myBiomodModelOut <- get(load(file.out))
} else {

  # Format Data with true absences
  myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                       resp.var = myResp,
                                       resp.xy = myRespXY,
                                       expl.var = myExpl)

  # Model single models
  myBiomodModelOut <- BIOMOD_Modeling(bm.format = myBiomodData,
                                      modeling.id = 'AllModels',
                                      models = c('RF', 'GLM'),
                                      CV.strategy = 'random',
                                      CV.nb.rep = 2,
                                      CV.perc = 0.8,
                                      OPT.strategy = 'bigboss',
                                      metric.eval = c('TSS', 'AUCroc'),
                                      var.import = 3,
                                      seed.val = 42)
}


## ------------------------------------------------------------------------ #
```

```
# Model ensemble models
myBiomodEM <- BIOMOD_EnsembleModeling(bm.mod = myBiomodModelOut,
                                      models.chosen = 'all',
                                      em.by = 'all',
                                      em.algo = c('EMmean', 'EMca'),
                                      metric.select = c('TSS'),
                                      metric.select.thresh = c(0.7),
                                      metric.eval = c('TSS', 'AUCroc'),
                                      var.import = 3,
                                      seed.val = 42)
myBiomodEM
```

BIOMOD.formated.data BIOMOD_FormatingData() *output object class*

Description

Class returned by BIOMOD_FormatingData, and used by bm_Tuning, bm_CrossValidation and BIOMOD_Modeling

Usage

```
## S4 method for signature 'numeric,data.frame'
BIOMOD.formated.data(
  sp,
  env,
  xy = NULL,
  dir.name = ".",
  data.type = NULL,
  sp.name = NULL,
  eval.sp = NULL,
  eval.env = NULL,
  eval.xy = NULL,
  na.rm = TRUE,
  data.mask = NULL,
  shared.eval.env = FALSE,
  filter.raster = FALSE
)

## S4 method for signature 'data.frame,ANY'
BIOMOD.formated.data(
  sp,
  env,
  xy = NULL,
  dir.name = ".",
  data.type = NULL,
```

```
  sp.name = NULL,
  eval.sp = NULL,
  eval.env = NULL,
  eval.xy = NULL,
  na.rm = TRUE,
  filter.raster = FALSE
)

## S4 method for signature 'numeric,matrix'
BIOMOD.formated.data(
  sp,
  env,
  xy = NULL,
  dir.name = ".",
  data.type = NULL,
  sp.name = NULL,
  eval.sp = NULL,
  eval.env = NULL,
  eval.xy = NULL,
  na.rm = TRUE,
  filter.raster = FALSE
)

## S4 method for signature 'numeric,SpatRaster'
BIOMOD.formated.data(
  sp,
  env,
  xy = NULL,
  dir.name = ".",
  data.type = NULL,
  sp.name = NULL,
  eval.sp = NULL,
  eval.env = NULL,
  eval.xy = NULL,
  na.rm = TRUE,
  shared.eval.env = FALSE,
  filter.raster = FALSE
)

## S4 method for signature 'BIOMOD.formated.data'
show(object)
```

Arguments

sp	a vector, a SpatVector without associated data (*if presence-only*), or a SpatVector object containing binary data (0 : absence, 1 : presence, NA : indeterminate) for a single species that will be used to build the species distribution model(s) *Note that old format from* **sp** *are still supported such as* SpatialPoints *(if presence-only) or* SpatialPointsDataFrame *object containing binary data.*

`env` a `matrix`, `data.frame`, `SpatVector` or `SpatRaster` object containing the explanatory variables (in columns or layers) that will be used to build the species distribution model(s).
Note that old format from **raster** *and* **sp** *are still supported such as* `RasterStack` *and* `SpatialPointsDataFrame` *objects.*

`xy` (*optional, default* `NULL`)
If `resp.var` is a `vector`, a 2-columns `matrix` or `data.frame` containing the corresponding `X` and `Y` coordinates that will be used to build the species distribution model(s)

`dir.name` a `character` corresponding to the modeling folder

`data.type` a `character` corresponding to the data type

`sp.name` a `character` corresponding to the species name

`eval.sp` (*optional, default* `NULL`)
A `vector`, a `SpatVector` without associated data (*if presence-only*), or a `SpatVector` object containing binary data (`0` : absence, `1` : presence, `NA` : indeterminate) for a single species that will be used to evaluate the species distribution model(s) with independent data
Note that old format from **sp** *are still supported such as* `SpatialPoints` *(if presence-only) or* `SpatialPointsDataFrame` *object containing binary data.*

`eval.env` (*optional, default* `NULL`)
A `matrix`, `data.frame`, `SpatVector` or `SpatRaster` object containing the explanatory variables (in columns or layers) that will be used to evaluate the species distribution model(s) with independent data
Note that old format from **raster** *and* **sp** *are still supported such as* `RasterStack` *and* `SpatialPointsDataFrame` *objects.*

`eval.xy` (*optional, default* `NULL`)
If `resp.var` is a `vector`, a 2-columns `matrix` or `data.frame` containing the corresponding `X` and `Y` coordinates that will be used to evaluate the species distribution model(s) with independent data

`na.rm` (*optional, default* `TRUE`)
A `logical` value defining whether points having one or several missing values for explanatory variables should be removed from the analysis or not

`data.mask` (*optional, default* `NULL`)
A `SpatRaster` object containing the mask of the studied area

`shared.eval.env`
(*optional, default* `FALSE`)
A `logical` value defining whether the explanatory variables used for the evaluation dataset are the same than the ones for calibration (if `eval.env` not provided for example) or not

`filter.raster` (*optional, default* `FALSE`)
If `env` is of raster type, a `logical` value defining whether `sp` is to be filtered when several points occur in the same raster cell

`object` a `BIOMOD.formated.data` object

Slots

`data.type` a `character` corresponding to the data type

`dir.name` a `character` corresponding to the modeling folder

`sp.name` a `character` corresponding to the species name

`coord` a 2-columns `data.frame` containing the corresponding `X` and `Y` coordinates

`data.species` a `vector` containing the species observations (`0`, `1` or `NA`)

`data.env.var` a `data.frame` containing explanatory variables

`data.mask` a `SpatRaster` object containing the mask of the studied area

`has.data.eval` a `logical` value defining whether evaluation data is given

`has.filter.raster` a `logical` value defining whether filtering have been done or not

`eval.coord` (*optional, default* `NULL`)
: A 2-columns `data.frame` containing the corresponding `X` and `Y` coordinates for evaluation data

`eval.data.species` (*optional, default* `NULL`)
: A `vector` containing the species observations (`0`, `1` or `NA`) for evaluation data

`eval.data.env.var` (*optional, default* `NULL`)
: A `data.frame` containing explanatory variables for evaluation data

`biomod2.version` a `character` corresponding to the biomod2 version

`call` a `language` object corresponding to the call used to obtain the object

Author(s)

Damien Georges

See Also

`BIOMOD_FormatingData`, `bm_Tuning`, `bm_CrossValidation`, `BIOMOD_Modeling`, `bm_RunModelsLoop`

Other Toolbox objects: `BIOMOD.ensemble.models.out`, `BIOMOD.formated.data.PA()`, `BIOMOD.models.options`, `BIOMOD.models.out`, `BIOMOD.options.dataset()`, `BIOMOD.options.default()`, `BIOMOD.projection.out`, `BIOMOD.rangesize.out`, `BIOMOD.stored.data`, `biomod2_ensemble_model`, `biomod2_model`

Examples

```
showClass("BIOMOD.formated.data")

## ------------------------------------------------------------------------ #
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'
```

```
# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



## ------------------------------------------------------------------------ #
# Format Data with true absences
myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                     resp.var = myResp,
                                     resp.xy = myRespXY,
                                     expl.var = myExpl)
myBiomodData
plot(myBiomodData)
summary(myBiomodData)
```

`BIOMOD.formated.data.PA` *BIOMOD_FormatingData() output object class (with pseudo-absences)*

Description

Class returned by BIOMOD_FormatingData, and used by bm_Tuning, bm_CrossValidation and BIOMOD_Modeling

Usage

```
## S4 method for signature 'numeric,data.frame'
BIOMOD.formated.data.PA(
  sp,
  env,
  xy = NULL,
  dir.name = ".",
  sp.name = NULL,
  eval.sp = NULL,
  eval.env = NULL,
  eval.xy = NULL,
  PA.strategy,
  PA.nb.rep = NULL,
  PA.nb.absences = NULL,
```

```
  PA.dist.min = 0,
  PA.dist.max = NULL,
  PA.sre.quant = NULL,
  PA.fact.aggr = NULL,
  PA.user.table = NULL,
  na.rm = TRUE,
  filter.raster = FALSE,
  seed.val = NULL
)

## S4 method for signature 'numeric,SpatRaster'
BIOMOD.formated.data.PA(
  sp,
  env,
  xy = NULL,
  dir.name = ".",
  sp.name = NULL,
  eval.sp = NULL,
  eval.env = NULL,
  eval.xy = NULL,
  PA.strategy,
  PA.nb.rep = NULL,
  PA.nb.absences = NULL,
  PA.dist.min = 0,
  PA.dist.max = NULL,
  PA.sre.quant = NULL,
  PA.fact.aggr = NULL,
  PA.user.table = NULL,
  na.rm = TRUE,
  filter.raster = FALSE,
  seed.val = NULL
)
```

Arguments

sp	a vector, a SpatVector without associated data (*if presence-only*), or a SpatVector object containing binary data (0 : absence, 1 : presence, NA : indeterminate) for a single species that will be used to build the species distribution model(s) *Note that old format from* **sp** *are still supported such as* SpatialPoints *(if presence-only) or* SpatialPointsDataFrame *object containing binary data.*
env	a matrix, data.frame, SpatVector or SpatRaster object containing the explanatory variables (in columns or layers) that will be used to build the species distribution model(s). *Note that old format from* **raster** *and* **sp** *are still supported such as* RasterStack *and* SpatialPointsDataFrame *objects.*
xy	(*optional, default* NULL) If resp.var is a vector, a 2-columns matrix or data.frame containing the corresponding X and Y coordinates that will be used to build the species distri-

bution model(s)

`dir.name` a `character` corresponding to the modeling folder

`sp.name` a `character` corresponding to the species name

`eval.sp` (*optional, default* `NULL`)
A `vector`, a `SpatVector` without associated data (*if presence-only*), or a `SpatVector` object containing binary data (`0` : absence, `1` : presence, `NA` : indeterminate) for a single species that will be used to evaluate the species distribution model(s) with independent data
Note that old format from **sp** *are still supported such as* `SpatialPoints` *(if presence-only) or* `SpatialPointsDataFrame` *object containing binary data.*

`eval.env` (*optional, default* `NULL`)
A `matrix`, `data.frame`, `SpatVector` or `SpatRaster` object containing the explanatory variables (in columns or layers) that will be used to evaluate the species distribution model(s) with independent data
Note that old format from **raster** *and* **sp** *are still supported such as* `RasterStack` *and* `SpatialPointsDataFrame` *objects.*

`eval.xy` (*optional, default* `NULL`)
If `resp.var` is a `vector`, a 2-columns `matrix` or `data.frame` containing the corresponding `X` and `Y` coordinates that will be used to evaluate the species distribution model(s) with independent data

`PA.strategy` (*optional, default* `NULL`)
If pseudo-absence selection, a `character` defining the strategy that will be used to select the pseudo-absence points. Must be `random`, `sre`, `disk` or `user.defined` (see Details)

`PA.nb.rep` (*optional, default* `NULL`)
If pseudo-absence selection, an `integer` corresponding to the number of sets (repetitions) of pseudo-absence points that will be drawn

`PA.nb.absences` (*optional, default* `NULL`)
If pseudo-absence selection, and `PA.strategy = 'random'` or `PA.strategy = 'sre'` or `PA.strategy = 'disk'`, an `integer` (or a `vector` of `integer` the same size as `PA.nb.rep`) corresponding to the number of pseudo-absence points that will be selected for each pseudo-absence repetition (true absences included)

`PA.dist.min` (*optional, default* `0`)
If pseudo-absence selection and `PA.strategy = 'disk'`, a `numeric` defining the minimal distance to presence points used to make the `disk` pseudo-absence selection (in the same projection system units as `coord`, see Details)

`PA.dist.max` (*optional, default* `NULL`)
If pseudo-absence selection and `PA.strategy = 'disk'`, a `numeric` defining the maximal distance to presence points used to make the `disk` pseudo-absence selection (in the same projection system units as `coord`, see Details)

`PA.sre.quant` (*optional, default* `NULL`)
If pseudo-absence selection and `PA.strategy = 'sre'`, a `numeric` between `0` and `0.5` defining the half-quantile used to make the `sre` pseudo-absence selection (see Details)

`PA.fact.aggr` (*optional, default* `NULL`)
If `strategy = 'random'` or `strategy = 'disk'`, a `integer` defining the factor of aggregation to reduce the resolution

`PA.user.table` (*optional, default* `NULL`)
If pseudo-absence selection and `PA.strategy = 'user.defined'`, a `matrix` or `data.frame` with as many rows as `resp.var` values, as many columns as `PA.nb.rep`, and containing `TRUE` or `FALSE` values defining which points will be used to build the species distribution model(s) for each repetition (see Details)

`na.rm` (*optional, default* `TRUE`)
A `logical` value defining whether points having one or several missing values for explanatory variables should be removed from the analysis or not

`filter.raster` (*optional, default* `FALSE`)
If `env` is of raster type, a `logical` value defining whether `sp` is to be filtered when several points occur in the same raster cell

`seed.val` (*optional, default* `NULL`)
An `integer` value corresponding to the new seed value to be set

Slots

`dir.name` a `character` corresponding to the modeling folder

`sp.name` a `character` corresponding to the species name

`coord` a 2-columns `data.frame` containing the corresponding `X` and `Y` coordinates

`data.species` a `vector` containing the species observations (`0`, `1` or `NA`)

`data.env.var` a `data.frame` containing explanatory variables

`data.mask` a `SpatRaster` object containing the mask of the studied area

`has.data.eval` a `logical` value defining whether evaluation data is given

`eval.coord` (*optional, default* `NULL`)
A 2-columns `data.frame` containing the corresponding `X` and `Y` coordinates for evaluation data

`eval.data.species` (*optional, default* `NULL`)
A `vector` containing the species observations (`0`, `1` or `NA`) for evaluation data

`eval.data.env.var` (*optional, default* `NULL`)
A `data.frame` containing explanatory variables for evaluation data

`PA.strategy` a `character` corresponding to the pseudo-absence selection strategy

`PA.table` a `data.frame` containing the corresponding table of selected pseudo-absences (indicated by `TRUE` or `FALSE`) from the `pa.tab` list element returned by the `bm_PseudoAbsences` function

Author(s)

Damien Georges

See Also

BIOMOD_FormatingData, bm_PseudoAbsences, bm_Tuning, bm_CrossValidation, BIOMOD_Modeling, bm_RunModelsLoop

Other Toolbox objects: BIOMOD.ensemble.models.out, BIOMOD.formated.data(), BIOMOD.models.options, BIOMOD.models.out, BIOMOD.options.dataset(), BIOMOD.options.default(), BIOMOD.projection.out, BIOMOD.rangesize.out, BIOMOD.stored.data, biomod2_ensemble_model, biomod2_model

Examples

```
showClass("BIOMOD.formated.data.PA")

## ------------------------------------------------------------------------ #
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Keep only presence informations
DataSpecies <- DataSpecies[which(DataSpecies[, myRespName] == 1), ]

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



## ------------------------------------------------------------------------ #
# Format Data with pseudo-absences : random method
myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                     resp.var = myResp,
                                     resp.xy = myRespXY,
                                     expl.var = myExpl,
                                     PA.nb.rep = 4,
                                     PA.strategy = 'random',
                                     PA.nb.absences = 1000)
myBiomodData
plot(myBiomodData)
```

BIOMOD.models.options bm_ModelingOptions *output object class*

Description

Class returned by bm_ModelingOptions and used by BIOMOD_Modeling

Usage

```
## S4 method for signature 'BIOMOD.models.options'
show(object)

## S4 method for signature 'BIOMOD.models.options'
print(x, dataset = "_allData_allRun")
```

Arguments

object	a BIOMOD.models.options object
x	a BIOMOD.models.options object
dataset	a character corresponding to the name of a dataset contained in the arg.values slot of the BIOMOD.options.dataset object for each model

Slots

models a vector containing model names for which options have been retrieved and defined, must be algo.datatype.package.function

options a list containing BIOMOD.options.dataset object for each model

Author(s)

Maya Guéguen

See Also

BIOMOD.options.default, BIOMOD.options.dataset, bm_ModelingOptions, bm_Tuning, BIOMOD_Modeling

Other Toolbox objects: BIOMOD.ensemble.models.out, BIOMOD.formated.data(), BIOMOD.formated.data.PA(), BIOMOD.models.out, BIOMOD.options.dataset(), BIOMOD.options.default(), BIOMOD.projection.out, BIOMOD.rangesize.out, BIOMOD.stored.data, biomod2_ensemble_model, biomod2_model

Examples

```
showClass("BIOMOD.models.options")
```

BIOMOD.models.out BIOMOD_Modeling() *output object class*

Description

Class returned by BIOMOD_Modeling, and used by BIOMOD_LoadModels, BIOMOD_PresenceOnly, BIOMOD_Projection and BIOMOD_EnsembleModeling

Usage

```
## S4 method for signature 'BIOMOD.models.out'
show(object)
```

Arguments

object	a BIOMOD.models.out object

Slots

modeling.id a character corresponding to the name (ID) of the simulation set

dir.name a character corresponding to the modeling folder

sp.name a character corresponding to the species name

data.type a character corresponding to the data type

expl.var.names a vector containing names of explanatory variables

models.computed a vector containing names of computed models

models.failed a vector containing names of failed models

has.evaluation.data a logical value defining whether evaluation data is given

scale.models a logical value defining whether models have been rescaled or not

formated.input.data a BIOMOD.stored.formated.data-class object containing informations from BIOMOD_FormatingData object

calib.lines a BIOMOD.stored.data.frame-class object containing calibration lines

models.options a BIOMOD.stored.options-class object containing informations from bm_ModelingOptions object

models.evaluation a BIOMOD.stored.data.frame-class object containing models evaluation

variables.importance a BIOMOD.stored.data.frame-class object containing variables importance

models.prediction a BIOMOD.stored.data.frame-class object containing models predictions

models.prediction.eval a BIOMOD.stored.data.frame-class object containing models predictions for evaluation data

link a character containing the file name of the saved object

call a language object corresponding to the call used to obtain the object

Author(s)

Damien Georges

See Also

BIOMOD_Modeling, BIOMOD_LoadModels, BIOMOD_PresenceOnly, BIOMOD_Projection, BIOMOD_EnsembleModeling, bm_VariablesImportance, bm_PlotEvalMean, bm_PlotEvalBoxplot, bm_PlotVarImpBoxplot, bm_PlotResponseCurves

Other Toolbox objects: BIOMOD.ensemble.models.out, BIOMOD.formated.data(), BIOMOD.formated.data.PA(), BIOMOD.models.options, BIOMOD.options.dataset(), BIOMOD.options.default(), BIOMOD.projection.out, BIOMOD.rangesize.out, BIOMOD.stored.data, biomod2_ensemble_model, biomod2_model

Examples

```
showClass("BIOMOD.models.out")

## ------------------------------------------------------------------------ #
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



## ------------------------------------------------------------------------ #
# Format Data with true absences
myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                     resp.var = myResp,
                                     resp.xy = myRespXY,
                                     expl.var = myExpl)

## ------------------------------------------------------------------------ #
# Model single models
myBiomodModelOut <- BIOMOD_Modeling(bm.format = myBiomodData,
                                    modeling.id = 'AllModels',
                                    models = c('RF', 'GLM'),
                                    CV.strategy = 'random',
```

```
                                    CV.nb.rep = 2,
                                    CV.perc = 0.8,
                                    OPT.strategy = 'bigboss',
                                    metric.eval = c('TSS', 'AUCroc'),
                                    var.import = 3,
                                    seed.val = 42)
myBiomodModelOut
```

BIOMOD.options.dataset

bm_ModelingOptions *output object class*

Description

Class returned by bm_ModelingOptions (a list of BIOMOD.options.dataset more exactly), and used by BIOMOD_Modeling

Usage

```
## S4 method for signature 'character'
BIOMOD.options.dataset(
  mod,
  typ,
  pkg,
  fun,
  strategy,
  user.val = NULL,
  user.base = NULL,
  tuning.fun = NULL,
  bm.format = NULL,
  calib.lines = NULL
)

## S4 method for signature 'BIOMOD.options.dataset'
show(object)

## S4 method for signature 'BIOMOD.options.dataset'
print(x, dataset = "_allData_allRun")
```

Arguments

mod	a character corresponding to the model name to be computed, must be either ANN, CTA, DNN, FDA, GAM, GBM, GLM, MARS, MAXENT, MAXNET, RF, SRE, XGBOOST
typ	a character corresponding to the data type to be used, must be either binary, binary.PA, abundance, compositional

pkg	a character corresponding to the package containing the model function to be called
fun	a character corresponding to the model function name to be called
strategy	a character corresponding to the method to select models' parameters values, must be either default, bigboss, user.defined, tuned
user.val	(*optional, default* NULL) A list containing parameters values
user.base	(*optional, default* NULL) A character, default or bigboss used when strategy = 'user.defined'. It sets the bases of parameters to be modified by user defined values.
tuning.fun	(*optional, default* NULL) A character corresponding to the model function name to be called through train function for tuning parameters
bm.format	(*optional, default* NULL) A BIOMOD.formated.data or BIOMOD.formated.data.PA object returned by the BIOMOD_FormatingData function
calib.lines	(*optional, default* NULL) A data.frame object returned by get_calib_lines or bm_CrossValidation functions, to explore the distribution of calibration and validation datasets
object	a BIOMOD.options.dataset object
x	a BIOMOD.options.dataset object
dataset	a character corresponding to the name of a dataset contained in the arg.values slot

Slots

model a character corresponding to the model

type a character corresponding to the data type (binary, binary.PA, abundance, compositional)

package a character corresponding to the package containing the model function to be called

func a character corresponding to the model function name to be called

args.names a vector containing character corresponding to the model function arguments

args.default a list containing for each dataset the default values for all arguments listed in args.names

args.values a list containing for each dataset the to-be-used values for all arguments listed in args.names

Author(s)

Maya Guéguen

See Also

BIOMOD.options.default, bm_ModelingOptions, bm_Tuning, BIOMOD_Modeling, bm_RunModelsLoop

Other Toolbox objects: BIOMOD.ensemble.models.out, BIOMOD.formated.data(), BIOMOD.formated.data.PA(), BIOMOD.models.options, BIOMOD.models.out, BIOMOD.options.default(), BIOMOD.projection.out, BIOMOD.rangesize.out, BIOMOD.stored.data, biomod2_ensemble_model, biomod2_model

Examples

```
showClass("BIOMOD.options.dataset")
```

BIOMOD.options.default

bm_ModelingOptions *output object class*

Description

Class returned by bm_ModelingOptions (a list of BIOMOD.options.dataset more exactly), and used by BIOMOD_Modeling

Usage

```
## S4 method for signature 'character,character'
BIOMOD.options.default(mod, typ, pkg, fun)
```

Arguments

mod	a character corresponding to the model name to be computed, must be either ANN, CTA, DNN, FDA, GAM, GBM, GLM, MARS, MAXENT, MAXNET, RF, SRE, XGBOOST
typ	a character corresponding to the data type to be used, must be either binary, binary.PA, abundance, compositional
pkg	a character corresponding to the package containing the model function to be called
fun	a character corresponding to the model function name to be called

Slots

model a character corresponding to the model

type a character corresponding to the data type (binary, binary.PA, abundance, compositional)

package a character corresponding to the package containing the model function to be called

func a character corresponding to the model function name to be called

args.names a vector containing character corresponding to the model function arguments

args.default a list containing for each dataset the default values for all arguments listed in args.names

Author(s)

Maya Guéguen

See Also

BIOMOD.options.dataset, bm_ModelingOptions, bm_Tuning, BIOMOD_Modeling, bm_RunModelsLoop

Other Toolbox objects: BIOMOD.ensemble.models.out, BIOMOD.formated.data(), BIOMOD.formated.data.PA(), BIOMOD.models.options, BIOMOD.models.out, BIOMOD.options.dataset(), BIOMOD.projection.out, BIOMOD.rangesize.out, BIOMOD.stored.data, biomod2_ensemble_model, biomod2_model

Examples

```
showClass("BIOMOD.options.default")
```

BIOMOD.projection.out BIOMOD_Projection() *output object class*

Description

Class returned by BIOMOD_Projection, and used by BIOMOD_EnsembleForecasting

Usage

```
## S4 method for signature 'BIOMOD.projection.out,missing'
plot(
  x,
  coord = NULL,
  plot.output,
  do.plot = TRUE,
  std = TRUE,
  scales,
  size,
  maxcell = 5e+05,
  ...
)

## S4 method for signature 'BIOMOD.projection.out'
show(object)
```

Arguments

x	a BIOMOD.projection.out object
coord	a 2-columns data.frame containing the corresponding X and Y
plot.output	(*optional, default* facet) a character determining the type of output: with plot.output = 'list' the function will return a list of plots (one plot per model) ; with 'facet' ; with plot.output = 'facet' the function will return a single plot with all asked projections as facet.

do.plot	(*optional, default* TRUE) a boolean determining whether the plot should be displayed or just returned.
std	(*optional, default* TRUE) a boolean controlling the limits of the color scales. With std = TRUE color scales are displayed between 0 and 1 (or 1000). With std = FALSE color scales are displayed between 0 and the maximum value observed.
scales	(*optional, default* fixed) a character determining whether x and y scales are shared among facet. Argument passed to facet_wrap. Possible values: 'fixed', 'free_x', 'free_y', 'free'.
size	(*optional, default* 0.75) a numeric determining the size of points on the plots and passed to geom_point.
maxcell	maximum number of cells to plot. Argument transmitted to plot.
...	additional parameters to be passed to get_predictions to select the models that will be plotted
object	a BIOMOD.projection.out object

Slots

modeling.id a character corresponding to the name (ID) of the simulation set

proj.name a character corresponding to the projection name

dir.name a character corresponding to the modeling folder

sp.name a character corresponding to the species name

expl.var.names a vector containing names of explanatory variables

coord a 2-columns matrix or data.frame containing the corresponding X and Y coordinates used to project the species distribution model(s)

scale.models a logical value defining whether models have been rescaled or not

models.projected a vector containing names of projected models

models.out a BIOMOD.stored.data object

type a character corresponding to the class of the val slot of the proj.out slot

data.type a character corresponding to the data type

proj.out a BIOMOD.stored.data object

call a language object corresponding to the call used to obtain the object

Author(s)

Damien Georges

See Also

BIOMOD_Projection, BIOMOD_EnsembleForecasting

Other Toolbox objects: BIOMOD.ensemble.models.out, BIOMOD.formated.data(), BIOMOD.formated.data.PA(), BIOMOD.models.options, BIOMOD.models.out, BIOMOD.options.dataset(), BIOMOD.options.default(), BIOMOD.rangesize.out, BIOMOD.stored.data, biomod2_ensemble_model, biomod2_model

Examples

```
showClass("BIOMOD.projection.out")

## ------------------------------------------------------------------------ #
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



## ------------------------------------------------------------------------ #
file.out <- paste0(myRespName, "/", myRespName, ".AllModels.models.out")
if (file.exists(file.out)) {
  myBiomodModelOut <- get(load(file.out))
} else {

  # Format Data with true absences
  myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                       resp.var = myResp,
                                       resp.xy = myRespXY,
                                       expl.var = myExpl)

  # Model single models
  myBiomodModelOut <- BIOMOD_Modeling(bm.format = myBiomodData,
                                      modeling.id = 'AllModels',
                                      models = c('RF', 'GLM'),
                                      CV.strategy = 'random',
                                      CV.nb.rep = 2,
                                      CV.perc = 0.8,
                                      OPT.strategy = 'bigboss',
                                      metric.eval = c('TSS', 'AUCroc'),
                                      var.import = 3,
                                      seed.val = 42)
}


## ------------------------------------------------------------------------ #
```

```
# Project single models
myBiomodProj <- BIOMOD_Projection(bm.mod = myBiomodModelOut,
                                  proj.name = 'Current',
                                  new.env = myExpl,
                                  models.chosen = 'all',
                                  metric.binary = 'all',
                                  metric.filter = 'all',
                                  build.clamping.mask = TRUE)
myBiomodProj
plot(myBiomodProj)
```

`BIOMOD.rangesize.out` `BIOMOD_RangeSize()` *output object class*

Description

Class returned by `BIOMOD_RangeSize`, and used by `bm_PlotRangeSize`

Slots

`Compt.By.Models` a `data.frame` containing the summary of range change for each comparison

`Diff.By.Pixel` a `SpatRaster` or a `data.frame` containing a value for each point/pixel of each comparison

`loss.gain` a `SpatRaster` or a `data.frame` containg for each point/pixel a value indicating a loss, a gain or a stable sitatution

`data.type` a `character` corresponding to the data type

`coord` a 2-columns `matrix` or `data.frame` containing the corresponding `X` and `Y` coordinates used to project the species distribution model(s)

`row.names` A `vector` containing tags matching models names splitted by '_' character

Author(s)

Hélène Blancheteau

See Also

Other Toolbox objects: `BIOMOD.ensemble.models.out`, `BIOMOD.formated.data()`, `BIOMOD.formated.data.PA()`, `BIOMOD.models.options`, `BIOMOD.models.out`, `BIOMOD.options.dataset()`, `BIOMOD.options.default()`, `BIOMOD.projection.out`, `BIOMOD.stored.data`, `biomod2_ensemble_model`, `biomod2_model`

BIOMOD.stored.data *BIOMOD_Modeling and BIOMOD_EnsembleModeling output object class*

Description

Classes used by BIOMOD_Modeling and BIOMOD_EnsembleModeling to build their output object (see BIOMOD.models.out objects)

Details

BIOMOD.stored.data is the basic object containing the slots inMemory and link.
All listed classes below are derived from BIOMOD.stored.data, and contain a val slot of specific type :

- BIOMOD.stored.data.frame : val is a data.frame
- BIOMOD.stored.SpatRaster : val is a PackedSpatRaster
- BIOMOD.stored.files : val is a character
- BIOMOD.stored.formated.data : val is a BIOMOD.formated.data object
- BIOMOD.stored.options : val is a BIOMOD.models.options object
- BIOMOD.stored.models.out : val is a BIOMOD.models.out object

Slots

inMemory a logical defining whether the val slot has been loaded in memory or not

link a character containing the file name of the saved val slot

val an object of type depending on the BIOMOD.stored.[...] class (see Details)

Author(s)

Damien Georges

See Also

BIOMOD.formated.data, BIOMOD.models.out, BIOMOD_Modeling, BIOMOD_EnsembleModeling, BIOMOD_Projection, BIOMOD_EnsembleForecasting

Other Toolbox objects: BIOMOD.ensemble.models.out, BIOMOD.formated.data(), BIOMOD.formated.data.PA(), BIOMOD.models.options, BIOMOD.models.out, BIOMOD.options.dataset(), BIOMOD.options.default(), BIOMOD.projection.out, BIOMOD.rangesize.out, biomod2_ensemble_model, biomod2_model

Examples

```
showClass("BIOMOD.stored.data")
showClass("BIOMOD.stored.data.frame")
showClass("BIOMOD.stored.SpatRaster")
showClass("BIOMOD.stored.files")
showClass("BIOMOD.stored.formated.data")
showClass("BIOMOD.stored.options")
showClass("BIOMOD.stored.models.out")
```

---

biomod2_ensemble_model

Ensemble model output object class (when running `BIOMOD_EnsembleModeling()`*)*

Description

Class created by `BIOMOD_EnsembleModeling`

Usage

```
## S4 method for signature 'biomod2_ensemble_model'
show(object)
```

Arguments

`object`	a `biomod2_ensemble_model` object

Details

`biomod2_model` is the basic object for **biomod2** ensemble species distribution models.
All listed classes below are derived from `biomod2_model`, and have a `model_class` slot specific value :

- `biomod2_ensemble_model` : `model_class` is `EM`
- `EMmean_biomod2_model` : `model_class` is `EMmean`
- `EMmedian_biomod2_model` : `model_class` is `EMmedian`
- `EMcv_biomod2_model` : `model_class` is `EMcv`
- `EMci_biomod2_model` : `model_class` is `EMci`
- `EMca_biomod2_model` : `model_class` is `EMca`
- `EMwmean_biomod2_model` : `model_class` is `EMwmean`
- `EMmode_biomod2_model` : `model_class` is `EMmode`
- `EMfreq_biomod2_model` : `model_class` is `EMfreq`

Slots

`modeling.id` a `character` corresponding to the name (ID) of the simulation set

`model_name` a `character` corresponding to the model name

`model_class` a `character` corresponding to the model class

`model_options` a `list` containing the model options

`model` the corresponding model object

`scaling_model` the corresponding scaled model object

`dir_name` a `character` corresponding to the modeling folder

`resp_name` a `character` corresponding to the species name

`expl_var_names` a `vector` containing names of explanatory variables

`expl_var_type` a `vector` containing classes of explanatory variables

`expl_var_range` a `list` containing ranges of explanatory variables

`model_evaluation` a `data.frame` containing the model evaluations

`model_variables_importance` a `data.frame` containing the model variables importance

Author(s)

Damien Georges

See Also

`biomod2_model`, `BIOMOD_EnsembleModeling`

Other Toolbox objects: `BIOMOD.ensemble.models.out`, `BIOMOD.formated.data()`, `BIOMOD.formated.data.PA()`, `BIOMOD.models.options`, `BIOMOD.models.out`, `BIOMOD.options.dataset()`, `BIOMOD.options.default()`, `BIOMOD.projection.out`, `BIOMOD.rangesize.out`, `BIOMOD.stored.data`, `biomod2_model`

Examples

```
showClass("biomod2_ensemble_model")
showClass("EMmean_biomod2_model")
showClass("EMmedian_biomod2_model")
showClass("EMcv_biomod2_model")
showClass("EMci_biomod2_model")
showClass("EMca_biomod2_model")
showClass("EMwmean_biomod2_model")
showClass("EMmode_biomod2_model")
showClass("EMfreq_biomod2_model")
```

biomod2_model *Single model output object class (when running* BIOMOD_Modeling()*)*

Description

Class created by BIOMOD_Modeling and bm_RunModel

Usage

```
## S4 method for signature 'biomod2_model'
show(object)
```

Arguments

object	a biomod2_model object

Details

biomod2_model is the basic object for **biomod2** single species distribution models.
All listed classes below are derived from biomod2_model, and have a model_class slot specific value :

- ANN_biomod2_model : model_class is ANN
- CTA_biomod2_model : model_class is CTA
- DNN_biomod2_model : model_class is DNN
- FDA_biomod2_model : model_class is FDA
- GBM_biomod2_model : model_class is GBM
- GLM_biomod2_model : model_class is GLM
- MARS_biomod2_model : model_class is MARS
- MAXENT_biomod2_model : model_class is MAXENT
- MAXNET_biomod2_model : model_class is MAXNET
- RF_biomod2_model : model_class is RF
- RFd_biomod2_model : model_class is RFd
- SRE_biomod2_model : model_class is SRE

Slots

model_name a character corresponding to the model name

model_class a character corresponding to the model class

model_type a character corresponding to the type of data

model_options a list containing the model options

model the corresponding model object

scaling_model the corresponding scaled model object

`levels_factor` the levels of the original data for ordinal and multiclass

`dir_name` a `character` corresponding to the modeling folder

`resp_name` a `character` corresponding to the species name

`expl_var_names` a `vector` containing names of explanatory variables

`expl_var_type` a `vector` containing classes of explanatory variables

`expl_var_range` a `list` containing ranges of explanatory variables

`model_evaluation` a `data.frame` containing the model evaluations

`model_variables_importance` a `data.frame` containing the model variables importance

Author(s)

Damien Georges

See Also

`BIOMOD_Modeling`, `bm_RunModel`

Other Toolbox objects: `BIOMOD.ensemble.models.out`, `BIOMOD.formated.data()`, `BIOMOD.formated.data.PA()`, `BIOMOD.models.options`, `BIOMOD.models.out`, `BIOMOD.options.dataset()`, `BIOMOD.options.default()`, `BIOMOD.projection.out`, `BIOMOD.rangesize.out`, `BIOMOD.stored.data`, `biomod2_ensemble_model`

Examples

```
showClass("biomod2_model")
showClass("ANN_biomod2_model")
showClass("CTA_biomod2_model")
showClass("DNN_biomod2_model")
showClass("FDA_biomod2_model")
showClass("GAM_biomod2_model")
showClass("GBM_biomod2_model")
showClass("GLM_biomod2_model")
showClass("MARS_biomod2_model")
showClass("MAXENT_biomod2_model")
showClass("MAXNET_biomod2_model")
showClass("RF_biomod2_model")
showClass("RFd_biomod2_model")
showClass("SRE_biomod2_model")
```

BIOMOD_EnsembleForecasting

Project ensemble species distribution models onto new environment

---

Description

This function allows to project ensemble models built with the `BIOMOD_EnsembleModeling` function onto new environmental data (*which can represent new areas, resolution or time scales for example*).

Usage

```
BIOMOD_EnsembleForecasting(
  bm.em,
  bm.proj = NULL,
  proj.name = NULL,
  new.env = NULL,
  new.env.xy = NULL,
  models.chosen = "all",
  metric.binary = NULL,
  metric.filter = NULL,
  na.rm = TRUE,
  nb.cpu = 1,
  ...
)
```

Arguments

bm.em	a BIOMOD.ensemble.models.out object returned by the BIOMOD_EnsembleModeling function
bm.proj	a BIOMOD.projection.out object returned by the BIOMOD_Projection function
proj.name	(*optional, default* NULL) If bm.proj = NULL, a character corresponding to the name (ID) of the projection set (*a new folder will be created within the simulation folder with this name*)
new.env	(*optional, default* NULL) If bm.proj = NULL, a matrix, data.frame or SpatRaster object containing the new explanatory variables (in columns or layers, with names matching the variables names given to the BIOMOD_FormatingData function to build bm.mod) that will be used to project the species distribution model(s) *Note that old format from* **raster** *are still supported such as* RasterStack *objects.*
new.env.xy	(*optional, default* NULL) If new.env is a matrix or a data.frame, a 2-columns matrix or data.frame containing the corresponding X and Y coordinates that will be used to project the ensemble species distribution model(s)
models.chosen	a vector containing model names to be kept, must be either all or a subselection of model names that can be obtained with the get_built_models function applied to bm.mod
metric.binary	(*optional, default* NULL) A vector containing evaluation metric names to be used to transform prediction values into binary values based on models evaluation scores obtained with the BIOMOD_EnsembleModeling function. Must be among all (same evaluation metrics than those of bm.mod) or POD, FAR, POFD, SR, ACCURACY, BIAS, AUCroc, AUCprg, TSS, KAPPA, OR, ORSS, CSI, ETS, BOYCE, MPA *Note that this is for binary data only.*

`metric.filter` (*optional, default* `NULL`)
A `vector` containing evaluation metric names to be used to transform prediction values into filtered values based on models evaluation scores obtained with the `BIOMOD_EnsembleModeling` function. Must be among `all` (same evaluation metrics than those of `bm.mod`) or `POD`, `FAR`, `POFD`, `SR`, `ACCURACY`, `BIAS`, `AUCroc`, `AUCprg`, `TSS`, `KAPPA`, `OR`, `ORSS`, `CSI`, `ETS`, `BOYCE`, `MPA`
Note that this is for binary data only.

`na.rm` (*optional, default* `TRUE`)
A `logical` value defining whether ensemble model projection should ignore missing values in single model projections or not (*ignored by* `EMwmean` *algorithm*)

`nb.cpu` (*optional, default* `1`)
An `integer` value corresponding to the number of computing resources to be used to parallelize the single models computatio

`...` (*optional, see Details*)

Details

`...` can take the following values :

digits (*optional, default* `0`) :
an `integer` value corresponding to the number of digits of the predictions

on_0_1000 (*optional, default* `TRUE`) :
a `logical` value defining whether `0 - 1` probabilities are to be converted to `0 - 1000` scale to save memory on backup

keep.in.memory (*optional, default* `TRUE`) :
a `logical` value defining whether all projections are to be kept loaded at once in memory, or only links pointing to hard drive are to be returned

do.stack (*optional, default* `TRUE`) :
a `logical` value defining whether all projections are to be saved as one `SpatRaster` object or several `SpatRaster` files (*the default if projections are too heavy to be all loaded at once in memory*)

output.format (*optional, default* `.RData` or `.tif`) :
a `character` value corresponding to the projections saving format on hard drive, must be either `.grd`, `.img`, `.tif` or `.RData` (the default if `new.env` is given as `matrix` or `data.frame`)

compress (*optional, default* `TRUE`) :
a `logical` or a `character` value defining whether and how objects should be compressed when saved on hard drive. Must be either `TRUE`, `FALSE`, `gzip` (for Windows OS) or `xz` (for other OS)

Value

A `BIOMOD.projection.out` object containing models projections, or links to saved outputs.
Models projections are stored out of R (for memory storage reasons) in `proj.name` folder created in the current working directory :

1. the output is a `data.frame` if `new.env` is a `matrix` or a `data.frame`

2. it is a SpatRaster if new.env is a SpatRaster (or several SpatRaster objects, if new.env is too large)
3. raw projections, as well as binary and filtered projections (if asked), are saved in the proj.name folder

Author(s)

Wilfried Thuiller, Damien Georges, Robin Engler

See Also

BIOMOD_FormatingData, bm_ModelingOptions, BIOMOD_Modeling, BIOMOD_EnsembleModeling, BIOMOD_RangeSize

Other Main functions: BIOMOD_EnsembleModeling(), BIOMOD_FormatingData(), BIOMOD_LoadModels(), BIOMOD_Modeling(), BIOMOD_Projection(), BIOMOD_RangeSize()

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)




# ---------------------------------------------------------------- #
file.out <- paste0(myRespName, "/", myRespName, ".AllModels.models.out")
if (file.exists(file.out)) {
  myBiomodModelOut <- get(load(file.out))
} else {

  # Format Data with true absences
  myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                       resp.var = myResp,
                                       resp.xy = myRespXY,
                                       expl.var = myExpl)
```

```
  # Model single models
  myBiomodModelOut <- BIOMOD_Modeling(bm.format = myBiomodData,
                                      modeling.id = 'AllModels',
                                      models = c('RF', 'GLM'),
                                      CV.strategy = 'random',
                                      CV.nb.rep = 2,
                                      CV.perc = 0.8,
                                      OPT.strategy = 'bigboss',
                                      metric.eval = c('TSS', 'AUCroc'),
                                      var.import = 3,
                                      seed.val = 42)
}


file.proj <- paste0(myRespName, "/proj_Current/", myRespName, ".Current.projection.out")
if (file.exists(file.proj)) {
  myBiomodProj <- get(load(file.proj))
} else {

  # Project single models
  myBiomodProj <- BIOMOD_Projection(bm.mod = myBiomodModelOut,
                                    proj.name = 'Current',
                                    new.env = myExpl,
                                    models.chosen = 'all',
                                    build.clamping.mask = TRUE)
}


file.EM <- paste0(myRespName, "/", myRespName, ".AllModels.ensemble.models.out")
if (file.exists(file.EM)) {
  myBiomodEM <- get(load(file.EM))
} else {

  # Model ensemble models
  myBiomodEM <- BIOMOD_EnsembleModeling(bm.mod = myBiomodModelOut,
                                        models.chosen = 'all',
                                        em.by = 'all',
                                        em.algo = c('EMmean', 'EMca'),
                                        metric.select = c('TSS'),
                                        metric.select.thresh = c(0.7),
                                        metric.eval = c('TSS', 'AUCroc'),
                                        var.import = 3,
                                        seed.val = 42)
}


# ---------------------------------------------------------------- #
# Project ensemble models (from single projections)
myBiomodEMProj <- BIOMOD_EnsembleForecasting(bm.em = myBiomodEM,
                                             bm.proj = myBiomodProj,
                                             models.chosen = 'all',
                                             metric.binary = 'all',
```

```
                                            metric.filter = 'all')

# Project ensemble models (building single projections)
myBiomodEMProj <- BIOMOD_EnsembleForecasting(bm.em = myBiomodEM,
                                            proj.name = 'CurrentEM',
                                            new.env = myExpl,
                                            models.chosen = 'all',
                                            metric.binary = 'all',
                                            metric.filter = 'all')
myBiomodEMProj
plot(myBiomodEMProj)
```

BIOMOD_EnsembleModeling *Create and evaluate an ensemble set of models and predictions*

Description

This function allows to combine a range of models built with the BIOMOD_Modeling function in one (or several) ensemble model. Modeling uncertainty can be assessed as well as variables importance, ensemble predictions can be evaluated against original data, and created ensemble models can be projected over new conditions (see Details).

Usage

```
BIOMOD_EnsembleModeling(
  bm.mod,
  models.chosen = "all",
  em.by = "PA+run",
  em.algo,
  metric.select = "all",
  metric.select.thresh = NULL,
  metric.select.table = NULL,
  metric.select.dataset = NULL,
  metric.eval = c("KAPPA", "TSS", "AUCroc"),
  var.import = 0,
  EMci.alpha = 0.05,
  EMwmean.decay = "proportional",
  nb.cpu = 1,
  seed.val = NULL,
  do.progress = TRUE
)
```

Arguments

bm.mod	a BIOMOD.models.out object returned by the BIOMOD_Modeling function

models.chosen a vector containing model names to be kept, must be either all or a sub-selection of model names that can be obtained with the get_built_models function applied to bm.mod

em.by a character corresponding to the way kept models will be combined to build the ensemble models, must be among all, algo, PA, PA+algo, PA+run

em.algo a vector corresponding to the ensemble models that will be computed, must be among EMmean, EMmedian, EMcv, EMci, EMca, EMwmean, EMmode, EMfreq

metric.select a vector containing evaluation metric names to be used to select single models based on their evaluation scores, must be among user.defined or AUCroc, AUCprg, TSS, KAPPA, ACCURACY, BIAS, POD, FAR, POFD, SR, CSI, ETS, OR, ORSS, BOYCE, MPA (*binary data*), RMSE, MAE, MSE, Rsquared, Rsquared_aj, Max_error (*abundance / count / relative data*), Accuracy, Recall, Precision, F1 (*ordinal data*)

metric.select.thresh (*optional, default* NULL)
A vector of numeric values corresponding to the minimum scores (one for each metric.select) below which single models will be excluded from the ensemble model building

metric.select.table (*optional, default* NULL)
If metric.select = 'user.defined', a data.frame containing evaluation scores calculated for each single models and that will be compared to metric.select.thresh values below which single models will be excluded from the ensemble model, with metric.select rownames, and models.chosen colnames

metric.select.dataset (*optional, default* validation if possible)
A character defining which dataset should be used to filter and/or weight the ensemble models, must be among calibration, validation, evaluation

metric.eval a vector containing evaluation metric names to be used, must be among AUCroc, AUCprg, TSS, KAPPA, ACCURACY, BIAS, POD, FAR, POFD, SR, CSI, ETS, OR, ORSS, BOYCE, MPA (*binary data*), RMSE, MAE, MSE, Rsquared, Rsquared_aj, Max_error (*abundance / count / relative data*), Accuracy, Recall, Precision, F1 (*ordinal data*)

var.import (*optional, default* NULL)
An integer corresponding to the number of permutations to be done for each variable to estimate variable importance

EMci.alpha (*optional, default* 0.05)
A numeric value corresponding to the significance level to estimate confidence interval

EMwmean.decay (*optional, default* proportional)
If em.algo = 'EMWmean', a numeric value defining the relative importance of weights. A high value will strongly discriminate *good* models from the bad ones (see Details). It is also possible to set it to proportional and weights will be proportional to the single models evaluation scores, or to provide a function.

nb.cpu (*optional, default* 1)
An integer value corresponding to the number of computing resources to be

used to parallelize the single models predictions and the ensemble models computation

`seed.val` (*optional, default* `NULL`)
An `integer` value corresponding to the new seed value to be set

`do.progress` (*optional, default* `TRUE`)
A `logical` value defining whether the progress bar is to be rendered or not

Details

Concerning models sub-selection (`models.chosen`) :
Applying `get_built_models` function to the `bm.mod` object gives the names of the single models created with the `BIOMOD_Modeling` function. The `models.chosen` argument can take either a sub-selection of these single model names, or the `all` default value, to decide which single models will be used for the ensemble model building.

Concerning models assembly rules (`em.by`) :
Single models built with the `BIOMOD_Modeling` function can be combined in 5 different ways to obtain ensemble models :

PA+run each combination of pseudo-absence and repetition datasets is done, *merging* algorithms together

PA+algo each combination of pseudo-absence and algorithm datasets is done, *merging* repetitions together

PA pseudo-absence datasets are considered individually, *merging* algorithms and repetitions together

algo algorithm datasets are considered individually, *merging* pseudo-absence and repetitions together

all all single models are combined into one

Hence, depending on the chosen method, the number of ensemble models built will vary.
If no evaluation data was given to the `BIOMOD_FormatingData` function, some ensemble model evaluations may be biased due to difference in data used for single model evaluations.
Be aware that all of these combinations are allowed, but some may not make sense depending mainly on how pseudo-absence datasets have been built and whether all of them have been used for all single models or not (see `PA.nb.absences` and `models.pa` parameters in `BIOMOD_FormatingData` and `BIOMOD_Modeling` functions respectively).

Concerning evaluation metrics :

metric.select metric(s) must be chosen among the ones used within the `BIOMOD_Modeling` function to build the `bm.mod` object, unless `metric.select = 'user.defined'` and therefore values will be provided through the `metric.select.table` parameter.
Each selected metric will be used at different steps of the ensemble modeling function to :

1. remove *low quality* single models having a score lower than `metric.select.thresh`

2. perform the binary transformation if em.algo = 'EMca'
3. weight models if em.algo = 'EMwmean'

Note that metrics are not combined together, and one ensemble model is built for each metric provided.

metric.select.table if metric.select = 'user.defined', this parameter allows to use evaluation metrics other than those calculated within **biomod2**. It must be a data.frame containing as many columns as models.chosen with matching names, and as many rows as evaluation metrics to be used. The number of rows must match the length of metric.select.thresh, and values will be compared to those defined in metric.select.thresh to remove *low quality* single models from the ensemble model building.

metric.select.dataset by default, *validation* datasets will be used, unless no validation is available (no cross-validation) in which case *calibration* datasets will be used

Concerning ensemble algorithms :
6 modeling techniques are currently available :

EMmedian median of probabilities over the selected models

Less sensitive to outliers than the mean

EMmean mean of probabilities over the selected models

EMwmean weighted mean of probabilities over the selected models

Probabilities are weighted according to their model evaluation scores obtained when building the bm.out object with the BIOMOD_Modeling function (*better a model is, more importance it has in the ensemble*) and summed.

The EMwmean.decay is the ratio between a weight and the next or previous one.
The formula is : W = W(-1) * EMwmean.decay.
For example, with the value of 1.6 *and* 4 *weights wanted, the relative importance of the weights will be* 1 / 1.6 / 2.56 (=1.6*1.6) / 4.096 (=2.56*1.6) *from the weakest to the strongest, and gives* 0.11 / 0.17 / 0.275 / 0.445 *considering that the sum of the weights is equal to one. The lower the* EMwmean.decay*, the smoother the differences between the weights enhancing a weak discrimination between models.*

If EMwmean.decay = 'proportional', the weights are assigned to each model proportionally to their evaluation scores. The discrimination is fairer than using the *decay* method where close scores can have strongly diverging weights, while the proportional method would assign them similar weights.

It is also possible to define the EMwmean.decay parameter as a function that will be applied to single models scores and transform them into weights.
For example, if EMwmean.decay = function(x) {x^2}*, the squared of evaluation score of each model will be used to weight the models predictions.*

EMca committee averaging over the selected models

Probabilities are first transformed into binary data according to the threshold defined when building the bm.out object with the BIOMOD_Modeling function (maximizing the evaluation

metric score over the *calibration* dataset). The committee averaging score is obtained by taking the average of these binary predictions.
It is built on the analogy of a simple vote :

- each single model votes for the species being either present (`1`) or absent (`0`)
- the sum of `1` is then divided by the number of single models *voting*

The interesting feature of this measure is that it gives both a prediction and a measure of uncertainty. When the prediction is close to `0` or `1`, it means that all models agree to predict `0` or `1` respectively. When the prediction is around `0.5`, it means that half the models predict `1` and the other half `0`.
Note that this is for binary data only.

EMci confidence interval around the mean of probabilities of the selected models

It creates 2 *ensemble* models :

- *LOWER* : there is less than `100 * EMci.alpha / 2` % of chance to get probabilities lower than the given ones
- *UPPER* : there is less than `100 * EMci.alpha / 2` % of chance to get probabilities upper than the given ones

These intervals are calculated with the following function :

$$I_c = [\bar{x} - \frac{t_\alpha sd}{\sqrt{n}}; \bar{x} + \frac{t_\alpha sd}{\sqrt{n}}]$$

EMcv coefficient of variation (`sd / mean`) of probabilities over the selected models

This is the only *ensemble* model that might not be over the same scale than the others, as CV is a measure of uncertainty rather a measure of probability of occurrence. It will be evaluated like all other ensemble models although its interpretation will be obviously different. If the CV gets a high evaluation score, it means that the uncertainty is high where the species is observed (which might not be a good feature of the model). *The lower is the score, the better are the models.*

EMmode mode of the predictions over the selected models

For multiclass and ordinal data, EMmode will return the most frequent class found for each point. This is the only *ensemble* model that will return categorical data and not numeric values.

EMfreq mode frequency of the predictions over the selected models

For multiclass and ordinal data, EMfreq will return the frequency of the mode found for each point. This is a way of assessing the uncertainty between models: the higher the frequency, the lower the uncertainty.

Value

A `BIOMOD.ensemble.models.out` object containing models outputs, or links to saved outputs.
Models outputs are stored out of R (for memory storage reasons) in 2 different folders created in the current working directory :

1. a *models* folder, named after the `resp.name` argument of `BIOMOD_FormatingData`, and containing all ensemble models
2. a *hidden* folder, named `.BIOMOD_DATA`, and containing outputs related files (original dataset, calibration lines, pseudo-absences selected, predictions, variables importance, evaluation values...), that can be retrieved with `get_[...]` or `load` functions, and used by other **biomod2** functions, like `BIOMOD_EnsembleForecasting`

Author(s)

Wilfried Thuiller, Damien Georges, Robin Engler

See Also

`BIOMOD_FormatingData`, `bm_ModelingOptions`, `bm_CrossValidation`, `bm_VariablesImportance`, `BIOMOD_Modeling`, `BIOMOD_EnsembleForecasting`, `bm_PlotEvalMean`, `bm_PlotEvalBoxplot`, `bm_PlotVarImpBoxplot`, `bm_PlotResponseCurves`

Other Main functions: `BIOMOD_EnsembleForecasting()`, `BIOMOD_FormatingData()`, `BIOMOD_LoadModels()`, `BIOMOD_Modeling()`, `BIOMOD_Projection()`, `BIOMOD_RangeSize()`

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



## ------------------------------------------------------------------------ #
file.out <- paste0(myRespName, "/", myRespName, ".AllModels.models.out")
if (file.exists(file.out)) {
  myBiomodModelOut <- get(load(file.out))
} else {

  # Format Data with true absences
  myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                       resp.var = myResp,
```

```
                                      resp.xy = myRespXY,
                                      expl.var = myExpl)

  # Model single models
  myBiomodModelOut <- BIOMOD_Modeling(bm.format = myBiomodData,
                                      modeling.id = 'AllModels',
                                      models = c('RF', 'GLM'),
                                      CV.strategy = 'random',
                                      CV.nb.rep = 2,
                                      CV.perc = 0.8,
                                      OPT.strategy = 'bigboss',
                                      metric.eval = c('TSS', 'AUCroc'),
                                      var.import = 3,
                                      seed.val = 42)
}

## ------------------------------------------------------------------------ #
# Model ensemble models
myBiomodEM <- BIOMOD_EnsembleModeling(bm.mod = myBiomodModelOut,
                                      models.chosen = 'all',
                                      em.by = 'all',
                                      em.algo = c('EMmean', 'EMca'),
                                      metric.select = c('TSS'),
                                      metric.select.thresh = c(0.7),
                                      metric.eval = c('TSS', 'AUCroc'),
                                      var.import = 3,
                                      seed.val = 42)
myBiomodEM

# Get evaluation scores & variables importance
get_evaluations(myBiomodEM)
get_variables_importance(myBiomodEM)

# Represent evaluation scores
bm_PlotEvalMean(bm.out = myBiomodEM, dataset = 'calibration')
bm_PlotEvalBoxplot(bm.out = myBiomodEM, group.by = c('algo', 'algo'))

# # Represent variables importance
# bm_PlotVarImpBoxplot(bm.out = myBiomodEM, group.by = c('expl.var', 'algo', 'algo'))
# bm_PlotVarImpBoxplot(bm.out = myBiomodEM, group.by = c('expl.var', 'algo', 'merged.by.PA'))
# bm_PlotVarImpBoxplot(bm.out = myBiomodEM, group.by = c('algo', 'expl.var', 'merged.by.PA'))

# # Represent response curves
# bm_PlotResponseCurves(bm.out = myBiomodEM,
#                       models.chosen = get_built_models(myBiomodEM),
#                       fixed.var = 'median')
# bm_PlotResponseCurves(bm.out = myBiomodEM,
#                       models.chosen = get_built_models(myBiomodEM),
#                       fixed.var = 'min')
# bm_PlotResponseCurves(bm.out = myBiomodEM,
#                       models.chosen = get_built_models(myBiomodEM, algo = 'EMmean'),
#                       fixed.var = 'median',
#                       do.bivariate = TRUE)
```

`BIOMOD_FormatingData` *Format input data, and select pseudo-absences if wanted, for usage in* **biomod2**

Description

This function gathers together all input data needed (*xy, presences/absences, explanatory variables, and the same for evaluation data if available*) to run **biomod2** models. It allows to select pseudo-absences if no absence data is available, with different strategies (see Details).

Usage

```
BIOMOD_FormatingData(
  resp.name,
  resp.var,
  resp.xy = NULL,
  expl.var,
  dir.name = ".",
  data.type = "binary",
  eval.resp.var = NULL,
  eval.resp.xy = NULL,
  eval.expl.var = NULL,
  PA.strategy = NULL,
  PA.nb.rep = NULL,
  PA.nb.absences = NULL,
  PA.dist.min = 0,
  PA.dist.max = NULL,
  PA.sre.quant = NULL,
  PA.fact.aggr = NULL,
  PA.user.table = NULL,
  na.rm = TRUE,
  filter.raster = FALSE,
  seed.val = NULL
)
```

Arguments

`resp.name`	a `character` corresponding to the species name
`resp.var`	a `vector`, a `SpatVector` without associated data (*if presence-only*), or a `SpatVector` object containing binary data (`1` : presence, `0` : absence, `NA` : indeterminate) or other data (see `data.type` and Details) for a single species that will be used to build the species distribution model(s) *Note that old format from* **sp** *are still supported such as* `SpatialPoints` *(if presence-only) or* `SpatialPointsDataFrame` *object containing binary data or other data.*

`resp.xy` (*optional, default* `NULL`)
If `resp.var` is a `vector`, a 2-columns `matrix` or `data.frame` containing the corresponding `X` and `Y` coordinates that will be used to build the species distribution model(s)

`expl.var` a `matrix`, `data.frame`, `SpatVector` or `SpatRaster` object containing the explanatory variables (in columns or layers) that will be used to build the species distribution model(s)
Note that old format from **raster** *and* **sp** *are still supported such as* `RasterStack` *and* `SpatialPointsDataFrame` *objects.*

`dir.name` (*optional, default* `.`)
A `character` corresponding to the modeling folder

`data.type` a `character`, corresponding to the response data type to be used, must be either `binary`, `count`, `multiclass`, `ordinal`, `relative`, or `abundance`, and match the data contained in `resp.var`
If not provided, `biomod2` *will try to guess.*

`eval.resp.var` (*optional, default* `NULL`)
A `vector` or a `SpatVector` object containing binary data (`1` : presence, `0` : absence) for a single species that will be used to evaluate the species distribution model(s) with independent data
Note that old format from **sp** *are still supported such as* `SpatialPoints` *(if presence-only) or* `SpatialPointsDataFrame` *object containing binary data.*

`eval.resp.xy` (*optional, default* `NULL`)
If `resp.var` is a `vector`, a 2-columns `matrix` or `data.frame` containing the corresponding `X` and `Y` coordinates that will be used to evaluate the species distribution model(s) with independent data

`eval.expl.var` (*optional, default* `NULL`)
A `matrix`, `data.frame`, `SpatVector` or `SpatRaster` object containing the explanatory variables (in columns or layers) that will be used to evalute the species distribution model(s) with independent data.
Note that old format from **raster** *and* **sp** *are still supported such as* `RasterStack` *and* `SpatialPointsDataFrame` *objects.*

`PA.strategy` (*optional, default* `NULL`)
If pseudo-absence selection, a `character` defining the strategy that will be used to select the pseudo-absence points. Must be `random`, `sre`, `disk` or `user.defined` (see Details)

`PA.nb.rep` (*optional, default* `NULL`)
If pseudo-absence selection, an `integer` corresponding to the number of sets (repetitions) of pseudo-absence points that will be drawn

`PA.nb.absences` (*optional, default* `NULL`)
If pseudo-absence selection, and `PA.strategy = 'random'` or `PA.strategy = 'sre'` or `PA.strategy = 'disk'`, an `integer` corresponding to the number of pseudo-absence points that will be selected for each pseudo-absence repetition (true absences included).
It can also be a `vector` of the same length as `PA.nb.rep` containing `integer` values corresponding to the different numbers of pseudo-absences to be selected (see Details)

`PA.dist.min` (*optional, default* `0`)
If pseudo-absence selection and `PA.strategy = 'disk'`, a `numeric` defining the minimal distance to presence points used to make the `disk` pseudo-absence selection (in the same projection system units as `resp.xy` and `expl.var`, see Details)

`PA.dist.max` (*optional, default* `NULL`)
If pseudo-absence selection and `PA.strategy = 'disk'`, a `numeric` defining the maximal distance to presence points used to make the `disk` pseudo-absence selection (in the same projection system units as `resp.xy` and `expl.var`, see Details)

`PA.sre.quant` (*optional, default* `NULL`)
If pseudo-absence selection and `PA.strategy = 'sre'`, a `numeric` between `0` and `0.5` defining the half-quantile used to make the `sre` pseudo-absence selection (see Details)

`PA.fact.aggr` (*optional, default* `NULL`)
If pseudo-absence selection and `PA.strategy = 'random'` or `PA.strategy = 'disk'`, an `integer` defining the factor of aggregation to reduce the spatial resolution of the environmental variables

`PA.user.table` (*optional, default* `NULL`)
If pseudo-absence selection and `PA.strategy = 'user.defined'`, a `matrix` or `data.frame` with as many rows as `resp.var` values, as many columns as `PA.nb.rep`, and containing `TRUE` or `FALSE` values defining which points will be used to build the species distribution model(s) for each repetition (see Details)

`na.rm` (*optional, default* `TRUE`)
A `logical` value defining whether points having one or several missing values for explanatory variables should be removed from the analysis or not

`filter.raster` (*optional, default* `FALSE`)
If `expl.var` is of raster type, a `logical` value defining whether `resp.var` is to be filtered when several points occur in the same raster cell

`seed.val` (*optional, default* `NULL`)
An `integer` value corresponding to the new seed value to be set

Details

This function gathers and formats all input data needed to run **biomod2** models. It supports different kind of inputs (e.g. `matrix`, `SpatVector`, `SpatRaster`) and provides different methods to select pseudo-absences if needed.

Concerning explanatory variables and XY coordinates :

- if `SpatRaster`, `RasterLayer` or `RasterStack` provided for `expl.var` or `eval.expl.var`, **biomod2** will extract the corresponding values from XY coordinates provided :
  - either through `resp.xy` or `eval.resp.xy` respectively
  - or through `resp.var` or `eval.resp.var`, if provided as `SpatVector` or `SpatialPointsDataFrame`

Be sure to give the objects containing XY coordinates in the same projection system than the raster objects !

- if `data.frame` or `matrix` provided for `expl.var` or `eval.expl.var`,
  biomod2 will simply merge it (`cbind`) with `resp.var` without considering XY coordinates.
  Be sure to give explanatory and response values in the same row order !

Concerning pseudo-absence selection (see `bm_PseudoAbsences`) :
Only in the case of `binary` *data !*

- if both presence and absence data are available : `PA.nb.rep = 0` and no pseudo-absence will be selected.
- if no absence data is available, several pseudo-absence repetitions are recommended (to estimate the effect of pseudo-absence selection), as well as high number of pseudo-absence points.
  Be sure not to select more pseudo-absence points than maximum number of pixels in the studied area !
- it is possible to create several pseudo-absence repetitions *with different number of points*, BUT with the same sampling strategy. `PA.nb.absences` must contain as many values as the number of sets of pseudo-absences (`PA.nb.rep`).

Response variable **biomod2** models single species at a time (no multi-species).
Hence, `resp.var` must be an uni-dimensional object, either :

- a `vector`, a one-column `matrix` or `data.frame`, a `SpatVector` (*without associated data - if presence-only*)
- a `SpatialPoints` (*if presence-only*)
- a `SpatialPointsDataFrame` or `SpatVector` object

If `resp.var` is a non-spatial object (`vector`, `matrix` or `data.frame`), XY coordinates must be provided through `resp.xy`.

Different data types are available, and require different values :

binary `1` : presences, `0` : true absences or `NA` : no information point (can be used to select pseudo-absences)
If no true absences are available, pseudo-absence selection must be done.

count positive `integer` values

multiclass `factor` values

ordinal ordered `factor` values

relative `numeric` values between `0` and `1`

abundance positive `numeric` values

Explanatory variables Factorial variables are allowed, but might lead to some pseudo-absence strategy or models omissions (e.g. `sre`).

Evaluation data Although **biomod2** provides tools to automatically divide dataset into calibration and validation parts through the modeling process (see `CV.[..]` parameters in `BIOMOD_Modeling` function ; or `bm_CrossValidation` function), it is also possible (and strongly advised) to directly provide two independent datasets, one for calibration/validation and one for evaluation

Pseudo-absence selection (see `bm_PseudoAbsences`) *Only in the case of* `binary` *data !*
If no true absences are available, pseudo-absences must be selected from the *background data*, meaning data there is no information whether the species of interest occurs or not. It corresponds either to the remaining pixels of the `expl.var` (if provided as a `SpatRaster` or `RasterStack`) or to the points identified as `NA` in `resp.var` (if `expl.var` provided as a `matrix` or `data.frame`).

Several methods are available to do this selection :

random all points of initial background are pseudo-absence candidates. `PA.nb.absences` are drawn randomly, for each `PA.nb.rep` requested.

sre pseudo-absences have to be selected in conditions (combination of explanatory variables) that differ in a defined proportion (`PA.sre.quant`) from those of presence points. A *Surface Range Envelop* model is first run over the species of interest (see `bm_SRE`), and pseudo-absences are selected outside this envelop.
This case is appropriate when all the species climatic niche has been sampled, otherwise it may lead to over-optimistic model evaluations and predictions !

disk pseudo-absences are selected within circles around presence points defined by `PA.dist.min` and `PA.dist.max` distance values (in the same projection system units as `coord` and `expl.var`). It allows to select pseudo-absence points that are not too close to (avoid same niche and pseudo-replication) or too far (localized sampling strategy) from presences.

user.defined pseudo-absences are defined in advance and given as `data.frame` through the `PA.user.table` parameter.

Value

A `BIOMOD.formated.data` or `BIOMOD.formated.data.PA` object that can be used to build species distribution model(s) with the `BIOMOD_Modeling` function.
`print/show`, `plot` and `summary` functions are available to have a summary of the created object.

Author(s)

Damien Georges, Wilfried Thuiller

See Also

`bm_PseudoAbsences`, `BIOMOD_Modeling`

Other Main functions: `BIOMOD_EnsembleForecasting()`, `BIOMOD_EnsembleModeling()`, `BIOMOD_LoadModels()`, `BIOMOD_Modeling()`, `BIOMOD_Projection()`, `BIOMOD_RangeSize()`

Examples

```
library(terra)

# Load species occurrences (6 species available)
```

```
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



# -------------------------------------------------------------#
# Format Data with true absences
myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                     resp.var = myResp,
                                     resp.xy = myRespXY,
                                     expl.var = myExpl)
myBiomodData
summary(myBiomodData)
plot(myBiomodData)


# -------------------------------------------------------------#
# # Transform true absences into potential pseudo-absences
# myResp.PA <- ifelse(myResp == 1, 1, NA)
#
# # Format Data with pseudo-absences : random method
# myBiomodData.r <- BIOMOD_FormatingData(resp.name = myRespName,
#                                        resp.var = myResp.PA,
#                                        resp.xy = myRespXY,
#                                        expl.var = myExpl,
#                                        PA.nb.rep = 4,
#                                        PA.nb.absences = 1000,
#                                        PA.strategy = 'random')
#
# # Format Data with pseudo-absences : disk method
# myBiomodData.d <- BIOMOD_FormatingData(resp.name = myRespName,
#                                        resp.var = myResp.PA,
#                                        resp.xy = myRespXY,
#                                        expl.var = myExpl,
#                                        PA.nb.rep = 4,
#                                        PA.nb.absences = 500,
#                                        PA.strategy = 'disk',
#                                        PA.dist.min = 5,
#                                        PA.dist.max = 35)
#
```

```
# # Format Data with pseudo-absences : SRE method
# myBiomodData.s <- BIOMOD_FormatingData(resp.name = myRespName,
#                                        resp.var = myResp.PA,
#                                        resp.xy = myRespXY,
#                                        expl.var = myExpl,
#                                        PA.nb.rep = 4,
#                                        PA.nb.absences = 1000,
#                                        PA.strategy = 'sre',
#                                        PA.sre.quant = 0.025)
#
# # Format Data with pseudo-absences : user.defined method
# myPAtable <- data.frame(PA1 = ifelse(myResp == 1, TRUE, FALSE),
#                         PA2 = ifelse(myResp == 1, TRUE, FALSE))
# for (i in 1:ncol(myPAtable)) myPAtable[sample(which(myPAtable[, i] == FALSE), 500), i] = TRUE
# myBiomodData.u <- BIOMOD_FormatingData(resp.name = myRespName,
#                                        resp.var = myResp.PA,
#                                        resp.xy = myRespXY,
#                                        expl.var = myExpl,
#                                        PA.strategy = 'user.defined',
#                                        PA.user.table = myPAtable)
#
# myBiomodData.r
# myBiomodData.d
# myBiomodData.s
# myBiomodData.u
# plot(myBiomodData.r)
# plot(myBiomodData.d)
# plot(myBiomodData.s)
# plot(myBiomodData.u)


# ----------------------------------------------------------------#
# # Select multiple sets of pseudo-absences
#
# # Transform true absences into potential pseudo-absences
# myResp.PA <- ifelse(myResp == 1, 1, NA)
#
# # Format Data with pseudo-absences : random method
# myBiomodData.multi <- BIOMOD_FormatingData(resp.name = myRespName,
#                                            resp.var = myResp.PA,
#                                            resp.xy = myRespXY,
#                                            expl.var = myExpl,
#                                            PA.nb.rep = 4,
#                                            PA.nb.absences = c(1000, 500, 500, 200),
#                                            PA.strategy = 'random')
# myBiomodData.multi
# summary(myBiomodData.multi)
# plot(myBiomodData.multi)
```

BIOMOD_LoadModels *Load species distribution models built with* **biomod2**

Description

This function loads individual models built with BIOMOD_Modeling or BIOMOD_EnsembleModeling functions.

Usage

```
BIOMOD_LoadModels(
  bm.out,
  full.name = NULL,
  PA = NULL,
  run = NULL,
  algo = NULL,
  merged.by.PA = NULL,
  merged.by.run = NULL,
  merged.by.algo = NULL,
  filtered.by = NULL
)
```

Arguments

bm.out	a BIOMOD.models.out or BIOMOD.ensemble.models.out object that can be obtained with the BIOMOD_Modeling or BIOMOD_EnsembleModeling functions
full.name	(*optional, default* NULL) A vector containing model names to be kept, must be either all or a sub-selection of model names that can be obtained with the get_built_models function applied to bm.out
PA	(*optional, default* NULL) A vector containing pseudo-absence set to be loaded, must be among PA1, PA2, ..., allData
run	(*optional, default* NULL) A vector containing repetition set to be loaded, must be among RUN1, RUN2, ..., allRun
algo	(*optional, default* NULL) A character containing algorithm to be loaded, must be either ANN, CTA, FDA, GAM, GBM, GLM, MARS, MAXENT, MAXNET, RF, RFd, SRE, XGBOOST
merged.by.PA	(*optional, default* NULL) A vector containing merged pseudo-absence set to be loaded, must be among PA1, PA2, ..., mergedData
merged.by.run	(*optional, default* NULL) A vector containing merged repetition set to be loaded, must be among RUN1, RUN2, ..., mergedRun

merged.by.algo (*optional, default* NULL)
A character containing merged algorithm to be loaded, must be among ANN, CTA, FDA, GAM, GBM, GLM, MARS, MAXENT, MAXNET, RF, RFd, SRE, XGBOOST, mergedAlgo

filtered.by (*optional, default* NULL)
A vector containing evaluation metric selected to filter single models to build the ensemble models, must be among AUCroc, AUCprg, TSS, KAPPA, ACCURACY, BIAS, POD, FAR, POFD, SR, CSI, ETS, OR, ORSS, BOYCE, MPA (*binary data*), RMSE, MAE, MSE, Rsquared, Rsquared_aj, Max_error (*abundance / count / relative data*), Accuracy, Recall, Precision, F1 (*ordinal data*)

Details

This function might be of particular use to load models and make response plot analyses.

Running the function providing only bm.out argument will load all models built by the BIOMOD_Modeling or BIOMOD_EnsembleModeling function, but a subselection of models can be done using the additional arguments (full.name, PA, run, algo, merged.by.PA, merged.by.run, merged.by.algo, filtered.by).

Value

A vector containing the names of the loaded models.

Author(s)

Damien Georges

See Also

BIOMOD_Modeling, BIOMOD_EnsembleModeling

Other Main functions: BIOMOD_EnsembleForecasting(), BIOMOD_EnsembleModeling(), BIOMOD_FormatingData(), BIOMOD_Modeling(), BIOMOD_Projection(), BIOMOD_RangeSize()

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]
```

```
# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



# -------------------------------------------------------------
file.out <- paste0(myRespName, "/", myRespName, ".AllModels.models.out")
if (file.exists(file.out)) {
  myBiomodModelOut <- get(load(file.out))
} else {

  # Format Data with true absences
  myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                       resp.var = myResp,
                                       resp.xy = myRespXY,
                                       expl.var = myExpl)

  # Model single models
  myBiomodModelOut <- BIOMOD_Modeling(bm.format = myBiomodData,
                                      modeling.id = 'AllModels',
                                      models = c('RF', 'GLM'),
                                      CV.strategy = 'random',
                                      CV.nb.rep = 2,
                                      CV.perc = 0.8,
                                      OPT.strategy = 'bigboss',
                                      metric.eval = c('TSS','AUCroc'),
                                      var.import = 3,
                                      seed.val = 42)
}


# -------------------------------------------------------------
# Loading some models built
BIOMOD_LoadModels(bm.out = myBiomodModelOut, algo = 'RF')
```

BIOMOD_Modeling *Run a range of species distribution models*

Description

This function allows to calibrate and evaluate a range of modeling techniques for a given species distribution. The dataset can be split up in calibration/validation parts, and the predictive power of the different models can be estimated using a range of evaluation metrics (see Details).

Usage

```
BIOMOD_Modeling(
  bm.format,
  modeling.id,
 models = c("ANN", "CTA", "DNN", "FDA", "GAM", "GBM", "GLM", "MARS", "MAXENT", "MAXNET",
    "RF", "RFd", "SRE", "XGBOOST"),
  models.pa = NULL,
  CV.strategy = NULL,
  CV.nb.rep = NULL,
  CV.perc = NULL,
  CV.k = NULL,
  CV.balance = NULL,
  CV.env.var = NULL,
  CV.strat = NULL,
  CV.user.table = NULL,
  CV.do.full.models = NULL,
  OPT.strategy = "default",
  OPT.user.val = NULL,
  OPT.user.base = NULL,
  OPT.user = NULL,
  metric.eval = NULL,
  var.import = 0,
  weights = NULL,
  prevalence = 0.5,
  scale.models = FALSE,
  nb.cpu = 1,
  seed.val = NULL,
  do.progress = TRUE
)
```

Arguments

bm.format	a BIOMOD.formated.data or BIOMOD.formated.data.PA object returned by the BIOMOD_FormatingData function
modeling.id	a character corresponding to the name (ID) of the simulation set (*a random number by default*)
models	a vector containing model names to be computed, must be among ANN, CTA, DNN, FDA, GAM, GBM, GLM, MARS, MAXENT, MAXNET, RF, RFd, SRE, XGBOOST
models.pa	(*optional, default* NULL) A list containing for each model a vector defining which pseudo-absence datasets are to be used, must be among colnames(bm.format@PA.table)
CV.strategy	(*optional, default* NULL) A character corresponding to the cross-validation selection strategy, must be among random, kfold, block, strat, env or user.defined
CV.nb.rep	(*optional, default* NULL) If strategy = 'random' or strategy = 'kfold', an integer corresponding to the number of sets (repetitions) of cross-validation points that will be drawn

`CV.perc` (*optional, default* `NULL`)
If `strategy = 'random'`, a `numeric` between `0` and `1` defining the percentage of data that will be kept for calibration

`CV.k` (*optional, default* `NULL`)
If `strategy = 'kfold'` or `strategy = 'strat'` or `strategy = 'env'`, an `integer` corresponding to the number of partitions

`CV.balance` (*optional, default* `NULL`)
If `strategy = 'strat'` or `strategy = 'env'`, a `character` corresponding to how data will be balanced between partitions, must be either `presences` or `absences`

`CV.env.var` (*optional, default* `NULL`)
If `strategy = 'env'`, a `character` corresponding to the environmental variables used to build the partition (all available variables by default), and for which `CV.k` partitions will be built

`CV.strat` (*optional, default* `NULL`)
If `strategy = 'strat'`, a `character` corresponding to how data will partitioned along gradient, must be among `x`, `y`, `both`

`CV.user.table` (*optional, default* `NULL`)
If `strategy = 'user.defined'`, a `matrix` or `data.frame` defining for each repetition (in columns) which observation lines should be used for models calibration (`TRUE`) and validation (`FALSE`)

`CV.do.full.models` (*optional, default* `NULL`)
A `logical` value defining whether models should be also calibrated and validated over the whole dataset (and pseudo-absence datasets) or not

`OPT.strategy` (*default* `'default'`)
A `character` corresponding to the method to select models' parameters values, must be either `default`, `bigboss`, `user.defined`, `tuned`

`OPT.user.val` (*optional, default* `NULL`)
A `list` containing parameters values for some (all) models

`OPT.user.base` (*optional, default* `NULL`)
A character, `default` or `bigboss` used when `OPT.strategy = 'user.defined'`. It sets the bases of parameters to be modified by user defined values.

`OPT.user` (*optional, default* `NULL`)
A `BIOMOD.models.options` object returned by the `bm_ModelingOptions` function

`metric.eval` a `vector` containing evaluation metric names to be used, must be among `AUCroc`, `AUCprg`, `TSS`, `KAPPA`, `ACCURACY`, `BIAS`, `POD`, `FAR`, `POFD`, `SR`, `CSI`, `ETS`, `OR`, `ORSS`, `BOYCE`, `MPA` (*binary data*), `RMSE`, `MAE`, `MSE`, `Rsquared`, `Rsquared_aj`, `Max_error` (*abundance / count / relative data*), `Accuracy`, `Recall`, `Precision`, `F1` (*multi-class / ordinal data*)

`var.import` (*optional, default* `0`)
An `integer` corresponding to the number of permutations to be done for each variable to estimate variable importance

`weights` (*optional, default* `NULL`)
A `vector` of `numeric` values corresponding to observation weights (one per observation, see Details)

`prevalence` (*optional, default* `0.5`)
A `numeric` between `0` and `1` corresponding to the species prevalence to build '*weighted response weights*' (see Details)

`scale.models` (*optional, default* `FALSE`)
A `logical` value defining whether all models predictions should be scaled with a binomial GLM or not

`nb.cpu` (*optional, default* `1`)
An `integer` value corresponding to the number of computing resources to be used to parallelize the single models computation

`seed.val` (*optional, default* `NULL`)
An `integer` value corresponding to the new seed value to be set

`do.progress` (*optional, default* `TRUE`)
A `logical` value defining whether the progress bar is to be rendered or not

Details

bm.format If pseudo-absences have been added to the original dataset (see `BIOMOD_FormatingData`), `PA.nb.rep *(nb.rep + 1)` models will be created.

models The set of models to be calibrated on the data. 12 modeling techniques are currently available :

- `ANN` : Artificial Neural Network (`nnet`)
- `CTA` : Classification Tree Analysis (`rpart`)
- `DNN` : Deep Neural Network (`cito`)
- `FDA` : Flexible Discriminant Analysis (`fda`)
- `GAM` : Generalized Additive Model (`gam`, `gam` or `bam`)
  (see `bm_ModelingOptions` for details on algorithm selection)
- `GBM` : Generalized Boosting Model, or usually called Boosted Regression Trees (`gbm`)
- `GLM` : Generalized Linear Model (`glm`)
- `MARS` : Multiple Adaptive Regression Splines (`earth`)
- `MAXENT` : Maximum Entropy (see Maxent website)
- `MAXNET` : Maximum Entropy (`maxnet`)
- `RF` : Random Forest (`randomForest`)
- `RFd` : Random Forest downsampled (`randomForest`)
- `SRE` : Surface Range Envelop or usually called BIOCLIM (`bm_SRE`)
- `XGBOOST` : eXtreme Gradient Boosting Training (`xgboost`)

	ANN	CTA	DNN	FDA	GAM	GBM	GLM	MARS	MAXENT	MAXNET
binary	x	x	x	x	x	x	x	x	x	x
multiclass		x	x	x				x		
ordinal		x	x	x	x		x	x		
abundance / count / relative		x	x		x	x	x	x		

models.pa Different models might respond differently to different numbers of pseudo-absences. It is possible to create sets of pseudo-absences with different numbers of points (see `BIOMOD_FormatingData`) and to assign only some of these datasets to each single model.

CV.[... parameters] Different methods are available to calibrate/validate the single models (see `bm_CrossValidation`).

OPT.[... parameters] Different methods are available to parameterize the single models (see `bm_ModelingOptions` and `BIOMOD.options.dataset`).

- `default` : only default parameter values of default parameters of the single models functions are retrieved. Nothing is changed so it might not give good results.
- `bigboss` : uses parameters pre-defined by **biomod2** team and that are available in the dataset `OptionsBigboss`.
  to be optimized in near future
- `user.defined` : updates default or bigboss parameters with some parameters values defined by the user (but matching the format of a `BIOMOD.models.options` object)
- `tuned` : calling the `bm_Tuning` function to try and optimize some default values

metric.eval *Please refer to JWGFVR website ("Forecast verification - methods, issues and FAQ") to get detailed description (simple/complex metrics).*
Several evaluation metrics can be selected.
Optimal value of each method can be obtained with the `get_optim_value` function.

simple
- `POD` : Probability of detection (hit rate)
- `FAR` : False alarm ratio
- `POFD` : Probability of false detection (fall-out)
- `SR` : Success ratio
- `ACCURACY` : Accuracy (fraction correct)
- `BIAS` : Bias score (frequency bias)

complex
- `AUCroc` : Area Under Curve of Relative operating characteristic
- `AUCprg` : Area Under Curve of Precision-Recall-Gain curve
- `TSS` : True skill statistic (Hanssen and Kuipers discriminant, Peirce's skill score)
- `KAPPA` : Cohen's Kappa (Heidke skill score)
- `OR` : Odds Ratio
- `ORSS` : Odds ratio skill score (Yule's Q)
- `CSI` : Critical success index (threat score)
- `ETS` : Equitable threat score (Gilbert skill score)

presence-only
- `BOYCE` : Boyce index
- `MPA` : Minimal predicted area (cutoff optimizing MPA to predict 90% of presences)

abundance / count / relative data
- `RMSE` : Root Mean Square Error
- `MSE` : Mean Square Error
- `MAE` : Mean Absolute Error
- `Rsquared` : R squared
- `Rsquared_aj` : R squared adjusted
- `Max_error` : Maximum error

multiclass/ordinal data
- `Accuracy` : Accuracy
- `Recall` : Macro average Recall

- `Precision` : Macro average Precision
- `F1` : Macro F1 score

Results after modeling can be obtained through the `get_evaluations` function.
Evaluation metric are calculated on the calibrating data (column `calibration`), on the cross-validation data (column `validation`) or on the evaluation data (column `evaluation`).
For cross-validation data, see `CV.[...]` *parameters in* `BIOMOD_Modeling` *function.*
For evaluation data, see `eval.[...]` *parameters in* `BIOMOD_FormatingData`.

var.import A value characterizing how much each variable has an impact on each model predictions can be calculated by randomizing the variable of interest and computing the correlation between original and shuffled variables (see `bm_VariablesImportance`).

weights & prevalence More or less weight can be given to some specific observations.
Automatically created `weights` will be `integer` values to prevent some modeling issues.
Note that `DNN`*,* `MAXENT`*,* `MAXNET` *and* `SRE` *models do not take weights into account.*

- If `prevalence` = `0.5` (the default), presences and absences will be weighted equally (*i.e. the weighted sum of presences equals the weighted sum of absences*).
- If `prevalence` is set below (*above*) `0.5`, more weight will be given to absences (*presences*).
- If `weights` is defined, `prevalence` argument will be ignored (*EXCEPT for* `MAXENT`).

scale.models A binomial GLM is created to scale predictions from 0 to 1.
`SRE` is never scaled, and `ANN` and `FDA` categorical models always are.
Note that it may lead to reduction in projected scale amplitude.
This parameter is quite experimental and it is recommended not to use it. It was developed in the idea to ensure comparable predictions by removing the scale prediction effect (*the more extended projections are, the more they influence ensemble forecasting results*).

Value

A `BIOMOD.models.out` object containing models outputs, or links to saved outputs.
Models outputs are stored out of R (for memory storage reasons) in 2 different folders created in the current working directory :

1. a *models* folder, named after the `resp.name` argument of `BIOMOD_FormatingData`, and containing all calibrated models for each repetition and pseudo-absence run
2. a *hidden* folder, named `.BIOMOD_DATA`, and containing outputs related files (original dataset, calibration lines, pseudo-absences selected, predictions, variables importance, evaluation values...), that can be retrieved with `get_[...]` or `load` functions, and used by other **biomod2** functions, like `BIOMOD_Projection` or `BIOMOD_EnsembleModeling`

Author(s)

Wilfried Thuiller, Damien Georges, Robin Engler

See Also

`glm`, `gam`, `gam`, `bam`, `gbm`, `rpart`, `nnet`, `cito`, `fda`, `earth`, `randomForest`, `maxnet`, `xgboost`, `BIOMOD_FormatingData`, `bm_ModelingOptions`, `bm_Tuning`, `bm_CrossValidation`, `bm_VariablesImportance`,

BIOMOD_Projection, BIOMOD_EnsembleModeling, bm_PlotEvalMean, bm_PlotEvalBoxplot, bm_PlotVarImpBoxplot, bm_PlotResponseCurves

Other Main functions: BIOMOD_EnsembleForecasting(), BIOMOD_EnsembleModeling(), BIOMOD_FormatingData(), BIOMOD_LoadModels(), BIOMOD_Projection(), BIOMOD_RangeSize()

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



# ------------------------------------------------------------------------ #
# Format Data with true absences
myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                     resp.var = myResp,
                                     resp.xy = myRespXY,
                                     expl.var = myExpl)


# ------------------------------------------------------------------------ #
# Model single models
myBiomodModelOut <- BIOMOD_Modeling(bm.format = myBiomodData,
                                    modeling.id = 'AllModels',
                                    models = c('RF', 'GLM'),
                                    CV.strategy = 'random',
                                    CV.nb.rep = 2,
                                    CV.perc = 0.8,
                                    OPT.strategy = 'bigboss',
                                    metric.eval = c('TSS','AUCroc'),
                                    var.import = 2,
                                    seed.val = 42)
myBiomodModelOut

# Get evaluation scores & variables importance
get_evaluations(myBiomodModelOut)
```

```
get_variables_importance(myBiomodModelOut)

# Represent evaluation scores
bm_PlotEvalMean(bm.out = myBiomodModelOut, dataset = 'calibration')
bm_PlotEvalMean(bm.out = myBiomodModelOut, dataset = 'validation')
bm_PlotEvalBoxplot(bm.out = myBiomodModelOut, group.by = c('algo', 'run'))

# # Represent variables importance
# bm_PlotVarImpBoxplot(bm.out = myBiomodModelOut, group.by = c('expl.var', 'algo', 'algo'))
# bm_PlotVarImpBoxplot(bm.out = myBiomodModelOut, group.by = c('expl.var', 'algo', 'run'))
# bm_PlotVarImpBoxplot(bm.out = myBiomodModelOut, group.by = c('algo', 'expl.var', 'run'))

# # Represent response curves
# mods <- get_built_models(myBiomodModelOut, run = 'RUN1')
# bm_PlotResponseCurves(bm.out = myBiomodModelOut,
#                       models.chosen = mods,
#                       fixed.var = 'median')
# bm_PlotResponseCurves(bm.out = myBiomodModelOut,
#                       models.chosen = mods,
#                       fixed.var = 'min')
# mods <- get_built_models(myBiomodModelOut, full.name = 'GuloGulo_allData_RUN2_RF')
# bm_PlotResponseCurves(bm.out = myBiomodModelOut,
#                       models.chosen = mods,
#                       fixed.var = 'median',
#                       do.bivariate = TRUE)
```

`BIOMOD_Projection` *Project a range of calibrated species distribution models onto new environment*

Description

This function allows to project a range of models built with the `BIOMOD_Modeling` function onto new environmental data (*which can represent new areas, resolution or time scales for example*).

Usage

```
BIOMOD_Projection(
  bm.mod,
  proj.name,
  new.env,
  new.env.xy = NULL,
  models.chosen = "all",
  metric.binary = NULL,
  metric.filter = NULL,
  build.clamping.mask = TRUE,
  nb.cpu = 1,
```

```
  seed.val = NULL,
  ...
)
```

Arguments

bm.mod	a BIOMOD.models.out object returned by the BIOMOD_Modeling function
proj.name	a character corresponding to the name (ID) of the projection set (*a new folder will be created within the simulation folder with this name*)
new.env	A matrix, data.frame or SpatRaster object containing the new explanatory variables (in columns or layers, with names matching the variables names given to the BIOMOD_FormatingData function to build bm.mod) that will be used to project the species distribution model(s) *Note that old format from* **raster** *are still supported such as* RasterStack *objects.*
new.env.xy	(*optional, default* NULL) If new.env is a matrix or a data.frame, a 2-columns matrix or data.frame containing the corresponding X and Y coordinates that will be used to project the species distribution model(s)
models.chosen	a vector containing model names to be kept, must be either all or a sub-selection of model names that can be obtained with the get_built_models function applied to bm.mod
metric.binary	(*optional, default* NULL) A vector containing evaluation metric names to be used to transform prediction values into binary values based on models evaluation scores obtained with the BIOMOD_Modeling function. Must be among all (same evaluation metrics than those of bm.mod) or POD, FAR, POFD, SR, ACCURACY, BIAS, AUCroc, AUCprg, TSS, KAPPA, OR, ORSS, CSI, ETS, BOYCE, MPA *Note that this is for binary data only.*
metric.filter	(*optional, default* NULL) A vector containing evaluation metric names to be used to transform prediction values into filtered values based on models evaluation scores obtained with the BIOMOD_Modeling function. Must be among all (same evaluation metrics than those of bm.mod) or POD, FAR, POFD, SR, ACCURACY, BIAS, AUCroc, AUCprg, TSS, KAPPA, OR, ORSS, CSI, ETS, BOYCE, MPA *Note that this is for binary data only.*
build.clamping.mask	(*optional, default* TRUE) A logical value defining whether a clamping mask should be built and saved on hard drive or not (see Details)
nb.cpu	(*optional, default* 1) An integer value corresponding to the number of computing resources to be used to parallelize the single models computation
seed.val	(*optional, default* NULL) An integer value corresponding to the new seed value to be set
...	(*optional, see Details*))

Details

If `models.chosen = 'all'`, projections are done for all calibration and pseudo-absences runs if applicable.
These projections may be used later by the `BIOMOD_EnsembleForecasting` function.

If `build.clamping.mask = TRUE`, a raster file will be saved within the projection folder. This mask values will correspond to the number of variables in each pixel that are out of their calibration / validation range, identifying locations where predictions are uncertain.

... can take the following values :

omit.na (*optional, default* `TRUE`) :
a `logical` value defining whether all not fully referenced environmental points will get `NA` as predictions or not

digits (*optional, default* `0`) :
an `integer` value corresponding to the number of digits of the predictions

on_0_1000 (*optional, default* `TRUE`) :
a `logical` value defining whether `0 - 1` probabilities are to be converted to `0 - 1000` scale to save memory on backup

keep.in.memory (*optional, default* `TRUE`) :
a `logical` value defining whether all projections are to be kept loaded at once in memory, or only links pointing to hard drive are to be returned

do.stack (*optional, default* `TRUE`) :
a `logical` value defining whether all projections are to be saved as one `SpatRaster` object or several `SpatRaster` files (*the default if projections are too heavy to be all loaded at once in memory*)

output.format (*optional, default* `.RData` or `.tif`) :
a `character` value corresponding to the projections saving format on hard drive, must be either `.grd`, `.img`, `.tif` or `.RData` (the default if `new.env` is given as `matrix` or `data.frame`)

compress (*optional, default* `TRUE`) :
a `logical` or a `character` value defining whether and how objects should be compressed when saved on hard drive. Must be either `TRUE`, `FALSE`, `gzip` (for Windows OS) or `xz` (for other OS)

overwrite (*optional, default* `do.stack`) :
a `logical` defining whether pre-existing projections with same modeling ID and project name should be replaced or not

Value

A `BIOMOD.projection.out` object containing models projections, or links to saved outputs.
Models projections are stored out of R (for memory storage reasons) in `proj.name` folder created in the current working directory :

1. the output is a `data.frame` if `new.env` is a `matrix` or a `data.frame`

2. it is a SpatRaster if new.env is a SpatRaster (or several SpatRaster objects, if new.env is too large)
3. raw projections, as well as binary and filtered projections (if asked), are saved in the proj.name folder

Author(s)

Wilfried Thuiller, Damien Georges

See Also

BIOMOD_Modeling, BIOMOD_EnsembleModeling, BIOMOD_RangeSize

Other Main functions: BIOMOD_EnsembleForecasting(), BIOMOD_EnsembleModeling(), BIOMOD_FormatingData(), BIOMOD_LoadModels(), BIOMOD_Modeling(), BIOMOD_RangeSize()

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



# -------------------------------------------------------------#
file.out <- paste0(myRespName, "/", myRespName, ".AllModels.models.out")
if (file.exists(file.out)) {
  myBiomodModelOut <- get(load(file.out))
} else {

  # Format Data with true absences
  myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                       resp.var = myResp,
                                       resp.xy = myRespXY,
                                       expl.var = myExpl)

  # Model single models
```

```
  myBiomodModelOut <- BIOMOD_Modeling(bm.format = myBiomodData,
                                      modeling.id = 'AllModels',
                                      models = c('RF', 'GLM'),
                                      CV.strategy = 'random',
                                      CV.nb.rep = 2,
                                      CV.perc = 0.8,
                                      OPT.strategy = 'bigboss',
                                      metric.eval = c('TSS','AUCroc'),
                                      var.import = 3,
                                      seed.val = 42)
}


# ----------------------------------------------------------------#
# Project single models
file.proj <- paste0(myRespName, "/proj_Current/", myRespName, ".Current.projection.out")
if (file.exists(file.proj)) {
  myBiomodProj <- get(load(file.proj))
} else {
myBiomodProj <- BIOMOD_Projection(bm.mod = myBiomodModelOut,
                                  proj.name = 'Current',
                                  new.env = myExpl,
                                  models.chosen = 'all')
}
myBiomodProj
plot(myBiomodProj)
```

`BIOMOD_RangeSize` *Analyze the range size differences between projections of species distribution models*

Description

This function allows to calculate the absolute number of locations (pixels) lost, stable and gained, as well as the corresponding relative proportions, between two (or more) binary projections of (ensemble) species distribution models (*which can represent new time scales or environmental scenarios for example*).

Usage

```
BIOMOD_RangeSize(
  proj.current,
  proj.future,
  models.chosen = "all",
  metric.binary = NULL
)
```

Arguments

proj.current	a BIOMOD.projection.out object containing the initial projection(s) of the (ensemble) species distribution model(s)
proj.future	a BIOMOD.projection.out object containing the final binary projection(s) of the (ensemble) species distribution model(s)
models.chosen	a vector containing model names to be kept, must be either all or a subselection of model names that can be obtained with the get_projected_models function
metric.binary	(*optional, default* NULL) A vector containing evaluation metric selected to transform predictions into binary values, must be among POD, FAR, POFD, SR, ACCURACY, BIAS, AUCroc, AUCprg, TSS, KAPPA, OR, ORSS, CSI, ETS, BOYCE, MPA

Value

A BIOMOD.rangesize.out containing principaly two objects :

Compt.By.Species a data.frame containing the summary of range change for each comparison

- Loss : number of pixels predicted to be lost
- Stable_Abs : number of pixels not currently occupied and not predicted to be
- Stable_Pres : number of pixels currently occupied and predicted to remain occupied
- Gain : number of pixels predicted to be gained
- PercLoss : percentage of pixels currently occupied and predicted to be lost (Loss / (Loss + Stable_Pres))
- PercGain : percentage of pixels predicted to be gained compare to the number of pixels currently occupied (Gain / (Loss + Stable_Pres))
- SpeciesRangeChange : percentage of pixels predicted to change (loss or gain) compare to the number of pixels currently occupied (PercGain - PercLoss)
- CurrentRangeSize : number of pixels currently occupied
- FutureRangeSize0Disp : number of pixels predicted to be occupied, assuming no migration
- FutureRangeSize1Disp : number of pixels predicted to be occupied, assuming migration

loss.gain an object in the same form than the input data (proj.current and proj.future) and containing a value for each point/pixel of each comparison among :

- -2 : predicted to be lost
- -1 : predicted to remain occupied
- 0 : predicted to remain unoccupied
- 1 : predicted to be gained

Diff.By.Pixel an object in the same form than the input data (proj.current and proj.future) and containing a value for each point/pixel of each comparison obtain with :

- Future - 2* Current for binary data
- Future - Current for nonbinary after rescaling Future and Current from 0 to 1.

Author(s)

Maya Guéguen, Hélène Blancheteau

See Also

BIOMOD_Projection, BIOMOD_EnsembleForecasting, bm_PlotRangeSize

Other Main functions: BIOMOD_EnsembleForecasting(), BIOMOD_EnsembleModeling(), BIOMOD_FormatingData(), BIOMOD_LoadModels(), BIOMOD_Modeling(), BIOMOD_Projection()

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



# ---------------------------------------------------------------- #
file.out <- paste0(myRespName, "/", myRespName, ".AllModels.models.out")
if (file.exists(file.out)) {
  myBiomodModelOut <- get(load(file.out))
} else {

  # Format Data with true absences
  myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                       resp.var = myResp,
                                       resp.xy = myRespXY,
                                       expl.var = myExpl)

  # Model single models
  myBiomodModelOut <- BIOMOD_Modeling(bm.format = myBiomodData,
                                      modeling.id = 'AllModels',
                                      models = c('RF', 'GLM'),
                                      CV.strategy = 'random',
                                      CV.nb.rep = 2,
                                      CV.perc = 0.8,
```

```
                                    OPT.strategy = 'bigboss',
                                    metric.eval = c('TSS', 'AUCroc'),
                                    var.import = 3,
                                    seed.val = 42)
}

models.proj <- get_built_models(myBiomodModelOut, algo = "RF")
  # Project single models
  myBiomodProj <- BIOMOD_Projection(bm.mod = myBiomodModelOut,
                                    proj.name = 'CurrentRangeSize',
                                    new.env = myExpl,
                                    models.chosen = models.proj,
                                    metric.binary = 'all',
                                    build.clamping.mask = TRUE)


# ---------------------------------------------------------------- #
# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_future)
myExplFuture <- terra::rast(bioclim_future)

# Project onto future conditions
myBiomodProjectionFuture <- BIOMOD_Projection(bm.mod = myBiomodModelOut,
                                              proj.name = 'FutureRangeSize',
                                              new.env = myExplFuture,
                                              models.chosen = models.proj,
                                              metric.binary = 'TSS')

# Compute differences
myBiomodRangeSize <- BIOMOD_RangeSize(proj.current = myBiomodProj,
                                      proj.future = myBiomodProjectionFuture,
                                      metric.binary = "TSS")


# Represent main results
bm_PlotRangeSize(bm.range = myBiomodRangeSize)
```

`BIOMOD_Report` *Produce summary outputs from a simulation folder*

Description

This function allows to produce summary report or ODMAP table from a **biomod2** simulation folder.

Usage

```
BIOMOD_Report(
  bm.out,
  strategy = "report",
 params.color = list(color1 = "#eb4034", color2 = "#e0918b", color3 = "#658f70"),
 params.ODMAP = list(O.mod.objective = NULL, O.boundary = NULL, O.obs.type = NULL,
    O.pred.type = NULL, D.eco.level = NULL, D.samp.design = NULL)
)
```

Arguments

bm.out	a BIOMOD.formated.data or BIOMOD.formated.data.PA object returned by the BIOMOD_FormatingData function ; or a BIOMOD.models.out or BIOMOD.ensemble.models.out object that can be obtained with the BIOMOD_Modeling or BIOMOD_EnsembleModeling functions
strategy	a character defining the type of summary file that will be produced, must be report, ODMAP or code (see Details)
params.color	a list containing 3 color values to custom the reports
params.ODMAP	a list containing values of some ODMAP fields to be filled in from pre-existing choices (see Details)

Details

This function gathers and formats all objects contained in one **biomod2** modeling folder to produce, based on Rmarkdown templates, standardized reports to help the user :

- summarize its modeling results
- share them through standardized informations through ODMAP protocol
- provide reproducible code

Type of report Different data types are available, and require different values :

report **biomod2** provides functions to summarize the information, such as print, plot or summary methods adapted to BIOMOD.[...].out objects, as well as get_[...] and bm_Plot[...] functions. All these are called here and applied to objects contained in the provided modeling folder.

ODMAP following Zurell et al. 2020, ODMAP (Overview, Data, Model, Assessment and Prediction) protocol aims to standardize documentation of modeling to help improve both transparency and reproducibility of results. ODMAP v1.0 website provides an application to fill this type of report. **biomod2** tries here to help one user to pre-fill the fields of this protocol.

code call slot contained within BIOMOD.formated.data, BIOMOD.models.out, BIOMOD.projection.out and BIOMOD.ensemble.models.out objects keep in memory the R command used to obtain them. All these calls are gathered here in one summary file.

Value

A standardized `.html` file obtained from an `Rmarkdown` template, and a `.csv` table in the case of ODMAP report.

Author(s)

Maya Guéguen

References

- Zurell D, Franklin J, König C, Bouchet PJ, Serra-Diaz JM, Dormann CF, Elith J, Fandos Guzman G, Feng X, Guillera-Arroita G, Guisan A, Leitão PJ, Lahoz-Monfort JJ, Park DS, Peterson AT, Rapacciuolo G, Schmatz DR, Schröder B, Thuiller W, Yates KL, Zimmermann NE, Merow C (**2020**). *A standard protocol for describing species distribution models.* Ecography 43: 1261-1277. doi:10.1111/ecog.04960

See Also

ODMAP, match.call

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)




# ------------------------------------------------------------------- #
file.out <- paste0(myRespName, "/", myRespName, ".AllModels.models.out")
if (file.exists(file.out)) {
  myBiomodModelOut <- get(load(file.out))
} else {

  # Format Data with true absences
```

```
  myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                       resp.var = myResp,
                                       resp.xy = myRespXY,
                                       expl.var = myExpl)

  # Model single models
  myBiomodModelOut <- BIOMOD_Modeling(bm.format = myBiomodData,
                                      modeling.id = 'AllModels',
                                      models = c('RF', 'GLM'),
                                      CV.strategy = 'random',
                                      CV.nb.rep = 2,
                                      CV.perc = 0.8,
                                      OPT.strategy = 'bigboss',
                                      metric.eval = c('TSS', 'AUCroc'),
                                      var.import = 3,
                                      seed.val = 42)
}


file.proj <- paste0(myRespName, "/proj_Current/", myRespName, ".Current.projection.out")
if (file.exists(file.proj)) {
  myBiomodProj <- get(load(file.proj))
} else {

  # Project single models
  myBiomodProj <- BIOMOD_Projection(bm.mod = myBiomodModelOut,
                                    proj.name = 'Current',
                                    new.env = myExpl,
                                    models.chosen = 'all',
                                    build.clamping.mask = TRUE)
}


file.EM <- paste0(myRespName, "/", myRespName, ".AllModels.ensemble.models.out")
if (file.exists(file.EM)) {
  myBiomodEM <- get(load(file.EM))
} else {

  # Model ensemble models
  myBiomodEM <- BIOMOD_EnsembleModeling(bm.mod = myBiomodModelOut,
                                        models.chosen = 'all',
                                        em.by = 'all',
                                        em.algo = c('EMmean', 'EMca'),
                                        metric.select = c('TSS'),
                                        metric.select.thresh = c(0.7),
                                        metric.eval = c('TSS', 'AUCroc'),
                                        var.import = 3,
                                        seed.val = 42)
}


# ------------------------------------------------------------- #
# Compile summary reports
```

```
# BIOMOD_Report(bm.out = myBiomodModelOut, strategy = 'report')
# BIOMOD_Report(bm.out = myBiomodProj, strategy = 'report')
# BIOMOD_Report(bm.out = myBiomodEM, strategy = 'report')

# BIOMOD_Report(bm.out = myBiomodModelOut, strategy = 'ODMAP')
# BIOMOD_Report(bm.out = myBiomodModelOut, strategy = 'code')
```

`bm_BinaryTransformation` *Convert probability values into binary values using a predefined threshold*

Description

This internal **biomod2** function allows to convert probability (not necessary between `0` and `1`) values into binary presence-absence (`0` or `1`) values according to a predefined threshold (see Details).

Usage

```
bm_BinaryTransformation(data, threshold, do.filtering = FALSE)

## S4 method for signature 'data.frame'
bm_BinaryTransformation(data, threshold, do.filtering = FALSE)

## S4 method for signature 'matrix'
bm_BinaryTransformation(data, threshold, do.filtering = FALSE)

## S4 method for signature 'numeric'
bm_BinaryTransformation(data, threshold, do.filtering = FALSE)

## S4 method for signature 'SpatRaster'
bm_BinaryTransformation(data, threshold, do.filtering = FALSE)
```

Arguments

`data`	a `vector`, a `matrix`, `data.frame`, or a `SpatRaster` containing the data to be converted
`threshold`	a `numeric` or a `vector` of `numeric` corresponding to the threshold used to convert the given data (see Details)
`do.filtering`	(*optional, default* `FALSE`) A `logical` value defining whether filtered data should be returned, or binary one (see Details)

Details

If data is a vector, threshold should be a single numeric value.
If data is a matrix, data.frame or SpatRaster, threshold should be a vector containing as many values as the number of columns or layers contained in data. If only one numeric value is given, the same threshold will be applied to all columns or layers.

If do.filtering = FALSE, binary (0 or 1) values are returned.
If do.filtering = TRUE, values will be *filtered* according to threshold, meaning that :

- data < threshold will return 0
- data >= threshold will return the actual values of data (not transformed in 1)

Value

An object of the same class than data and containing either binary (0 or 1) values, or filtered values.

Author(s)

Wilfried Thuiller, Damien Georges

See Also

BIOMOD_Projection, BIOMOD_EnsembleForecasting

Other Secondary functions: bm_CrossValidation(), bm_FindOptimStat(), bm_MakeFormula(), bm_ModelAnalysis(), bm_ModelingOptions(), bm_PlotEvalBoxplot(), bm_PlotEvalMean(), bm_PlotRangeSize(), bm_PlotResponseCurves(), bm_PlotVarImpBoxplot(), bm_PseudoAbsences(), bm_RangeSize(), bm_RunModelsLoop(), bm_SRE(), bm_SampleBinaryVector(), bm_SampleFactorLevels(), bm_Tuning(), bm_VariablesImportance()

Examples

```
## Generate a 0-1000 vector (normal distribution)
vec.d <- rnorm(100, 500, 100)

## From continuous to binary / filtered vector
vec.d_bin <- bm_BinaryTransformation(data = vec.d, threshold = 500)
vec.d_filt <- bm_BinaryTransformation(data = vec.d, threshold = 500, do.filtering = TRUE)
cbind(vec.d, vec.d_bin, vec.d_filt)
```

bm_CrossValidation *Build cross-validation table*

Description

This internal **biomod2** function allows to build a cross-validation table according to 6 different methods : `random`, `kfold`, `block`, `strat`, `env` or `user.defined` (see Details).

Usage

```
bm_CrossValidation(
  bm.format,
  strategy,
  nb.rep = NULL,
  perc = NULL,
  k = NULL,
  balance = NULL,
  strat = NULL,
  env.var = NULL,
  user.table = NULL,
  do.full.models = FALSE,
  seed.val = NULL
)

bm_CrossValidation_user.defined(bm.format, ...)

## S4 method for signature 'BIOMOD.formated.data'
bm_CrossValidation_user.defined(bm.format, user.table)

## S4 method for signature 'BIOMOD.formated.data.PA'
bm_CrossValidation_user.defined(bm.format, user.table)

bm_CrossValidation_random(bm.format, ...)

## S4 method for signature 'BIOMOD.formated.data'
bm_CrossValidation_random(bm.format, nb.rep, perc)

## S4 method for signature 'BIOMOD.formated.data.PA'
bm_CrossValidation_random(bm.format, nb.rep, perc)

bm_CrossValidation_kfold(bm.format, ...)

## S4 method for signature 'BIOMOD.formated.data'
bm_CrossValidation_kfold(bm.format, nb.rep, k)

## S4 method for signature 'BIOMOD.formated.data.PA'
bm_CrossValidation_kfold(bm.format, nb.rep, k)
```

```
bm_CrossValidation_block(bm.format, ...)

## S4 method for signature 'BIOMOD.formated.data'
bm_CrossValidation_block(bm.format)

## S4 method for signature 'BIOMOD.formated.data.PA'
bm_CrossValidation_block(bm.format)

bm_CrossValidation_strat(bm.format, ...)

## S4 method for signature 'BIOMOD.formated.data'
bm_CrossValidation_strat(bm.format, balance, strat, k)

## S4 method for signature 'BIOMOD.formated.data.PA'
bm_CrossValidation_strat(bm.format, balance, strat, k)

bm_CrossValidation_env(bm.format, ...)

## S4 method for signature 'BIOMOD.formated.data'
bm_CrossValidation_env(bm.format, balance, k, env.var)

## S4 method for signature 'BIOMOD.formated.data.PA'
bm_CrossValidation_env(bm.format, balance, k, env.var)
```

Arguments

bm.format	a BIOMOD.formated.data or BIOMOD.formated.data.PA object returned by the BIOMOD_FormatingData function
strategy	a character corresponding to the cross-validation selection strategy, must be among random, kfold, block, strat, env or user.defined
nb.rep	(*optional, default* NULL) If strategy = 'random' or strategy = 'kfold', an integer corresponding to the number of sets (repetitions) of cross-validation points that will be drawn
perc	(*optional, default* NULL) If strategy = 'random', a numeric between 0 and 1 defining the percentage of data that will be kept for calibration
k	(*optional, default* NULL) If strategy = 'kfold' or strategy = 'strat' or strategy = 'env', an integer corresponding to the number of partitions
balance	(*optional, default* NULL) If strategy = 'strat' or strategy = 'env', a character corresponding to how data will be balanced between partitions, must be either presences or absence
strat	(*optional, default* NULL) If strategy = 'env', a character corresponding to how data will partitioned along gradient, must be among x, y, both

`env.var` (*optional*)
If `strategy = 'env'`, a `character` corresponding to the environmental variables used to build the partition. `k` partitions will be built for each environmental variables. *By default the function uses all environmental variables available.*

`user.table` (*optional, default* `NULL`)
If `strategy = 'user.defined'`, a `matrix` or `data.frame` defining for each repetition (in columns) which observation lines should be used for models calibration (`TRUE`) and validation (`FALSE`)

`do.full.models` (*optional, default* `FALSE`)
A `logical` value defining whether models should be also calibrated and validated over the whole dataset (and pseudo-absence datasets) or not

`seed.val` (*optional, default* `NULL`)
An `integer` value corresponding to the new seed value to be set

`...` (*optional, one or several of the listed above arguments depending on the selected method*)

Details

Several parameters are available within the function and some of them can be used with different cross-validation strategies :

.....	random	kfold	block	strat	env
nb.rep.	x.....	x....	.....	.....	...
perc...	x.....	.....	.....	.....	...
k.....	.....	x....	.....	x....	x..
balance	.....	.....	.....	x....	x..
strat..	.....	.....	.....	x....	...

Concerning column names of `matrix` output :

The number of columns depends on the strategy selected. The column names are given *a posteriori* of the selection, ranging from 1 to the number of columns. If `do.full.models = TRUE`, columns merging runs (and/or pseudo-absence datasets) are added at the end.

Concerning cross-validation strategies :

random Most simple method to calibrate and validate a model is to split the original dataset in two datasets : one to calibrate the model and the other one to validate it. The splitting can be repeated `nb.rep` times.

k-fold The k-fold method splits the original dataset in `k` datasets of equal sizes : each part is used successively as the validation dataset while the other `k-1` parts are used for the calibration, leading to `k` calibration/validation ensembles. This multiple splitting can be repeated `nb.rep` times.

block It may be used to test for model overfitting and to assess transferability in geographic space. `block` stratification was described in *Muscarella et al. 2014* (see References). Four bins of equal size are partitioned (bottom-left, bottom-right, top-left and top-right).

stratified It may be used to test for model overfitting and to assess transferability in geographic space. `x` and `y` stratification was described in *Wenger and Olden 2012* (see References). `y` stratification uses `k` partitions along the y-gradient, `x` stratification does the same for the x-gradient. `both` returns `2k` partitions: `k` partitions stratified along the x-gradient and `k` partitions stratified along the y-gradient.

environmental It may be used to test for model overfitting and to assess transferability in environmental space. It returns `k` partitions for each variable given in `env.var`.

user-defined Allow the user to give its own crossvalidation table. For a presence-absence dataset, column names must be formatted as: `_allData_RUNx` with `x` an integer. For a presence-only dataset for which several pseudo-absence dataset were generated, column names must be formatted as: `_PAx_RUNy` with `x` an integer and `PAx` an existing pseudo-absence dataset and `y` an integer

Concerning balance parameter :

If `balance = 'presences'`, presences are divided (balanced) equally over the partitions (e.g. *Fig. 1b in Muscarelly et al. 2014*). Absences or pseudo-absences will however be unbalanced over the partitions especially if the presences are clumped on an edge of the study area.

If `balance = 'absences'`, absences (resp. pseudo-absences or background) are divided (balanced) as equally as possible between the partitions (geographical balanced bins given that absences are spread over the study area equally, approach similar to *Fig. 1 in Wenger et Olden 2012*). Presences will however be unbalanced over the partitions especially if the presences are clumped on an edge of the study area.

Value

A `matrix` or `data.frame` defining for each repetition (in columns) which observation lines should be used for models calibration (`TRUE`) and validation (`FALSE`).

Author(s)

Maya Guéguen

References

- Muscarella R, Galante PJ, Soley-Guardia M, Boria RA, Kass JM, Uriarte M, Anderson RP (**2014**). *ENMeval: An R package for conducting spatially independent evaluations and estimating optimal model complexity for Maxent ecological niche models.* Methods in Ecology and Evolution, 5, 1198-1205. doi:10.1111/2041210X.12261
- Wenger SJ and Olden JD (**2012**). *Assessing transferability of ecological models: an underappreciated aspect of statistical validation.* Methods in Ecology and Evolution, 3, 260-267. doi:10.1111/j.2041210X.2011.00170.x

See Also

get.block, kfold, BIOMOD_FormatingData, BIOMOD_Modeling

Other Secondary functions: bm_BinaryTransformation(), bm_FindOptimStat(), bm_MakeFormula(), bm_ModelAnalysis(), bm_ModelingOptions(), bm_PlotEvalBoxplot(), bm_PlotEvalMean(), bm_PlotRangeSize(), bm_PlotResponseCurves(), bm_PlotVarImpBoxplot(), bm_PseudoAbsences(), bm_RangeSize(), bm_RunModelsLoop(), bm_SRE(), bm_SampleBinaryVector(), bm_SampleFactorLevels(), bm_Tuning(), bm_VariablesImportance()

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



# ---------------------------------------------------------------- #
# Format Data with true absences
myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                     resp.var = myResp,
                                     resp.xy = myRespXY,
                                     expl.var = myExpl)

# ---------------------------------------------------------------- #
# Create the different validation datasets

# random selection
cv.r <- bm_CrossValidation(bm.format = myBiomodData,
                           strategy = "random",
                           nb.rep = 3,
                           perc = 0.8)

# k-fold selection
cv.k <- bm_CrossValidation(bm.format = myBiomodData,
                           strategy = "kfold",
                           nb.rep = 2,
```

```
                            k = 3)

# block selection
cv.b <- bm_CrossValidation(bm.format = myBiomodData,
                            strategy = "block")

# stratified selection (geographic)
cv.s <- bm_CrossValidation(bm.format = myBiomodData,
                            strategy = "strat",
                            k = 2,
                            balance = "presences",
                            strat = "x")

# stratified selection (environmental)
cv.e <- bm_CrossValidation(bm.format = myBiomodData,
                            strategy = "env",
                            k = 2,
                            balance = "presences")

head(cv.r)
apply(cv.r, 2, table)
head(cv.k)
apply(cv.k, 2, table)
head(cv.b)
apply(cv.b, 2, table)
head(cv.s)
apply(cv.s, 2, table)
head(cv.e)
apply(cv.e, 2, table)
```

`bm_FindOptimStat` *Calculate the best score according to a given evaluation method*

Description

This internal **biomod2** function allows the user to find the threshold to convert continuous values into binary ones leading to the best score for a given evaluation metric.

Usage

```
bm_FindOptimStat(
  metric.eval = "TSS",
  obs,
  fit,
  nb.thresh = 100,
  threshold = NULL,
  boyce.bg.env = NULL,
```

```
  mpa.perc = 0.9,
  k = NULL
)

get_optim_value(metric.eval)

bm_CalculateStatBin(misc, metric.eval = "TSS")

bm_CalculateStatAbun(metric.eval, obs, fit, k)
```

Arguments

metric.eval	a character corresponding to the evaluation metric to be used, must be either AUCroc, AUCprg, TSS, KAPPA, ACCURACY, BIAS, POD, FAR, POFD, SR, CSI, ETS, OR, ORSS, BOYCE, MPA (*binary data*), RMSE, MAE, MSE, Rsquared, Rsquared_aj, Max_error (*abundance / count / relative data*), Accuracy, Recall, Precision, F1 (*multiclass/ordinal data*)
obs	a vector of observed values (binary, 0 or 1)
fit	a vector of fitted values (continuous)
nb.thresh	an integer corresponding to the number of thresholds to be tested over the range of fitted values
threshold	(*optional, default* NULL) A numeric corresponding to the threshold used to convert the given data
boyce.bg.env	(*optional, default* NULL) A matrix, data.frame, SpatVector or SpatRaster object containing values of environmental variables (in columns or layers) extracted from the background (*if presences are to be compared to background instead of absences or pseudo-absences selected for modeling*) *Note that old format from* **raster** *and* **sp** *are still supported such as* RasterStack *and* SpatialPointsDataFrame *objects.*
mpa.perc	a numeric between 0 and 1 corresponding to the percentage of correctly classified presences for Minimal Predicted Area (see ecospat.mpa() in **ecospat**)
k	an integer corresponding to the number of environmental variables used in the model
misc	a matrix corresponding to a contingency table

Details

Please refer to JWGFVR website ("Forecast verification - methods, issues and FAQ") to get detailed description (simple/complex metrics).
Optimal value of each method can be obtained with the get_optim_value function.

simple

- POD : Probability of detection (hit rate)
- FAR : False alarm ratio
- POFD : Probability of false detection (fall-out)
- SR : Success ratio

- ACCURACY : Accuracy (fraction correct)
- BIAS : Bias score (frequency bias)

complex
- AUCroc : Area Under Curve of Relative operating characteristic
- AUCprg : Area Under Curve of Precision-Recall-Gain curve
- TSS : True skill statistic (Hanssen and Kuipers discriminant, Peirce's skill score)
- KAPPA : Cohen's Kappa (Heidke skill score)
- OR : Odds Ratio
- ORSS : Odds ratio skill score (Yule's Q)
- CSI : Critical success index (threat score)
- ETS : Equitable threat score (Gilbert skill score)

presence-only
- BOYCE : Boyce index
- MPA : Minimal predicted area (cutoff optimizing MPA to predict 90% of presences)

abundance / count / relative data
- RMSE : Root Mean Square Error
- MSE : Mean Square Error
- MAE : Mean Absolute Error
- Rsquared : R squared
- Rsquared_aj : R squared adjusted
- Max_error : Maximum error

multiclass/ordinal data
- Accuracy : Accuracy
- Recall : Macro average Recall
- Precision : Macro average Precision
- F1 : Macro F1 score

Note that if a value is given to threshold, no optimization will be done, and only the score for this threshold will be returned.

The Boyce index returns NA values for SRE models because it can not be calculated with binary predictions.
This is also the reason why some NA values might appear for GLM models if they did not converge.

Value

A 1 row x 5 columns data.frame containing :

- metric.eval : the chosen evaluation metric
- cutoff : the associated cut-off used to transform the continuous values into binary
- sensitivity : the sensibility obtained on fitted values with this threshold
- specificity : the specificity obtained on fitted values with this threshold
- best.stat : the best score obtained for the chosen evaluation metric

Note

In order to break dependency loop between packages **biomod2** and **ecospat**, code of ecospat.boyce() and ecospat.mpa() in **ecospat**) functions have been copied within this file from version 3.2.2 (august 2022).

Author(s)

Damien Georges

References

- Engler R, Guisan A, Rechsteiner L (**2004**). *An improved approach for predicting the distribution of rare and endangered species from occurrence and pseudo-absence data.* Journal of Applied Ecology, 41(2), 263-274. doi:10.1111/j.00218901.2004.00881.x
- Hirzel AH, Le Lay G, Helfer V, Randin C, Guisan A (**2006**). *Evaluating the ability of habitat suitability models to predict species presences.* Ecological Modelling, 199(2), 142-152. doi:10.1016/j.ecolmodel.2006.05.017

See Also

`ecospat.boyce()` and `ecospat.mpa()` in **ecospat**, `BIOMOD_Modeling`, `bm_RunModelsLoop`, `BIOMOD_EnsembleModeling`

Other Secondary functions: `bm_BinaryTransformation()`, `bm_CrossValidation()`, `bm_MakeFormula()`, `bm_ModelAnalysis()`, `bm_ModelingOptions()`, `bm_PlotEvalBoxplot()`, `bm_PlotEvalMean()`, `bm_PlotRangeSize()`, `bm_PlotResponseCurves()`, `bm_PlotVarImpBoxplot()`, `bm_PseudoAbsences()`, `bm_RangeSize()`, `bm_RunModelsLoop()`, `bm_SRE()`, `bm_SampleBinaryVector()`, `bm_SampleFactorLevels()`, `bm_Tuning()`, `bm_VariablesImportance()`

Examples

```
## Generate a binary vector
vec.a <- sample(c(0, 1), 100, replace = TRUE)

## Generate a 0-1000 vector (random drawing)
vec.b <- runif(100, min = 0, max = 1000)

## Generate a 0-1000 vector (biased drawing)
BiasedDrawing <- function(x, m1 = 300, sd1 = 200, m2 = 700, sd2 = 200) {
  return(ifelse(x < 0.5, rnorm(1, m1, sd1), rnorm(1, m2, sd2)))
}
vec.c <- sapply(vec.a, BiasedDrawing)
vec.c[which(vec.c < 0)] <- 0
vec.c[which(vec.c > 1000)] <- 1000

## Find optimal threshold for a specific evaluation metric
bm_FindOptimStat(metric.eval = 'TSS', fit = vec.b, obs = vec.a)
bm_FindOptimStat(metric.eval = 'TSS', fit = vec.c, obs = vec.a, nb.thresh = 100)
bm_FindOptimStat(metric.eval = 'TSS', fit = vec.c, obs = vec.a, threshold = 280)
```

bm_MakeFormula *Standardized formula maker*

Description

This internal **biomod2** function allows the user to create easily a standardized formula that can be used later by statistical models.

Usage

```
bm_MakeFormula(
  resp.name,
  expl.var,
  type = "simple",
  interaction.level = 0,
  k = NULL
)
```

Arguments

resp.name	a character corresponding to the response variable name
expl.var	a matrix or data.frame containing the explanatory variables that will be used at the modeling step
type	a character corresponding to the wanted type of formula, must be simple, quadratic, polynomial or s_smoother
interaction.level	an integer corresponding to the interaction level depth between explanatory variables
k	(*optional, default* NULL) If type = 's_smoother', an integer corresponding to the smoothing parameter value of s or s arguments

Details

It is advised to give only a subset of expl.var table to avoid useless memory consuming.
If some explanatory variables are factorial, expl.var must be a data.frame whose corresponding columns are defined as factor.

Value

A formula class object that can be directly given to most of R statistical models.

Author(s)

Damien Georges

See Also

formula, s, s, bm_ModelingOptions, bm_Tuning, bm_RunModelsLoop

Other Secondary functions: bm_BinaryTransformation(), bm_CrossValidation(), bm_FindOptimStat(), bm_ModelAnalysis(), bm_ModelingOptions(), bm_PlotEvalBoxplot(), bm_PlotEvalMean(), bm_PlotRangeSize(), bm_PlotResponseCurves(), bm_PlotVarImpBoxplot(), bm_PseudoAbsences(), bm_RangeSize(), bm_RunModelsLoop(), bm_SRE(), bm_SampleBinaryVector(), bm_SampleFactorLevels(), bm_Tuning(), bm_VariablesImportance()

Examples

```
## Create simple simulated data
myResp.s <- sample(c(0, 1), 20, replace = TRUE)
myExpl.s <- data.frame(var1 = sample(c(0, 1), 100, replace = TRUE),
                       var2 = rnorm(100),
                       var3 = 1:100)

## Generate automatic formula
bm_MakeFormula(resp.name = 'myResp.s',
               expl.var = head(myExpl.s),
               type = 'quadratic',
               interaction.level = 0)
```

---

bm_ModelAnalysis *Analyze the residuals of the single models*

Description

This function return several graphs to help analyse the single models.

Usage

```
bm_ModelAnalysis(
  bm.mod,
  models.chosen = "all",
  color.by = "full.name",
  do.plot = TRUE
)
```

Arguments

bm.mod	a BIOMOD.models.out object returned by the BIOMOD_Modeling function
models.chosen	a vector containing model names to be kept, must be either all or a sub-selection of model names that can be obtained with the get_built_models function applied to bm.mod

`color.by` a `character` corresponding to the way plots will be colored, must be among `full.name`, `PA`, `run` or `algo`

`do.plot` (*optional, default* `TRUE`)
A `logical` value defining whether the plot is to be rendered or not

Details

5 plots can be obtained with this function :

residuals ~ observations number to detect outliers. The x-axis only helps to find the outlier number.

residuals Q-Q plot to check if residuals follow a normal distribution. Points should follow the black line.

residuals histogram to check if residuals follow a normal distribution. Histogram should represent a gaussian distribution.

residuals ~ fitted values to detect an heteroscedasticity of the residuals. It corresponds to the Tukey-Anscombe plot.

R scores to detect overfitting. Representing `Rsquared` and `Rsquared_aj` values, there should not be a big gap between calibration and validation values.

Please note that all plots are made for all models, independently to the different models assumptions. It is up to the user to interpret the graphics.

Value

A `list` containing :

1. a `data.frame` with variables, predicted values and model residuals
2. a `data.frame` with the R score values
3. 5 `ggplot` objects (from A to E) representing the different analyses.

Author(s)

Hélène Blancheteau

See Also

Other Secondary functions: `bm_BinaryTransformation()`, `bm_CrossValidation()`, `bm_FindOptimStat()`, `bm_MakeFormula()`, `bm_ModelingOptions()`, `bm_PlotEvalBoxplot()`, `bm_PlotEvalMean()`, `bm_PlotRangeSize()`, `bm_PlotResponseCurves()`, `bm_PlotVarImpBoxplot()`, `bm_PseudoAbsences()`, `bm_RangeSize()`, `bm_RunModelsLoop()`, `bm_SRE()`, `bm_SampleBinaryVector()`, `bm_SampleFactorLevels()`, `bm_Tuning()`, `bm_VariablesImportance()`

Other Plot functions: `bm_PlotEvalBoxplot()`, `bm_PlotEvalMean()`, `bm_PlotRangeSize()`, `bm_PlotResponseCurves()`, `bm_PlotVarImpBoxplot()`

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



# -------------------------------------------------------------#
file.out <- paste0(myRespName, "/", myRespName, ".AllModels.models.out")
if (file.exists(file.out)) {
  myBiomodModelOut <- get(load(file.out))
} else {

  # Format Data with true absences
  myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                       resp.var = myResp,
                                       resp.xy = myRespXY,
                                       expl.var = myExpl)

  # Model single models
  myBiomodModelOut <- BIOMOD_Modeling(bm.format = myBiomodData,
                                      modeling.id = 'AllModels',
                                      models = c('RF', 'GLM'),
                                      CV.strategy = 'random',
                                      CV.nb.rep = 2,
                                      CV.perc = 0.8,
                                      OPT.strategy = 'bigboss',
                                      metric.eval = c('TSS','AUCroc'),
                                      var.import = 3,
                                      seed.val = 42)
}


# -------------------------------------------------------------#
# Explore single models
myBiomodAnalysis <- bm_ModelAnalysis(bm.mod = myBiomodModelOut, color.by = "run")
```

```
plot(myBiomodAnalysis$plot.outliers)
```

`bm_ModelingOptions` *Configure the modeling options for each selected model*

Description

Parameterize and/or tune **biomod2**'s single models options.

Usage

```
bm_ModelingOptions(
  data.type = "binary",
  models,
  strategy,
  user.val = NULL,
  user.base = NULL,
  bm.format = NULL,
  calib.lines = NULL
)
```

Arguments

`data.type`	a `character` corresponding to the data type to be used, must be either `binary`, `count`, `relative`, `abundance`, `multiclass`, `ordinal`
`models`	a `vector` containing model names to be computed, must be among `ANN`, `CTA`, `DNN`, `FDA`, `GAM`, `GBM`, `GLM`, `MARS`, `MAXENT`, `MAXNET`, `RF`, `RFd`, `SRE`, `XGBOOST`
`strategy`	a `character` corresponding to the method to select models' parameters values, must be either `default`, `bigboss`, `user.defined`, `tuned` (see Details)
`user.val`	(*optional, default* `NULL`) A `list` containing parameters values for some (all) models
`user.base`	(*optional, default* `NULL`) If `strategy = 'user.defined'`, a `character` corresponding to the basic set of options to be modified by user defined values, must be either `default` or `bigboss` (see Details)
`bm.format`	(*optional, default* `NULL`) A `BIOMOD.formated.data` or `BIOMOD.formated.data.PA` object returned by the `BIOMOD_FormatingData` function
`calib.lines`	(*optional, default* `NULL`) A `data.frame` object returned by `get_calib_lines` or `bm_CrossValidation` functions

Details

This function creates a BIOMOD.models.options object containing parameter values for each single model that can be run within **biomod2** through BIOMOD_Modeling function.

11 different algorithms are currently available, with different versions for some (GAM, MAXENT, RF). These models are associated with *binary* or *nonbinary* datatypes, but all combinations are not available. They are all listed within the ModelsTable dataset.

Different strategies are available to set those parameters, through the strategy argument :

default all parameters names and values are directly retrieve from functions to be called through formalArgs and formals functions

bigboss default parameter values are updated with values predefined by **biomod2** team (see OptionsBigboss)

user.defined default parameter values are updated with values provided by the user

tuned default parameter values are updated by calling bm_Tuning function

To define the same options for all datasets of a model, options can be provided through the user.val parameter as a list whose name is for_all_datasets.

Value

A BIOMOD.models.options of object that can be used to build species distribution model(s) with the BIOMOD_Modeling function.

Note

MAXENT being the only external model (not called through a R package), default parameters, and their values, are the following :

FLAG	default	Description
path_to_maxent.jar	getwd()	a character corresponding to path to maxent.jar file
memory_allocated	512	an integer corresponding to the amount of memory (in Mo) reserved for java to
initial_heap_size	NULL	a character corresponding to initial heap space (shared memory space) allocated
max_heap_size	NULL	a character corresponding to maximum heap space (shared memory space) alloca
background_data_dir	'default'	a character corresponding to path to folder where explanatory variables are store
visible	FALSE	a logical value defining whether MAXENT user interface is to be used or not
linear	TRUE	a logical value defining whether linear features are to be used or not
quadratic	TRUE	a logical value defining whether quadratic features are to be used or not
product	TRUE	a logical value defining whether product features are to be used or not
threshold	TRUE	a logical value defining whether threshold features are to be used or not
hinge	TRUE	a logical value defining whether hinge features are to be used or not
l2lqthreshold	10	an integer corresponding to the number of samples at which quadratic features st
lq2lqptthreshold	80	an integer corresponding to the number of samples at which product and thresho
hingethreshold	15	an integer corresponding to the number of samples at which hinge features start b
beta_lqp	-1.0	a numeric corresponding to the regularization parameter to be applied to all linear
beta_threshold	-1.0	a numeric corresponding to the regularization parameter to be applied to all thresh
beta_hinge	-1.0	a numeric corresponding to the regularization parameter to be applied to all hinge
beta_categorical	-1.0	a numeric corresponding to the regularization parameter to be applied to all catego

betamultiplier	1	a numeric corresponding to the number by which multiply all automatic regulariza
defaultprevalence	0.5	a numeric corresponding to the default prevalence of the modelled species (*probal*
togglelayerselected	NULL	a character defining the prefix to be used to REMOVE environmental layers who
maximumbackground	10000	an integer corresponding to the maximum number of pixels to be selected randor
maximumiterations	500	an integer corresponding to the maximum number of iterations for training
convergencethreshold	0.00005	a numeric corresponding to the drop in log loss per iteration below which the train
autofeature	TRUE	a logical value defining whether feature classes to be used will automatically be
jackknife	FALSE	a logical value defining whether variables' importance is to be measured or not.
writeclampgrid	FALSE	a logical value defining whether clamping mask is to be saved or not (*absolute d*
writemess	FALSE	a logical value defining whether multidimensional environmental similarity surfa
logfile	'maxent.log'	a character corresponding to file name in which debugging information will be w
verbose	FALSE	a logical value defining whether detailed diagnostics for debugging should be giv

Author(s)

Damien Georges, Wilfried Thuiller, Maya Guéguen

See Also

ModelsTable, OptionsBigboss, BIOMOD.models.options, bm_Tuning, BIOMOD_Modeling

Other Secondary functions: bm_BinaryTransformation(), bm_CrossValidation(), bm_FindOptimStat(), bm_MakeFormula(), bm_ModelAnalysis(), bm_PlotEvalBoxplot(), bm_PlotEvalMean(), bm_PlotRangeSize(), bm_PlotResponseCurves(), bm_PlotVarImpBoxplot(), bm_PseudoAbsences(), bm_RangeSize(), bm_RunModelsLoop(), bm_SRE(), bm_SampleBinaryVector(), bm_SampleFactorLevels(), bm_Tuning(), bm_VariablesImportance()

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)
```

```
# ------------------------------------------------------------------#
# Format Data with true absences
myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                     resp.var = myResp,
                                     resp.xy = myRespXY,
                                     expl.var = myExpl)

# k-fold selection
cv.k <- bm_CrossValidation(bm.format = myBiomodData,
                           strategy = 'kfold',
                           nb.rep = 2,
                           k = 3)


# ------------------------------------------------------------------#
allModels <- c('ANN', 'CTA', 'DNN', 'FDA', 'GAM', 'GBM', 'GLM', 'MARS'
               , 'MAXENT', 'MAXNET', 'RF', 'RFd', 'SRE', 'XGBOOST')

# default parameters
opt.d <- bm_ModelingOptions(data.type = 'binary',
                            models = allModels,
                            strategy = 'default')

# providing formated data
opt.df <- bm_ModelingOptions(data.type = 'binary',
                             models = allModels,
                             strategy = 'default',
                             bm.format = myBiomodData,
                             calib.lines = cv.k)

opt.d
opt.d@models
opt.d@options$ANN.binary.nnet.nnet
names(opt.d@options$ANN.binary.nnet.nnet@args.values)

opt.df@options$ANN.binary.nnet.nnet
names(opt.df@options$ANN.binary.nnet.nnet@args.values)


# ------------------------------------------------------------------#
# bigboss parameters
opt.b <- bm_ModelingOptions(data.type = 'binary',
                            models = allModels,
                            strategy = 'bigboss')

# user defined parameters
user.SRE <- list('_allData_allRun' = list(quant = 0.01))
user.XGBOOST <- list('_allData_allRun' = list(nrounds = 10))
user.val <- list(SRE.binary.biomod2.bm_SRE = user.SRE
                 , XGBOOST.binary.xgboost.xgb_train = user.XGBOOST)

opt.u <- bm_ModelingOptions(data.type = 'binary',
                            models = c('SRE', 'XGBOOST'),
```

```
                              strategy = 'user.defined',
                              user.base = 'bigboss',
                              user.val = user.val)

opt.b
opt.u

## Not run:
# tuned parameters with formated data
opt.t <- bm_ModelingOptions(data.type = 'binary',
                              models = c('SRE', 'XGBOOST'),
                              strategy = 'tuned',
                              bm.format = myBiomodData)
opt.t

## End(Not run)
```

`bm_PlotEvalBoxplot` *Plot boxplot of evaluation scores*

Description

This function represents boxplot of evaluation scores of species distribution models, from `BIOMOD.models.out` or `BIOMOD.ensemble.models.out` objects that can be obtained from `BIOMOD_Modeling` or `BIOMOD_EnsembleModeling` functions. Scores are represented according to 2 grouping methods (see Details).

Usage

```
bm_PlotEvalBoxplot(
  bm.out,
  dataset = "calibration",
  group.by = c("algo", "run"),
  do.plot = TRUE,
  ...
)
```

Arguments

`bm.out`	a `BIOMOD.models.out` or `BIOMOD.ensemble.models.out` object that can be obtained with the `BIOMOD_Modeling` or `BIOMOD_EnsembleModeling` functions
`dataset`	a `character` corresponding to the dataset upon which evaluation metrics have been calculated and that is to be represented, must be among `calibration`, `validation`, `evaluation`

group.by	a 2-length vector containing the way kept models will be represented, must be among full.name, PA, run, algo (if bm.out is a BIOMOD.models.out object), or full.name, merged.by.PA, merged.by.run, merged.by.algo (if bm.out is a BIOMOD.ensemble.models.out object)
do.plot	(*optional, default* TRUE) A logical value defining whether the plot is to be rendered or not
...	some additional arguments (see Details)

Details

... can take the following values :

- main : a character corresponding to the graphic title
- scales : a character corresponding to the scales argument of the facet_wrap function, must be either fixed, free_x, free_y or free

Value

A list containing a data.frame with evaluation scores and the corresponding ggplot object representing them in boxplot.

Author(s)

Damien Georges, Maya Guéguen

See Also

BIOMOD.models.out, BIOMOD.ensemble.models.out, BIOMOD_Modeling, BIOMOD_EnsembleModeling, get_evaluations

Other Secondary functions: bm_BinaryTransformation(), bm_CrossValidation(), bm_FindOptimStat(), bm_MakeFormula(), bm_ModelAnalysis(), bm_ModelingOptions(), bm_PlotEvalMean(), bm_PlotRangeSize(), bm_PlotResponseCurves(), bm_PlotVarImpBoxplot(), bm_PseudoAbsences(), bm_RangeSize(), bm_RunModelsLoop(), bm_SRE(), bm_SampleBinaryVector(), bm_SampleFactorLevels(), bm_Tuning(), bm_VariablesImportance()

Other Plot functions: bm_ModelAnalysis(), bm_PlotEvalMean(), bm_PlotRangeSize(), bm_PlotResponseCurves(), bm_PlotVarImpBoxplot()

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
```

```
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



# -------------------------------------------------------------
file.out <- paste0(myRespName, "/", myRespName, ".AllModels.models.out")
if (file.exists(file.out)) {
  myBiomodModelOut <- get(load(file.out))
} else {

  # Format Data with true absences
  myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                       resp.var = myResp,
                                       resp.xy = myRespXY,
                                       expl.var = myExpl)

  # Model single models
  myBiomodModelOut <- BIOMOD_Modeling(bm.format = myBiomodData,
                                      modeling.id = 'AllModels',
                                      models = c('RF', 'GLM'),
                                      CV.strategy = 'random',
                                      CV.nb.rep = 2,
                                      CV.perc = 0.8,
                                      OPT.strategy = 'bigboss',
                                      metric.eval = c('TSS', 'ROC'),
                                      var.import = 3,
                                      seed.val = 42)
}


# -------------------------------------------------------------
# Get evaluation scores
get_evaluations(myBiomodModelOut)

# Represent evaluation scores
bm_PlotEvalBoxplot(bm.out = myBiomodModelOut, group.by = c('algo', 'run'))
```

---

bm_PlotEvalMean *Plot mean evaluation scores*

Description

This function represents mean evaluation scores (and their standard deviation) of species distribution models, from BIOMOD.models.out or BIOMOD.ensemble.models.out objects that can be obtained from BIOMOD_Modeling or BIOMOD_EnsembleModeling functions. Scores are represented according to 2 different evaluation methods, and models can be grouped (see Details).

Usage

```
bm_PlotEvalMean(
  bm.out,
  metric.eval = NULL,
  dataset = "calibration",
  group.by = "algo",
  do.plot = TRUE,
  ...
)
```

Arguments

bm.out	a BIOMOD.models.out or BIOMOD.ensemble.models.out object that can be obtained with the BIOMOD_Modeling or BIOMOD_EnsembleModeling functions
metric.eval	a 2-length vector containing evaluation metric names to be used, must be among the metrics used for bm.out and that can be obtained with the get_evaluations function applied to bm.out
dataset	a character corresponding to the dataset upon which evaluation metrics have been calculated and that is to be represented, must be among calibration, validation, evaluation
group.by	a character corresponding to the way kept models will be combined to compute mean and sd evaluation scores, must be among full.name, PA, run, algo (if bm.out is a BIOMOD.models.out object), or full.name, merged.by.PA, merged.by.run, merged.by.algo (if bm.out is a BIOMOD.ensemble.models.out object)
do.plot	(*optional, default* TRUE) A logical value defining whether the plot is to be rendered or not
...	some additional arguments (see Details)

Details

... can take the following values :

- xlim : an integer corresponding to the x maximum limit to represent
- ylim : an integer corresponding to the y maximum limit to represent
- main : a character corresponding to the graphic title
- col : a vector containing new color values

Value

A `list` containing a `data.frame` with mean and standard deviation of evaluation scores and the corresponding `ggplot` object representing them according to 2 different evaluation methods.

Author(s)

Damien Georges, Maya Guéguen

See Also

`BIOMOD.models.out`, `BIOMOD.ensemble.models.out`, `BIOMOD_Modeling`, `BIOMOD_EnsembleModeling`, `get_evaluations`

Other Secondary functions: `bm_BinaryTransformation()`, `bm_CrossValidation()`, `bm_FindOptimStat()`, `bm_MakeFormula()`, `bm_ModelAnalysis()`, `bm_ModelingOptions()`, `bm_PlotEvalBoxplot()`, `bm_PlotRangeSize()`, `bm_PlotResponseCurves()`, `bm_PlotVarImpBoxplot()`, `bm_PseudoAbsences()`, `bm_RangeSize()`, `bm_RunModelsLoop()`, `bm_SRE()`, `bm_SampleBinaryVector()`, `bm_SampleFactorLevels()`, `bm_Tuning()`, `bm_VariablesImportance()`

Other Plot functions: `bm_ModelAnalysis()`, `bm_PlotEvalBoxplot()`, `bm_PlotRangeSize()`, `bm_PlotResponseCurves()`, `bm_PlotVarImpBoxplot()`

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



# -------------------------------------------------------------------
file.out <- paste0(myRespName, "/", myRespName, ".AllModels.models.out")
if (file.exists(file.out)) {
  myBiomodModelOut <- get(load(file.out))
} else {

  # Format Data with true absences
```

```
  myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                       resp.var = myResp,
                                       resp.xy = myRespXY,
                                       expl.var = myExpl)

  # Model single models
  myBiomodModelOut <- BIOMOD_Modeling(bm.format = myBiomodData,
                                      modeling.id = 'AllModels',
                                      models = c('RF', 'GLM'),
                                      CV.strategy = 'random',
                                      CV.nb.rep = 2,
                                      CV.perc = 0.8,
                                      OPT.strategy = 'bigboss',
                                      metric.eval = c('TSS', 'ROC'),
                                      var.import = 3,
                                      seed.val = 42)
}


# ----------------------------------------------------------------
# Get evaluation scores
get_evaluations(myBiomodModelOut)

# Represent mean evaluation scores
bm_PlotEvalMean(bm.out = myBiomodModelOut)
```

`bm_PlotRangeSize` *Plot species range change*

Description

This function represents species range change from object that can be obtained from `BIOMOD_RangeSize` function. Several graphics can be obtained, representing global counts or proportions of gains / losses, as well as spatial representations (see Details).

Usage

```
bm_PlotRangeSize(
  bm.range,
  do.count = TRUE,
  do.perc = TRUE,
  do.maps = TRUE,
  do.mean = TRUE,
  do.plot = TRUE
)
```

Arguments

bm.range	an BIOMOD.rangesize.out object returned by the BIOMOD_RangeSize function
do.count	(*optional, default* TRUE) A logical value defining whether the count plot is to be computed or not
do.perc	(*optional, default* TRUE) A logical value defining whether the percentage plot is to be computed or not
do.maps	(*optional, default* TRUE) A logical value defining whether the maps plot is to be computed or not
do.mean	(*optional, default* TRUE) A logical value defining whether the mean maps plot is to be computed or not
do.plot	(*optional, default* TRUE) A logical value defining whether the plots are to be rendered or not

Details

4 plots can be obtained with this function :

Count barplot representing absolute number of locations (pixels) lost, stable and gained

Percentage barplot representing percentage of locations (pixels) lost, stable, and the corresponding Species Range Change (PercGain - PercLoss)

SRC models maps representing spatially locations (pixels) lost, stable and gained for each single distribution model

SRC community averaging maps representing spatially locations (pixels) lost, stable and gained, taking the majoritary value across single distribution models (and representing the percentage of models' agreement)

Please see bm_RangeSize *function for more details about the values.*

Value

A list containing one or several data.frame and the corresponding ggplot object representing species range change.

Author(s)

Maya Guéguen, Hélène Blancheteau

See Also

BIOMOD_RangeSize bm_RangeSize

Other Secondary functions: bm_BinaryTransformation(), bm_CrossValidation(), bm_FindOptimStat(), bm_MakeFormula(), bm_ModelAnalysis(), bm_ModelingOptions(), bm_PlotEvalBoxplot(), bm_PlotEvalMean(), bm_PlotResponseCurves(), bm_PlotVarImpBoxplot(), bm_PseudoAbsences(), bm_RangeSize(), bm_RunModelsLoop(), bm_SRE(), bm_SampleBinaryVector(), bm_SampleFactorLevels(), bm_Tuning(), bm_VariablesImportance()

Other Plot functions: bm_ModelAnalysis(), bm_PlotEvalBoxplot(), bm_PlotEvalMean(), bm_PlotResponseCurves(), bm_PlotVarImpBoxplot()

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



# -------------------------------------------------------------#
file.out <- paste0(myRespName, "/", myRespName, ".AllModels.models.out")
if (file.exists(file.out)) {
  myBiomodModelOut <- get(load(file.out))
} else {

  # Format Data with true absences
  myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                       resp.var = myResp,
                                       resp.xy = myRespXY,
                                       expl.var = myExpl)

  # Model single models
  myBiomodModelOut <- BIOMOD_Modeling(bm.format = myBiomodData,
                                      modeling.id = 'AllModels',
                                      models = c('RF', 'GLM'),
                                      CV.strategy = 'random',
                                      CV.nb.rep = 2,
                                      CV.perc = 0.8,
                                      OPT.strategy = 'bigboss',
                                      metric.eval = c('TSS', 'ROC'),
                                      var.import = 3,
                                      seed.val = 42)
}

models.proj <- get_built_models(myBiomodModelOut, algo = "RF")
  # Project single models
  myBiomodProj <- BIOMOD_Projection(bm.mod = myBiomodModelOut,
                                    proj.name = 'CurrentRangeSize',
                                    new.env = myExpl,
```

```
                                    models.chosen = models.proj,
                                    metric.binary = 'all')


# ------------------------------------------------------------------#
# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_future)
myExplFuture <- terra::rast(bioclim_future)


# Project onto future conditions
myBiomodProjectionFuture <- BIOMOD_Projection(bm.mod = myBiomodModelOut,
                                              proj.name = 'FutureRangeSize',
                                              new.env = myExplFuture,
                                              models.chosen = models.proj,
                                              metric.binary = 'TSS')

# Compute differences
myBiomodRangeSize <- BIOMOD_RangeSize(proj.current = myBiomodProj,
                                      proj.future = myBiomodProjectionFuture,
                                      metric.binary = "TSS")


# Represent main results
bm_PlotRangeSize(bm.range = myBiomodRangeSize)
```

`bm_PlotResponseCurves` *Plot response curves*

Description

This function represents response curves of species distribution models, from `BIOMOD.models.out` or `BIOMOD.ensemble.models.out` objects that can be obtained from `BIOMOD_Modeling` or `BIOMOD_EnsembleModeling` functions. Response curves can be represented in either 2 or 3 dimensions (meaning 1 or 2 explanatory variables at a time, see Details).

Usage

```
bm_PlotResponseCurves(
  bm.out,
  models.chosen = "all",
  new.env = get_formal_data(bm.out, "expl.var"),
  show.variables = get_formal_data(bm.out, "expl.var.names"),
  fixed.var = "mean",
  do.bivariate = FALSE,
```

```
  do.plot = TRUE,
  do.progress = TRUE,
  ...
)
```

Arguments

bm.out	a BIOMOD.models.out or BIOMOD.ensemble.models.out object that can be obtained with the BIOMOD_Modeling or BIOMOD_EnsembleModeling functions
models.chosen	a vector containing model names to be kept, must be either all or a sub-selection of model names that can be obtained with the get_built_models function
new.env	a matrix, data.frame or SpatRaster object containing the new explanatory variables (in columns or layers, with names matching the variables names given to the BIOMOD_FormatingData function to build bm.out) that will be used to project the species distribution model(s) *Note that old format from* **raster** *are still supported such as* RasterStack *objects.*
show.variables	a vector containing the names of the explanatory variables present into new.env parameter and to be plotted
fixed.var	a character corresponding to the statistic to be used to fix as constant the remaining variables other than the one used to predict response, must be either mean, median, min, max
do.bivariate	(*optional, default* FALSE) A logical value defining whether the response curves are to be represented in 3 dimensions (meaning 2 explanatory variables at a time) or not (meaning only 1)
do.plot	(*optional, default* TRUE) A logical value defining whether the plot is to be rendered or not
do.progress	(*optional, default* TRUE) A logical value defining whether the progress bar is to be rendered or not
...	some additional arguments (see Details)

Details

This function is an adaptation of the Evaluation Strip method proposed by Elith et al. (2005). To build the predicted response curves :

- n-1 variables are set constant to a fixed value determined by the fixed.var parameter (in the case of categorical variable, the most represented class is taken)
- the remaining variable is made to vary throughout its range given by the new.env parameter
- predicted values are computed with these n-1 fixed variables, and this studied variable varying

If do.bivariate = TRUE, 2 variables are varying at the same time.

The response curves obtained show the sensibility of the model to the studied variable.
Note that this method does not account for interactions between variables.

... can take the following values :

- `main` : a `character` corresponding to the graphic title

Value

A `list` containing a `data.frame` with variables and predicted values and the corresponding `ggplot` object representing response curves.

Author(s)

Damien Georges, Maya Guéguen

References

- Elith J, Ferrier S, Huettmann F, Leathwick JR (**2005**). *The evaluation strip: A new and robust method for plotting predicted responses from species distribution models.* Ecological Modelling, 186, 280-289. doi:10.1016/j.ecolmodel.2004.12.007

See Also

`BIOMOD.models.out`, `BIOMOD.ensemble.models.out`, `BIOMOD_Modeling`, `BIOMOD_EnsembleModeling`

Other Secondary functions: `bm_BinaryTransformation()`, `bm_CrossValidation()`, `bm_FindOptimStat()`, `bm_MakeFormula()`, `bm_ModelAnalysis()`, `bm_ModelingOptions()`, `bm_PlotEvalBoxplot()`, `bm_PlotEvalMean()`, `bm_PlotRangeSize()`, `bm_PlotVarImpBoxplot()`, `bm_PseudoAbsences()`, `bm_RangeSize()`, `bm_RunModelsLoop()`, `bm_SRE()`, `bm_SampleBinaryVector()`, `bm_SampleFactorLevels()`, `bm_Tuning()`, `bm_VariablesImportance()`

Other Plot functions: `bm_ModelAnalysis()`, `bm_PlotEvalBoxplot()`, `bm_PlotEvalMean()`, `bm_PlotRangeSize()`, `bm_PlotVarImpBoxplot()`

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]
```

```
# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



# -------------------------------------------------------------------#
file.out <- paste0(myRespName, "/", myRespName, ".AllModels.models.out")
if (file.exists(file.out)) {
  myBiomodModelOut <- get(load(file.out))
} else {

  # Format Data with true absences
  myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                       resp.var = myResp,
                                       resp.xy = myRespXY,
                                       expl.var = myExpl)

  # Model single models
  myBiomodModelOut <- BIOMOD_Modeling(bm.format = myBiomodData,
                                      modeling.id = 'AllModels',
                                      models = c('RF', 'GLM'),
                                      CV.strategy = 'random',
                                      CV.nb.rep = 2,
                                      CV.perc = 0.8,
                                      OPT.strategy = 'bigboss',
                                      metric.eval = c('TSS', 'ROC'),
                                      var.import = 3,
                                      seed.val = 42)
}


# -------------------------------------------------------------------#
# Represent response curves
mods <- get_built_models(myBiomodModelOut, run = 'RUN1')
bm_PlotResponseCurves(bm.out = myBiomodModelOut,
                      models.chosen = mods,
                      fixed.var = 'median')
## fixed.var can also be set to 'min', 'max' or 'mean'
# bm_PlotResponseCurves(bm.out = myBiomodModelOut,
#                       models.chosen = mods,
#                       fixed.var = 'min')

# Bivariate case (one model)
# variables can be selected with argument 'show.variables'
# models can be selected with argument 'models.chosen'
mods <- get_built_models(myBiomodModelOut, full.name = 'GuloGulo_allData_RUN2_RF')
bm_PlotResponseCurves(bm.out = myBiomodModelOut,
                      show.variables = c("bio4","bio12","bio11"),
                      models.chosen = mods,
                      fixed.var = 'median',
                      do.bivariate = TRUE)
```

bm_PlotVarImpBoxplot *Plot boxplot of variables importance*

Description

This function represents boxplot of variables importance of species distribution models, from BIOMOD.models.out or BIOMOD.ensemble.models.out objects that can be obtained from BIOMOD_Modeling or BIOMOD_EnsembleModeling functions. Scores are represented according to 3 grouping methods (see Details).

Usage

```
bm_PlotVarImpBoxplot(
  bm.out,
  group.by = c("run", "expl.var", "algo"),
  do.plot = TRUE,
  ...
)
```

Arguments

bm.out	a BIOMOD.models.out or BIOMOD.ensemble.models.out object that can be obtained with the BIOMOD_Modeling or BIOMOD_EnsembleModeling functions
group.by	a 3-length vector containing the way kept models will be represented, must be among full.name, PA, run, algo, expl.var (if bm.out is a BIOMOD.models.out object), or full.name, merged.by.PA, merged.by.run, merged.by.algo, expl.var (if bm.out is a BIOMOD.ensemble.models.out object)
do.plot	(*optional, default* TRUE) A logical value defining whether the plot is to be rendered or not
...	some additional arguments (see Details)

Details

... can take the following values :

- main : a character corresponding to the graphic title

Value

A list containing a data.frame with variables importance and the corresponding ggplot object representing them in boxplot.

Author(s)

Damien Georges, Maya Guéguen

See Also

BIOMOD.models.out, BIOMOD.ensemble.models.out, BIOMOD_Modeling, BIOMOD_EnsembleModeling, get_variables_importance

Other Secondary functions: bm_BinaryTransformation(), bm_CrossValidation(), bm_FindOptimStat(), bm_MakeFormula(), bm_ModelAnalysis(), bm_ModelingOptions(), bm_PlotEvalBoxplot(), bm_PlotEvalMean(), bm_PlotRangeSize(), bm_PlotResponseCurves(), bm_PseudoAbsences(), bm_RangeSize(), bm_RunModelsLoop(), bm_SRE(), bm_SampleBinaryVector(), bm_SampleFactorLevels(), bm_Tuning(), bm_VariablesImportance()

Other Plot functions: bm_ModelAnalysis(), bm_PlotEvalBoxplot(), bm_PlotEvalMean(), bm_PlotRangeSize(), bm_PlotResponseCurves()

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



# -------------------------------------------------------------
file.out <- paste0(myRespName, "/", myRespName, ".AllModels.models.out")
if (file.exists(file.out)) {
  myBiomodModelOut <- get(load(file.out))
} else {

  # Format Data with true absences
  myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                       resp.var = myResp,
                                       resp.xy = myRespXY,
                                       expl.var = myExpl)

  # Model single models
  myBiomodModelOut <- BIOMOD_Modeling(bm.format = myBiomodData,
                                      modeling.id = 'AllModels',
                                      models = c('RF', 'GLM'),
                                      CV.strategy = 'random',
                                      CV.nb.rep = 2,
```

```
                                    CV.perc = 0.8,
                                    OPT.strategy = 'bigboss',
                                    metric.eval = c('TSS','ROC'),
                                    var.import = 3,
                                    seed.val = 42)
}


# ----------------------------------------------------------------
# Get variables importance
get_variables_importance(myBiomodModelOut)

# Represent variables importance
bm_PlotVarImpBoxplot(bm.out = myBiomodModelOut, group.by = c('expl.var', 'algo', 'algo'))
bm_PlotVarImpBoxplot(bm.out = myBiomodModelOut, group.by = c('expl.var', 'algo', 'PA'))
bm_PlotVarImpBoxplot(bm.out = myBiomodModelOut, group.by = c('algo', 'expl.var', 'PA'))
```

`bm_PseudoAbsences` *Select pseudo-absences*

Description

This internal **biomod2** function allows to select pseudo-absences according to 4 different methods : `random`, `sre`, `disk` or `user.defined` (see Details).

Usage

```
bm_PseudoAbsences(
  resp.var,
  expl.var,
  strategy,
  nb.rep = NULL,
  nb.absences = NULL,
  sre.quant = NULL,
  dist.min = 0,
  dist.max = NULL,
  fact.aggr = NULL,
  user.table = NULL,
  seed.val = NULL
)

bm_PseudoAbsences_user.defined(resp.var, expl.var, ...)

## S4 method for signature 'ANY,SpatVector'
bm_PseudoAbsences_user.defined(resp.var, expl.var, user.table)
```

```
## S4 method for signature 'ANY,SpatRaster'
bm_PseudoAbsences_user.defined(resp.var, expl.var, user.table)

bm_PseudoAbsences_random(resp.var, expl.var, ...)

## S4 method for signature 'ANY,SpatVector'
bm_PseudoAbsences_random(resp.var, expl.var, nb.absences, nb.rep, fact.aggr)

## S4 method for signature 'ANY,SpatRaster'
bm_PseudoAbsences_random(resp.var, expl.var, nb.absences, nb.rep, fact.aggr)

bm_PseudoAbsences_sre(resp.var, expl.var, ...)

## S4 method for signature 'ANY,SpatVector'
bm_PseudoAbsences_sre(resp.var, expl.var, sre.quant, nb.absences, nb.rep)

## S4 method for signature 'ANY,SpatRaster'
bm_PseudoAbsences_sre(resp.var, expl.var, sre.quant, nb.absences, nb.rep)

bm_PseudoAbsences_disk(resp.var, expl.var, ...)

## S4 method for signature 'ANY,SpatVector'
bm_PseudoAbsences_disk(
  resp.var,
  expl.var,
  dist.min,
  dist.max,
  nb.absences,
  nb.rep,
  fact.aggr
)

## S4 method for signature 'ANY,SpatRaster'
bm_PseudoAbsences_disk(
  resp.var,
  expl.var,
  dist.min,
  dist.max,
  nb.absences,
  nb.rep,
  fact.aggr
)
```

Arguments

resp.var	a vector, SpatialPoints or SpatialPointsDataFrame object containing binary data (0 : absence, 1 : presence, NA : indeterminate) for a single species that will be used to find the pseudo-absences

expl.var	a matrix, data.frame, SpatialPointsDataFrame or SpatRaster object containing the explanatory variables (in columns or layers) that will be used to find the pseudo-absences
strategy	a character corresponding to the pseudo-absence selection strategy, must be among random, sre, disk or user.defined
nb.rep	(*optional, default* NULL) If strategy != 'user.defined', an integer corresponding to the number of sets (repetitions) of pseudo-absence points that will be drawn
nb.absences	(*optional, default* NULL) If strategy = 'random' or strategy = 'sre' or strategy = 'disk', an integer corresponding to the number of pseudo-absence points that will be selected for each pseudo-absence repetition (true absences included). It can also be a vector of the same length as nb.rep containing integer values corresponding to the different numbers of pseudo-absences to be selected (see Details)
sre.quant	(*optional, default* NULL) If strategy = 'sre', a numeric between 0 and 0.5 defining the half-quantile used to make the sre pseudo-absence selection (see bm_SRE)
dist.min	(*optional, default* 0) If strategy = 'disk', a numeric defining the minimal distance to presence points used to make the disk pseudo-absence selection (in the same projection system units as expl.var)
dist.max	(*optional, default* NULL) If strategy = 'disk', a numeric defining the maximal distance to presence points used to make the disk pseudo-absence selection (in the same projection system units as expl.var)
fact.aggr	(*optional, default* NULL) If strategy = 'random' or strategy = 'disk', an integer defining the factor of aggregation to reduce the spatial resolution of the environmental variables
user.table	(*optional, default* NULL) If strategy = 'user.defined', a matrix or data.frame with as many rows as resp.var values, as many columns as nb.rep, and containing TRUE or FALSE values defining which points will be used to build the species distribution model(s) for each repetition
seed.val	(*optional, default* NULL) An integer value corresponding to the new seed value to be set
...	(*optional, one or several of the listed above arguments depending on the selected method*)

Details

Concerning random selection :

The idea is to select pseudo-absences randomly in spatial locations where the species has not been sampled. This method is the simplest one and the most appropriate if lacking information about the presence sampling (non-exhaustive, biased sampling, etc).

Concerning SRE selection (see bm_SRE) :

The idea is to select pseudo-absences in spatial locations whose environmental conditions are different from those of the presence points. This method is appropriate when most of the environmental space of the species has been sampled.

Concerning disk selection :

The idea is to select pseudo-absences, not too close from presence points, but not too far away either. This method is appropriate when most of the spatial range of the species has been sampled.

Concerning user defined selection :

The user can provide pseudo-absences locations through a table containing spatial locations in rows, pseudo-absences repetitions in columns, and TRUE/FALSE values indicating whether each point is to be considered as pseudo-absence or not for each dataset.

Concerning multiple size selection :

It is possible to select create several pseudo-absence repetitions with different number of points, BUT with the same sampling strategy. nb.absences must contain as many values as the number of sets of pseudo-absences (nb.rep).

Value

A list containing the following elements :

- xy : the coordinates of the species observations
- sp : the values of the species observations (0, 1 or NA)
- env : the explanatory variables
- pa.tab : the corresponding table of selected pseudo-absences (indicated by TRUE or FALSE)

Author(s)

Wilfried Thuiller, Damien Georges

See Also

bm_SRE, BIOMOD.formated.data.PA, BIOMOD_FormatingData

Other Secondary functions: bm_BinaryTransformation(), bm_CrossValidation(), bm_FindOptimStat(), bm_MakeFormula(), bm_ModelAnalysis(), bm_ModelingOptions(), bm_PlotEvalBoxplot(), bm_PlotEvalMean(), bm_PlotRangeSize(), bm_PlotResponseCurves(), bm_PlotVarImpBoxplot(), bm_RangeSize(), bm_RunModelsLoop(), bm_SRE(), bm_SampleBinaryVector(), bm_SampleFactorLevels(), bm_Tuning(), bm_VariablesImportance()

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)




# ---------------------------------------------------------------- #
# Create the different pseudo-absence datasets

# Transform true absences into potential pseudo-absences
myResp.PA <- ifelse(myResp == 1, 1, NA)
myResp.PA.vect <- vect(cbind(myRespXY, myResp.PA), geom = c("X_WGS84","Y_WGS84"))

# random method
PA.r <- bm_PseudoAbsences(resp.var = myResp.PA.vect,
                          expl.var = myExpl,
                          nb.rep = 4,
                          nb.absences = 1000,
                          strategy = 'random')

# disk method
PA.d <- bm_PseudoAbsences(resp.var = myResp.PA.vect,
                          expl.var = myExpl,
                          nb.rep = 4,
                          nb.absences = 500,
                          strategy = 'disk',
                          dist.min = 5,
                          dist.max = 35)

# SRE method
PA.s <- bm_PseudoAbsences(resp.var = myResp.PA.vect,
                          expl.var = myExpl,
                          nb.rep = 4,
                          nb.absences = 1000,
                          strategy = 'sre',
```

```
                          sre.quant = 0.025)

# user.defined method
myPAtable <- data.frame(PA1 = ifelse(myResp == 1, TRUE, FALSE),
                        PA2 = ifelse(myResp == 1, TRUE, FALSE))
for (i in 1:ncol(myPAtable)) myPAtable[sample(which(myPAtable[, i] == FALSE), 500), i] = TRUE
PA.u <- bm_PseudoAbsences(resp.var = myResp.PA.vect,
                          expl.var = myExpl,
                          strategy = 'user.defined',
                          user.table = myPAtable)

str(PA.r)
head(PA.r$pa.tab)
apply(PA.r$pa.tab, 2, table)

head(PA.d$pa.tab)
apply(PA.d$pa.tab, 2, table)

head(PA.s$pa.tab)
apply(PA.s$pa.tab, 2, table)

tail(PA.u$pa.tab)
apply(PA.u$pa.tab, 2, table)


# random method : different number of PA
PA.r_mult <- bm_PseudoAbsences(resp.var = myResp.PA.vect,
                               expl.var = myExpl,
                               nb.rep = 4,
                               nb.absences = c(1000, 500, 500, 200),
                               strategy = 'random')

str(PA.r_mult)
head(PA.r_mult$pa.tab)
apply(PA.r_mult$pa.tab, 2, table)
```

---

bm_RangeSize *Analyze the range size differences between projections of species distribution models*

Description

This function allows to calculate the absolute number of locations (pixels) lost, stable and gained, as well as the corresponding relative proportions, between two (or more) binary projections of (ensemble) species distribution models (*which can represent new time scales or environmental scenarios for example*).

Usage

```
bm_RangeSize(proj.current, proj.future, ordinal = FALSE)

## S4 method for signature 'data.frame,data.frame'
bm_RangeSize(proj.current, proj.future, ordinal = FALSE)

## S4 method for signature 'SpatRaster,SpatRaster'
bm_RangeSize(proj.current, proj.future, ordinal = FALSE)
```

Arguments

`proj.current`	a `data.frame`, `RasterLayer` or `SpatRaster` object containing the initial binary projection(s) of the (ensemble) species distribution model(s)
`proj.future`	a `data.frame`, `RasterLayer` or `SpatRaster` object containing the final binary projection(s) of the (ensemble) species distribution model(s)
`ordinal`	(*optional, default* `FALSE`) A `logical` value defining whether the projections are to be considered as ordinal data or not

Details

Note that **this function is only relevant to compare binary projections, made on the same area with the same resolution**.

Comparison between `proj.current` and `proj.future` depends on the number of projection columns/layers in both objects :

`proj.current`	`proj.future`
1 projection (*e.g. data.frame with 1 column, SpatRaster with 1 layer*)	**1 projection** (*e.g. data.frame with 1 column, Spa*
`n` **projections** (*e.g. data.frame with n columns, SpatRaster with n layers*)	`n` **projections** (*e.g. data.frame with n columns, Spa*
`1` **projection** (*e.g. data.frame with 1 column, SpatRaster with 1 layer*)	`n` **projections** (*e.g. data.frame with n columns, Spa*

For binary data,`Diff.By.Pixel` object is obtained by applying the simple following formula :

$$proj.future - 2 * proj.current$$

For non binary data, `Diff.By.Pixel` object is obtained by rescaling the two projection to a 0 to 1 scale and applying the simple following formula :

$$proj.future - proj.current$$

Value

A `list` containing two objects :

Compt.By.Models a `data.frame` containing the summary of range change for each comparison

- `Loss` : number of pixels predicted to be lost
- `Stable_Abs` : number of pixels not currently occupied and not predicted to be

- `Stable_Pres` : number of pixels currently occupied and predicted to remain occupied
- `Gain` : number of pixels predicted to be gained

- `PercLoss` : percentage of pixels currently occupied and predicted to be lost
(`Loss / (Loss + Stable_Pres)`)
- `PercGain` : percentage of pixels predicted to be gained compare to the number of pixels currently occupied
(`Gain / (Loss + Stable_Pres)`)
- `SpeciesRangeChange` : percentage of pixels predicted to change compare to the number of pixels currently occupied
(`PercGain - PercLoss`)

- `CurrentRangeSize` : number of pixels currently occupied
(`Loss + Stable_Pres`)
- `FutureRangeSize.NoDisp` : number of pixels predicted to be occupied, assuming no migration
(`CurrentRangeSize`)
- `FutureRangeSize.FullDisp` : number of pixels predicted to be occupied, assuming migration
(`Gain + Stable_Pres`)

loss.gain an object in the same form than the input data (`proj.current` and `proj.future`) and containing a value for each point/pixel of each comparison among :

- `-2` : predicted to be lost
- `-1` : predicted to remain occupied
- `0` : predicted to remain unoccupied
- `1` : predicted to be gained

Diff.By.Pixel an object in the same form than the input data (`proj.current` and `proj.future`) and containing a value for each point/pixel of each comparison obtain with :

- `Future - 2* Current` for binary data
- `Future - Current` for nonbinary

Author(s)

Wilfried Thuiller, Damien Georges, Bruno Lafourcade

See Also

`BIOMOD_Projection`, `BIOMOD_EnsembleForecasting`, `bm_PlotRangeSize`

Other Secondary functions: `bm_BinaryTransformation()`, `bm_CrossValidation()`, `bm_FindOptimStat()`, `bm_MakeFormula()`, `bm_ModelAnalysis()`, `bm_ModelingOptions()`, `bm_PlotEvalBoxplot()`, `bm_PlotEvalMean()`, `bm_PlotRangeSize()`, `bm_PlotResponseCurves()`, `bm_PlotVarImpBoxplot()`, `bm_PseudoAbsences()`, `bm_RunModelsLoop()`, `bm_SRE()`, `bm_SampleBinaryVector()`, `bm_SampleFactorLevels()`, `bm_Tuning()`, `bm_VariablesImportance()`

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



# ------------------------------------------------------------------ #
file.out <- paste0(myRespName, "/", myRespName, ".AllModels.models.out")
if (file.exists(file.out)) {
  myBiomodModelOut <- get(load(file.out))
} else {

  # Format Data with true absences
  myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                       resp.var = myResp,
                                       resp.xy = myRespXY,
                                       expl.var = myExpl)

  # Model single models
  myBiomodModelOut <- BIOMOD_Modeling(bm.format = myBiomodData,
                                      modeling.id = 'AllModels',
                                      models = c('RF', 'GLM'),
                                      CV.strategy = 'random',
                                      CV.nb.rep = 2,
                                      CV.perc = 0.8,
                                      OPT.strategy = 'bigboss',
                                      metric.eval = c('TSS', 'ROC'),
                                      var.import = 3,
                                      seed.val = 42)
}

models.proj <- get_built_models(myBiomodModelOut, algo = "RF")
  # Project single models
  myBiomodProj <- BIOMOD_Projection(bm.mod = myBiomodModelOut,
                                    proj.name = 'CurrentRangeSize',
                                    new.env = myExpl,
```

```
                                    models.chosen = models.proj,
                                    metric.binary = 'all',
                                    build.clamping.mask = TRUE)


# ---------------------------------------------------------------- #
# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_future)
myExplFuture <- terra::rast(bioclim_future)

# Project onto future conditions
myBiomodProjectionFuture <- BIOMOD_Projection(bm.mod = myBiomodModelOut,
                                              proj.name = 'FutureRangeSize',
                                              new.env = myExplFuture,
                                              models.chosen = models.proj,
                                              metric.binary = 'TSS')

# Load current and future binary projections
CurrentProj <- get_predictions(myBiomodProj,
                               metric.binary = "TSS",
                               model.as.col = TRUE)
FutureProj <- get_predictions(myBiomodProjectionFuture,
                               metric.binary = "TSS",
                               model.as.col = TRUE)
# Compute differences
myBiomodRangeSize <- bm_RangeSize(proj.current = CurrentProj, proj.future = FutureProj)

myBiomodRangeSize$Compt.By.Models
plot(myBiomodRangeSize$Diff.By.Pixel)
```

`bm_RunModelsLoop` *Loop to compute all single species distribution models*

Description

This internal **biomod2** function allows the user to compute all single species distribution models (asked by the `BIOMOD_Modeling` function).

Usage

```
bm_RunModelsLoop(
  bm.format,
  modeling.id,
  models,
  models.pa,
  calib.lines,
  bm.options,
```

```
  metric.eval,
  var.import,
  weights,
  scale.models = FALSE,
  nb.cpu = 1,
  seed.val = NULL,
  do.progress = TRUE
)

bm_RunModel(
  model,
  run.name,
  dir.name = ".",
  modeling.id = "",
  Data,
  bm.options,
  calib.lines.vec,
  eval.data = NULL,
  metric.eval = c("AUCroc", "TSS", "KAPPA"),
  var.import = 0,
  weights.vec,
  scale.models = FALSE,
  nb.cpu = 1,
  seed.val = NULL,
  do.progress = TRUE
)
```

Arguments

bm.format	a BIOMOD.formated.data or BIOMOD.formated.data.PA object returned by the BIOMOD_FormatingData function
modeling.id	a character corresponding to the name (ID) of the simulation set (*a random number by default*)
models	a vector containing model names to be computed, must be among ANN, CTA, DNN, FDA, GAM, GBM, GLM, MARS, MAXENT, MAXNET, RF, RFd, SRE, XGBOOST
models.pa	(*optional, default* NULL) A list containing for each model a vector defining which pseudo-absence datasets are to be used, must be among colnames(bm.format@PA.table)
calib.lines	a matrix containing calibration / validation lines for each pseudo-absence (or allData) x repetition (or allRun) combination that can be obtained with the bm_CrossValidation function
bm.options	a BIOMOD.models.options object returned by the bm_ModelingOptions function
metric.eval	a vector containing evaluation metric names to be used, must be among AUCroc, AUCprg, TSS, KAPPA, ACCURACY, BIAS, POD, FAR, POFD, SR, CSI, ETS, OR, ORSS, BOYCE, MPA (*binary data*), RMSE, MAE, MSE, Rsquared, Rsquared_aj, Max_error (*abundance / count / relative data*), Accuracy, Recall, Precision, F1 (*multi-class/ordinal data*)

var.import	(*optional, default* NULL) An integer corresponding to the number of permutations to be done for each variable to estimate variable importance
weights	a matrix containing observation weights for each pseudo-absence (or allData) dataset
scale.models	(*optional, default* FALSE) A logical value defining whether all models predictions should be scaled with a binomial GLM or not
nb.cpu	(*optional, default* 1) An integer value corresponding to the number of computing resources to be used to parallelize the single models computation
seed.val	(*optional, default* NULL) An integer value corresponding to the new seed value to be set
do.progress	(*optional, default* TRUE) A logical value defining whether the progress bar is to be rendered or not
model	a character corresponding to the model name to be computed, must be either ANN, CTA, DNN, FDA, GAM, GBM, GLM, MARS, MAXENT, MAXNET, RF, SRE, XGBOOST
run.name	a character corresponding to the model to be run (sp.name + pa.id + run.id)
dir.name	(*optional, default* .) A character corresponding to the modeling folder
Data	a data.frame containing observations, coordinates and environmental variables that can be obtained with the get_species_data function
calib.lines.vec	a vector containing calibration / validation lines for the concerned pseudo-absence (or allData) x repetition (or allRun) combination
eval.data	(*optional, default* NULL) A data.frame containing validation observations, coordinates and environmental variables that can be obtained with the get_eval_data function
weights.vec	a vector containing observation weights the concerned pseudo-absence (or allData) dataset

Value

A list containing for each model a list containing the following elements :

- model : the name of correctly computed model
- calib.failure : the name of incorrectly computed model
- pred : the prediction outputs for calibration data
- pred.eval : the prediction outputs for evaluation data
- evaluation : the evaluation outputs returned by the bm_FindOptimStat function
- var.import : the mean of variables importance returned by the bm_VariablesImportance function

Author(s)

Damien Georges

See Also

rpart, prune, gbm, nnet, earth, fda, mars, maxnet, randomForest, xgboost, bm_ModelingOptions, BIOMOD_Modeling, bm_MakeFormula, bm_SampleFactorLevels, bm_FindOptimStat, bm_VariablesImportance

Other Secondary functions: bm_BinaryTransformation(), bm_CrossValidation(), bm_FindOptimStat(), bm_MakeFormula(), bm_ModelAnalysis(), bm_ModelingOptions(), bm_PlotEvalBoxplot(), bm_PlotEvalMean(), bm_PlotRangeSize(), bm_PlotResponseCurves(), bm_PlotVarImpBoxplot(), bm_PseudoAbsences(), bm_RangeSize(), bm_SRE(), bm_SampleBinaryVector(), bm_SampleFactorLevels(), bm_Tuning(), bm_VariablesImportance()

---

bm_SampleBinaryVector *Sample binary vector*

Description

This internal **biomod2** function allows the user to sample a binary vector keeping the same proportion of 0 and 1 as the initial vector.

Usage

```
bm_SampleBinaryVector(obs, ratio, as.logical = FALSE, seedval = NULL)
```

Arguments

obs	a vector containing binary values (either 0 or 1)
ratio	a numeric between 0 and 1 corresponding to the proportion of obs values to sample
as.logical	(*optional, default* FALSE) A logical value defining whether output should be returned as a vector of TRUE/FALSE values or integer values corresponding to the indices of obs elements to be kept
seedval	(*optional, default* NULL) An integer value corresponding to the new seed value to be set

Value

A list containing the following elements :

- calibration : elements selected for calibration
- validation : elements selected for validation (complementary to the calibration set)

Author(s)

Damien Georges

See Also

Other Secondary functions: bm_BinaryTransformation(), bm_CrossValidation(), bm_FindOptimStat(), bm_MakeFormula(), bm_ModelAnalysis(), bm_ModelingOptions(), bm_PlotEvalBoxplot(), bm_PlotEvalMean(), bm_PlotRangeSize(), bm_PlotResponseCurves(), bm_PlotVarImpBoxplot(), bm_PseudoAbsences(), bm_RangeSize(), bm_RunModelsLoop(), bm_SRE(), bm_SampleFactorLevels(), bm_Tuning(), bm_VariablesImportance()

Examples

```
## Generate a binary vector
vec.a <- sample(c(0, 1), 100, replace = TRUE)

## Generate calibration / validation datasets
bm_SampleBinaryVector(obs = vec.a, ratio = 0.7)
```

---

bm_SampleFactorLevels *Sample all levels of a factorial variable*

Description

This internal **biomod2** function allows the user to sample all levels of all the factorial variables contained in a data.frame or SpatRaster object.

Usage

```
bm_SampleFactorLevels(expl.var, mask.out = NULL, mask.in = NULL)
```

Arguments

expl.var	a data.frame or SpatRaster object containing the explanatory variables (in columns or layers)
mask.out	a data.frame or SpatRaster object containing the area that has already been sampled (*factor levels within this mask will not be sampled*)
mask.in	a data.frame or SpatRaster object containing areas where factor levels are to be sampled in priority. *Note that if after having explored these masks, some factor levels remain unsampled, they will be sampled in the reference input object* expl.var.

Details

The expl.var, mask.out and mask.in parameters must be coherent in terms of dimensions :

- same number of rows for data.frame objects
- same resolution, projection system and number of cells for SpatRaster objects

If mask.in contains several columns (data.frame) or layers (SpatRaster), then their order matters : they will be considered successively to sample missing factor levels.

- Values in data.frame will be understood as :
  - FALSE : out of mask
  - TRUE : in mask
- Values in SpatRaster will be understood as :
  - NA : out of mask
  - not NA : in mask

Value

A vector of numeric values corresponding to either row (data.frame) or cell (SpatRaster) numbers, each referring to a single level of a single factorial variable.

In case no factorial variable is found in the input object, NULL is returned.

Author(s)

Damien Georges

See Also

bm_PseudoAbsences, bm_CrossValidation

Other Secondary functions: bm_BinaryTransformation(), bm_CrossValidation(), bm_FindOptimStat(), bm_MakeFormula(), bm_ModelAnalysis(), bm_ModelingOptions(), bm_PlotEvalBoxplot(), bm_PlotEvalMean(), bm_PlotRangeSize(), bm_PlotResponseCurves(), bm_PlotVarImpBoxplot(), bm_PseudoAbsences(), bm_RangeSize(), bm_RunModelsLoop(), bm_SRE(), bm_SampleBinaryVector(), bm_Tuning(), bm_VariablesImportance()

Examples

```
library(terra)

## Create raster data
ras.1 <- ras.2 <- mask.out <- rast(nrows = 10, ncols = 10)
ras.1[] <- as.factor(rep(c(1, 2, 3, 4, 5), each = 20))
ras.1 <- as.factor(ras.1)
ras.2[] <- rnorm(100)
stk <- c(ras.1, ras.2)
names(stk) <- c("varFact", "varNorm")

## define a mask for already sampled points
mask.out[1:40] <- 1

## define a list of masks where we want to sample in priority
mask.in <- list(ras.1, ras.1)
mask.in[[1]][1:80] <- NA ## only level 5 should be sampled in this mask
```

```
mask.in[[1]][21:80] <- NA ## only levels 1 and 5 should be sampled in this mask

## Sample all factor levels
samp1 <- bm_SampleFactorLevels(expl.var = stk, mask.out = mask.out)
samp2 <- bm_SampleFactorLevels(expl.var = stk, mask.in = mask.in)
samp3 <- bm_SampleFactorLevels(expl.var = stk, mask.out = mask.out, mask.in = mask.in)
```

---

bm_SRE *Surface Range Envelope*

Description

This internal **biomod2** function allows the user to run a rectilinear surface range envelop (SRE) (equivalent to BIOCLIM) using the extreme percentiles (as recommended by Nix or Busby, see References and Details).

Usage

```
bm_SRE(
  resp.var = NULL,
  expl.var = NULL,
  new.env = NULL,
  quant = 0.025,
  do.extrem = FALSE
)
```

Arguments

resp.var	a vector, a SpatVector without associated data (*if presence-only*), or a SpatVector object containing binary data (0 : absence, 1 : presence, NA : indeterminate) for a single species that will be used to build the species distribution model(s) *Note that old format from* **sp** *are still supported such as* SpatialPoints *(if presence-only) or* SpatialPointsDataFrame *object containing binary data.*
expl.var	a matrix, data.frame, SpatVector or SpatRaster object containing the explanatory variables (in columns or layers) that will be used to build the SRE model *Note that old format from* **raster** *and* **sp** *are still supported such as* RasterStack *and* SpatialPointsDataFrame *objects.*
new.env	a matrix, data.frame, SpatVector or SpatRaster object containing the explanatory variables (in columns or layers) that will be used to predict the SRE model *Note that old format from* **raster** *and* **sp** *are still supported such as* RasterStack *and* SpatialPointsDataFrame *objects.*
quant	a numeric between 0 and 0.5 defining the half-quantile corresponding to the most extreme value for each variable not to be taken into account for determining the tolerance boundaries of the considered species (see Details)

`do.extrem` (*optional, default* `FALSE`)
A `logical` value defining whether a `matrix` containing extreme conditions supported should be returned or not

Details

Please refer to References to get more information about surface range envelop models.

This method is highly influenced by the extremes of the data input. Whereas a linear model can discriminate the extreme values from the main tendency, the SRE considers them as important as any other data point leading to changes in predictions.

The more (non-collinear) variables, the more restrictive the model will be.

Predictions are returned as binary (`0` or `1`) values, a site being either potentially suitable for all the variables, or out of bounds for at least one variable and therefore considered unsuitable.

`quant` determines the threshold from which the data will be taken into account for calibration. The default value of `0.05` induces that the `5%` most extreme values will be avoided for each variable on each side of its distribution along the gradient, meaning that a total of `10%` of the data will not be considered.

Value

A `vector` or a `SpatRaster` object, containing binary (`0` or `1`) values.

Author(s)

Wilfried Thuiller, Bruno Lafourcade, Damien Georges

References

- Nix HA (**1986**). *A biogeographic analysis of Australian elapid snakes.* In: Atlas of Elapid Snakes of Australia. (Ed.) R. Longmore, pp. 4-15. Australian Flora and Fauna Series Number 7. Australian Government Publishing Service: Canberra.
- Busby J (**1991**). *BIOCLIM - a bioclimate analysis and prediction system.* Plant protection quarterly, 6, 8-9.

See Also

`bm_PseudoAbsences`, `BIOMOD_FormatingData`, `bm_ModelingOptions`, `bm_Tuning`, `bm_RunModelsLoop`, `BIOMOD_Modeling`,

Other Secondary functions: `bm_BinaryTransformation()`, `bm_CrossValidation()`, `bm_FindOptimStat()`, `bm_MakeFormula()`, `bm_ModelAnalysis()`, `bm_ModelingOptions()`, `bm_PlotEvalBoxplot()`, `bm_PlotEvalMean()`, `bm_PlotRangeSize()`, `bm_PlotResponseCurves()`, `bm_PlotVarImpBoxplot()`, `bm_PseudoAbsences()`,

bm_RangeSize(), bm_RunModelsLoop(), bm_SampleBinaryVector(), bm_SampleFactorLevels(), bm_Tuning(), bm_VariablesImportance()

Examples

```
library(terra)

## Load real data
data(DataSpecies)
myResp.r <- as.numeric(DataSpecies[, 'GuloGulo'])

data(bioclim_current)
myExpl.r <- rast(bioclim_current)

myRespXY <- DataSpecies[which(myResp.r == 1), c('X_WGS84', 'Y_WGS84')]
myResp.v <- classify(subset(myExpl.r, 1),
                     matrix(c(-Inf, Inf, 0), ncol = 3, byrow = TRUE))
myResp.v[cellFromXY(myResp.v, myRespXY)] <- 1

## Compute SRE for several quantile values
sre.100 <- bm_SRE(resp.var = myResp.v,
                  expl.var = myExpl.r,
                  new.env = myExpl.r,
                  quant = 0)
sre.095 <- bm_SRE(resp.var = myResp.v,
                  expl.var = myExpl.r,
                  new.env = myExpl.r,
                  quant = 0.025)
sre.090 <- bm_SRE(resp.var = myResp.v,
                  expl.var = myExpl.r,
                  new.env = myExpl.r,
                  quant = 0.05)

## Visualize results
res <- c(myResp.v, sre.100, sre.095, sre.090)
names(res) <- c("Original distribution", "Full data calibration"
                , "Over 95 percent", "Over 90 percent")
plot(res)
```

---

bm_Tuning *Tune models parameters*

Description

This internal **biomod2** function allows to tune single model parameters and select more efficient ones based on an evaluation metric.

Usage

```
bm_Tuning(
  model,
  tuning.fun,
  do.formula = FALSE,
  do.stepAIC = FALSE,
  bm.options,
  bm.format,
  calib.lines = NULL,
  metric.eval = "TSS",
  metric.AIC = "AIC",
  weights = NULL,
  ctrl.train = NULL,
 params.train = list(ANN.size = c(2, 4, 6, 8), ANN.decay = c(0.01, 0.05, 0.1), ANN.bag =
   FALSE, DNN.hidden = list(depth = 3, width = 100), DNN.bias = TRUE, DNN.lambda =
   0.001, DNN.alpha = 1, DNN.lr = c(1e-04, 0.1), DNN.batchsize = 100, DNN.epochs = 150,
   FDA.degree = 1:2, FDA.nprune = 2:25, GAM.select = c(TRUE, FALSE), GAM.method =
   c("GCV.Cp", "GACV.Cp", "REML", "P-REML", "ML", "P-ML"), GAM.span = c(0.3, 0.5, 0.7),
   GAM.degree = 1, GBM.n.trees = c(500, 1000, 2500), GBM.interaction.depth = seq(2, 8,
    by = 3), GBM.shrinkage = c(0.001,
     0.01, 0.1), GBM.n.minobsinnode = 10,
   MARS.degree = 1:2, MARS.nprune = 2:max(21, 2 * ncol(bm.format@data.env.var) + 1),
   MAXENT.algorithm = "maxnet", MAXENT.parallel = TRUE, MAXENT.tune.args = list(rm =
    seq(0.5, 1, 0.5), fc = c("L")), MAXENT.partitions = "randomkfold",
   MAXENT.partition.settings = list(orientation = "lat_lon", aggregation.factor = 2,
    kfolds = 10), MAXENT.user.grp = NULL, RF.mtry = 1:min(10,
   ncol(bm.format@data.env.var)), RFd.mtry = 1:min(10, ncol(bm.format@data.env.var)),
    SRE.quant = c(0,
     0.0125, 0.025, 0.05, 0.1), XGBOOST.nrounds = 50,
    XGBOOST.max_depth = 1, XGBOOST.eta = c(0.3, 0.4), XGBOOST.gamma = 0,
    XGBOOST.colsample_bytree = c(0.6, 0.8), XGBOOST.min_child_weight = 1,
    XGBOOST.subsample = 0.5)
)
```

Arguments

model	a character corresponding to the algorithm to be tuned, must be either ANN, CTA, DNN, FDA, GAM, GBM, GLM, MARS, MAXENT, MAXNET, RF, RFd, SRE, XGBOOST
tuning.fun	a character corresponding to the model function name to be called through train function for tuning parameters (see ModelsTable dataset)
do.formula	(*optional, default* FALSE) A logical value defining whether formula is to be optimized or not
do.stepAIC	(*optional, default* FALSE) A logical value defining whether variables selection is to be performed for GLM and GAM models or not
bm.options	a BIOMOD.options.default or BIOMOD.options.dataset object returned by the bm_ModelingOptions function

bm.format	a BIOMOD.formated.data or BIOMOD.formated.data.PA object returned by the BIOMOD_FormatingData function
calib.lines	(*optional, default* NULL) A data.frame object returned by get_calib_lines or bm_CrossValidation functions
metric.eval	a character corresponding to the evaluation metric to be used, must be either AUC, Kappa or TSS for SRE only ; auc.val.avg, auc.diff.avg, or.mtp.avg, or.10p.avg, AICc for MAXENT only ; ROC or TSS for all other models
metric.AIC	a character corresponding to the AIC metric to be used, must be either AIC or BIC
weights	(*optional, default* NULL) A vector of numeric values corresponding to observation weights (one per observation, see Details)
ctrl.train	(*optional, default* NULL) A trainControl object
params.train	a list containing values of model parameters to be tested (see Details)

Details

Concerning ctrl.train **parameter :**

Set by default to :

```
ctrl.train <- caret::trainControl(method = "repeatedcv", repeats = 3, number = 10,
summaryFunction = caret::twoClassSummary,
classProbs = TRUE, returnData = FALSE)
```

Concerning params.train **parameter :**

All elements of the list must have names matching model.parameter_name format, parameter_name being one of the parameter of the tuning.fun function called by caret package and that can be found through the getModelInfo function.

Currently, the available parameters to be tuned are the following :

ANN size, decay, bag

CTA maxdepth

DNN hidden , bias, lambda, alpha, lr, batchsize, 150

FDA degree, nprune

GAM.gam span, degree

GAM.mgcv select, method

GBM n.trees, interaction.depth, shrinkage, n.minobsinnode

MARS degree, nprune

MAXENT algorithm,tune.args, parallel, partitions, partition.settings, user.grp

RF mtry

RFd mtry

SRE quant

XGBOOST nrounds, max_depth, eta, gamma, colsampl_bytree, min_child_weight, subsample

The expand.grid function is used to build a matrix containing all combinations of parameters to be tested.

Value

A BIOMOD.models.options object (see bm_ModelingOptions) with optimized parameters

Note

- No tuning for GLM and MAXNET
- MAXENT is tuned through ENMevaluate function which is calling either :
  - maxnet (by defining MAXENT.algorithm = 'maxnet') (*default*)
  - Java version of Maxent defined in **dismo** package (by defining MAXENT.algorithm = 'maxent.jar')
- DNN is tuned through tune function. The values include in params.train are the lower or upper range which hyperparameters are sampled. If there is only one value, the hyperparameter is fixed by biomod2 (inclunding the width and depth of hidden parameters.)
- SRE is tuned through bm_SRE function
- All other models are tuned through train function
- No optimization of formula for DNN, MAXENT, MAXNET, SRE and XGBOOST
- No interaction included in formula for CTA
- Variables selection only for GAM.gam and GLM

Author(s)

Frank Breiner, Maya Guéguen, Hélène Blancheteau

See Also

trainControl, train, ENMevaluate, ModelsTable, BIOMOD.models.options, bm_ModelingOptions, BIOMOD_Modeling

Other Secondary functions: bm_BinaryTransformation(), bm_CrossValidation(), bm_FindOptimStat(), bm_MakeFormula(), bm_ModelAnalysis(), bm_ModelingOptions(), bm_PlotEvalBoxplot(), bm_PlotEvalMean(), bm_PlotRangeSize(), bm_PlotResponseCurves(), bm_PlotVarImpBoxplot(), bm_PseudoAbsences(), bm_RangeSize(), bm_RunModelsLoop(), bm_SRE(), bm_SampleBinaryVector(), bm_SampleFactorLevels(), bm_VariablesImportance()

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



# ---------------------------------------------------------------- #
# Format Data with true absences
myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                     resp.var = myResp,
                                     resp.xy = myRespXY,
                                     expl.var = myExpl)


# ---------------------------------------------------------------- #
# List of all models currently available in `biomod2` (and their related package and function)
# Some of them can be tuned through the `train` function of the `caret` package
# (and corresponding training function to be used is indicated)
data(ModelsTable)
ModelsTable

allModels <- c('ANN', 'CTA', 'FDA', 'GAM', 'GBM', 'GLM'
               , 'MARS', 'MAXENT', 'MAXNET', 'RF', 'SRE', 'XGBOOST')

# default parameters
opt.d <- bm_ModelingOptions(data.type = 'binary',
                            models = allModels,
                            strategy = 'default')

# tune parameters for Random Forest model
tuned.rf <- bm_Tuning(model = 'RF',
                      tuning.fun = 'rf', ## see in ModelsTable
                      do.formula = FALSE,
                      bm.options = opt.d@options$RF.binary.randomForest.randomForest,
                      bm.format = myBiomodData)
tuned.rf
```

```
## Not run:
# tune parameters for GAM (from mgcv package) model
tuned.gam <- bm_Tuning(model = 'GAM',
                       tuning.fun = 'gam', ## see in ModelsTable
                       do.formula = TRUE,
                       do.stepAIC = TRUE,
                       bm.options = opt.d@options$GAM.binary.mgcv.gam,
                       bm.format = myBiomodData)
tuned.gam

## End(Not run)
```

`bm_VariablesImportance` *Variables' importance calculation*

Description

This internal **biomod2** function allows the user to compute a variable importance value for each variable involved in the given model.

Usage

```
bm_VariablesImportance(
  bm.model,
  expl.var,
  variables = NULL,
  method = "full_rand",
  nb.rep = 1,
  seed.val = NULL,
  do.progress = TRUE,
  temp.workdir = NULL
)
```

Arguments

`bm.model`	a `biomod2_model` object (or `nnet`, `rpart`, `fda`, `gam`, `glm`, `lm`, `gbm`, `mars`, `randomForest`, `xgb.Booster`) that can be obtained with the `get_formal_model` function
`expl.var`	a `data.frame` containing the explanatory variables that will be used to compute the variables importance
`variables`	(*optional, default* `NULL`) A `vector` containing the names of the explanatory variables that will be considered

method	a character corresponding to the randomization method to be used, must be full_rand (*only method available so far*)
nb.rep	an integer corresponding to the number of permutations to be done for each variable
seed.val	(*optional, default* NULL) An integer value corresponding to the new seed value to be set
do.progress	(*optional, default* TRUE) A logical value defining whether the progress bar is to be rendered or not
temp.workdir	(*optional, default* NULL) A character value corresponding to the folder name containing temporal prediction files when using MAXENT

Details

For each variable to be evaluated :

1. shuffle the original variable
2. compute model prediction with shuffled variable
3. calculate Pearson's correlation (Spearman for ordinal or multiclass data) between reference and shuffled predictions
4. return score as 1 - cor

The highest the value, the less reference and shuffled predictions are correlated, and the more influence the variable has on the model. A value of 0 assumes no influence of the variable on the model.

Note that this calculation does not account for variables' interactions.

The same principle is used in randomForest.

Value

A 3 columns data.frame containing variable's importance scores for each permutation run :

- expl.var : the considered explanatory variable (the one permuted)
- rand : the ID of the permutation run
- var.imp : the variable's importance score

Author(s)

Damien Georges

See Also

randomForest, bm_RunModelsLoop, BIOMOD_Modeling, BIOMOD_EnsembleModeling, bm_PlotVarImpBoxplot, get_variables_importance

Other Secondary functions: bm_BinaryTransformation(), bm_CrossValidation(), bm_FindOptimStat(), bm_MakeFormula(), bm_ModelAnalysis(), bm_ModelingOptions(), bm_PlotEvalBoxplot(), bm_PlotEvalMean(), bm_PlotRangeSize(), bm_PlotResponseCurves(), bm_PlotVarImpBoxplot(), bm_PseudoAbsences(), bm_RangeSize(), bm_RunModelsLoop(), bm_SRE(), bm_SampleBinaryVector(), bm_SampleFactorLevels(), bm_Tuning()

Examples

```
## Create simple simulated data
myResp.s <- sample(c(0, 1), 20, replace = TRUE)
myExpl.s <- data.frame(var1 = sample(c(0, 1), 100, replace = TRUE),
                       var2 = rnorm(100),
                       var3 = 1:100)

## Compute variables importance
mod <- glm(var1 ~ var2 + var3, data = myExpl.s)
bm_VariablesImportance(bm.model = mod,
                       expl.var = myExpl.s[, c('var2', 'var3')],
                       method = "full_rand",
                       nb.rep = 3)
```

`DataSpecies` *Presence-Absence data to build test SDM*

Description

A dataset covering all the continent with presence/absence data for 6 mammal species. Presence/absence were derived from range maps downloaded at IUCN.

Usage

```
DataSpecies
```

Format

A `data.frame` object with 2488 rows and 10 variables:

X_WGS84 Longitude

Y_WGS84 Latitude

ConnochaetesGnou Presence (1) or Absence (0) for black wildebeest

GuloGulo Presence (1) or Absence (0) for wolverine

PantheraOnca Presence (1) or Absence (0) for jaguar

PteropusGiganteus Presence (1) or Absence (0) for Indian flying fox

TenrecEcaudatus Presence (1) or Absence (0) for tailless tenrec

VulpesVulpes Presence (1) or Absence (0) for red fox

DataSTOC *Abundance to build test SDM*

Description

A dataset covering France with abundance data for 20 bird species. Data comes from the French Breeding Bird Survey (STOC program) of Vigie-Nature (Fontaine et al. 2020). The original dataset contains 2,948 sites in which bird species have been observed from 2001 to 2024. Original raw data is not provided here, but might be made available on request by contacting Benoît Fontaine (benoit.fontaine [at] mnhn.fr).

Usage

```
DataSTOC
```

Format

A `data.frame` object with 2006 rows and 30 columns (10 variables and 20 species):

site Observation site ID

period yearly average headcounts have been computed over 2 times periods: 2006-2011 and 2012-2017

X_WGS84 Longitude

Y_WGS84 Latitude

temp annual mean temperature (CHELSA)

precip annual mean precipitation (CHELSA)

cover_agri percentage of agricultural habitat cover around each point (European Union's Copernicus Land Monitoring Service information)

cover_water percentage of water habitat cover around each point (European Union's Copernicus Land Monitoring Service information)

cover_wet percentage of wetland habitat cover around each point (European Union's Copernicus Land Monitoring Service information)

sdiv_hab Shannon habitat diversity (CORINE)

Alauda.arvensis abundance data for Eurasian skylark

Cettia.cetti abundance data for Cetti's warbler

Coccothraustes.coccothraustes abundance data for hawfinch

Cuculus.canorus abundance data for common cuckoo

Emberiza.calandra abundance data for corn bunting

Emberiza.citrinella abundance data for yellowhammer

Erithacus.rubecula abundance data for European robin

Fulica.atra abundance data for Eurasian coot

Luscinia.megarhynchos abundance data for common nightingale

Passer.domesticus abundance data for house sparrow

Perdix.perdix abundance data for grey partridge

Periparus.ater abundance data for coal tit

Phylloscopus.bonelli abundance data for western Bonelli's warbler

Phylloscopus.trochilus abundance data for willow warbler

Regulus.regulus abundance data for goldcrest

Serinus.serinus abundance data for European serin

Sitta.europaea abundance data for Eurasian nuthatch

Streptopelia.decaocto abundance data for Eurasian collared dove

Sylvia.melanocephala abundance data for Sardinian warbler

Turdus.philomelos abundance data for song thrush

References

- STOC EPS - Vigie-Nature (**2025**). *French Breeding Bird Monitoring Scheme.* Muséum National d'Histoire Naturelle (MNHN) - Office Français pour la Biodiversité (OFB) - Ligue pour la Protection des Oiseaux (LPO).
- Fontaine B, Moussy C, Chiffard Carricaburu J, Dupuis J, Corolleur E, Schmaltz L, Lorrillière R, Loïs G, Gaudard C (**2020**). *Suivi des oiseaux communs en France 1989-2019 : 30 ans de suivis participatifs.* MNHN - Centre d'Ecologie et des Sciences de la Conservation, LPO BirdLife France - Service Connaissance, Ministère de la Transition écologique et solidaire. 46 pp.
- Brun P, Zimmermann NE, Hari C, Pellissier L, Karger DN (**2022**). *CHELSA-BIOCLIM+ A novel set of global climate-related predictors at kilometre-resolution.* EnviDat. doi:10.16904/envidat.332
- CORINE Land Cover 2018 (raster 100 m), Europe, 6-yearly - **version 2020_20u1**, May 2020. European Environment Agency. doi:10.2909/960998c118704e8280516485205ebbac

---

`getters.bm` *Functions to extract informations from* `biomod2_model` *objects*

Description

These functions allow the user to easily retrieve single models (formal or scaled) from `biomod2_model` objects from the modeling step.

Usage

```
## S4 method for signature 'biomod2_model'
get_formal_model(object)

## S4 method for signature 'biomod2_model'
get_scaling_model(object)
```

Arguments

object	a biomod2_model object

Value

get_formal_model an object from the model slot of a biomod2_model object

get_scaling_model an object from the scaling_model slot of a biomod2_model object

Author(s)

Damien Georges

See Also

biomod2_model

Other Toolbox functions: getters.out, load_stored_object(), predict2.bm, predict2.em, setters

---

getters.out *Functions to extract informations from* BIOMOD.models.out, BIOMOD.projection.out *or* BIOMOD.ensemble.models.out *objects*

Description

These functions allow the user to easily retrieve informations stored in the different **biomod2** objects from the different modeling steps, such as modeling options and formated data, models used or not, predictions, evaluations, variables importance.

Usage

```
## S4 method for signature 'BIOMOD.formated.data'
get_species_data(obj)

## S4 method for signature 'BIOMOD.formated.data.PA'
get_species_data(obj)

## S4 method for signature 'BIOMOD.formated.data'
get_eval_data(obj)

## S4 method for signature 'BIOMOD.models.out'
get_options(obj)

## S4 method for signature 'BIOMOD.models.out'
get_calib_lines(obj, as.data.frame = FALSE, PA = NULL, run = NULL)

## S4 method for signature 'BIOMOD.models.out'
```

```
get_formal_data(obj, subinfo = NULL)

## S4 method for signature 'BIOMOD.models.out'
get_predictions(
  obj,
  evaluation = FALSE,
  full.name = NULL,
  PA = NULL,
  run = NULL,
  algo = NULL,
  model.as.col = FALSE
)

## S4 method for signature 'BIOMOD.models.out'
get_built_models(obj, full.name = NULL, PA = NULL, run = NULL, algo = NULL)

## S4 method for signature 'BIOMOD.models.out'
get_evaluations(
  obj,
  full.name = NULL,
  PA = NULL,
  run = NULL,
  algo = NULL,
  metric.eval = NULL
)

## S4 method for signature 'BIOMOD.models.out'
get_variables_importance(
  obj,
  full.name = NULL,
  PA = NULL,
  run = NULL,
  algo = NULL,
  expl.var = NULL
)

## S4 method for signature 'BIOMOD.projection.out'
get_projected_models(
  obj,
  full.name = NULL,
  PA = NULL,
  run = NULL,
  algo = NULL,
  merged.by.algo = NULL,
  merged.by.run = NULL,
  merged.by.PA = NULL,
  filtered.by = NULL
)
```

```
## S4 method for signature 'BIOMOD.projection.out'
free(obj)

## S4 method for signature 'BIOMOD.projection.out'
get_predictions(
  obj,
  metric.binary = NULL,
  metric.filter = NULL,
  full.name = NULL,
  PA = NULL,
  run = NULL,
  algo = NULL,
  merged.by.algo = NULL,
  merged.by.run = NULL,
  merged.by.PA = NULL,
  filtered.by = NULL,
  model.as.col = FALSE,
  ...
)

## S4 method for signature 'BIOMOD.ensemble.models.out'
get_formal_data(obj, subinfo = NULL)

## S4 method for signature 'BIOMOD.ensemble.models.out'
get_built_models(
  obj,
  full.name = NULL,
  merged.by.algo = NULL,
  merged.by.run = NULL,
  merged.by.PA = NULL,
  filtered.by = NULL,
  algo = NULL
)

## S4 method for signature 'BIOMOD.ensemble.models.out'
get_kept_models(obj)

## S4 method for signature 'BIOMOD.ensemble.models.out'
get_predictions(
  obj,
  evaluation = FALSE,
  full.name = NULL,
  merged.by.algo = NULL,
  merged.by.run = NULL,
  merged.by.PA = NULL,
  filtered.by = NULL,
  algo = NULL,
```

```
  model.as.col = FALSE
)

## S4 method for signature 'BIOMOD.ensemble.models.out'
get_evaluations(
  obj,
  full.name = NULL,
  merged.by.algo = NULL,
  merged.by.run = NULL,
  merged.by.PA = NULL,
  filtered.by = NULL,
  algo = NULL,
  metric.eval = NULL
)

## S4 method for signature 'BIOMOD.ensemble.models.out'
get_variables_importance(
  obj,
  full.name = NULL,
  merged.by.algo = NULL,
  merged.by.run = NULL,
  merged.by.PA = NULL,
  filtered.by = NULL,
  algo = NULL,
  expl.var = NULL
)
```

Arguments

obj	a BIOMOD.formated.data, BIOMOD.formated.data.PA, BIOMOD.models.out, BIOMOD.projection.out or BIOMOD.ensemble.models.out object
as.data.frame	a logical defining whether output should be returned as data.frame or array object
PA	(*optional, default* NULL) A vector containing pseudo-absence set to be loaded, must be among PA1, PA2, ..., allData
run	(*optional, default* NULL) A vector containing repetition set to be loaded, must be among RUN1, RUN2, ..., allRun
subinfo	a character corresponding to the information to be extracted, must be among NULL, expl.var.names, resp.var, expl.var, MinMax, eval.resp.var, eval.expl.var (see Details)
evaluation	a logical defining whether evaluation data should be used or not
full.name	(*optional, default* NULL) A vector containing model names to be kept, must be either all or a sub-selection of model names that can be obtained with the get_built_models function

algo (*optional, default* NULL)
A character containing algorithm to be loaded, must be either ANN, CTA, DNN, FDA, GAM, GBM, GLM, MARS, MAXENT, MAXNET, RF, SRE, XGBOOST

model.as.col (*optional, default* FALSE)
A boolean given to get_predictions. If TRUE prediction are returned as a wide data.frame with each column containing predictions for a single model and corresponding to the old output given by **biomod2** in version < 4.2-2. If FALSE predictions are returned as a long data.frame with many additional informations readily available.

metric.eval (*optional, default* NULL)
A vector containing evaluation metric to be kept, must be among POD, FAR, POFD, SR, ACCURACY, BIAS, AUCroc, AUCprg, TSS, KAPPA, OR, ORSS, CSI, ETS, BOYCE, MPA

expl.var (*optional, default* NULL)
A vector containing explanatory variables to be kept, that can be obtained with the get_formal_data(obj, subinfo = 'expl.var.names') function

merged.by.algo (*optional, default* NULL)
A character containing merged algorithm to be loaded, must be among ANN, CTA, DNN, FDA, GAM, GBM, GLM, MARS, MAXENT, MAXNET, RF, SRE, XGBOOST, mergedAlgo

merged.by.run (*optional, default* NULL)
A vector containing merged repetition set to be loaded, must be among RUN1, RUN2, ..., mergedRun

merged.by.PA (*optional, default* NULL)
A vector containing merged pseudo-absence set to be loaded, must be among PA1, PA2, ..., mergedData

filtered.by (*optional, default* NULL)
A vector containing evaluation metric selected to filter single models to build the ensemble models, must be among POD, FAR, POFD, SR, ACCURACY, BIAS, AUCroc, AUCprg, TSS, KAPPA, OR, ORSS, CSI, ETS, BOYCE, MPA

metric.binary (*optional, default* NULL)
A vector containing evaluation metric selected to transform predictions into binary values, must be among POD, FAR, POFD, SR, ACCURACY, BIAS, AUCroc, AUCprg, TSS, KAPPA, OR, ORSS, CSI, ETS, BOYCE, MPA

metric.filter (*optional, default* NULL)
A vector containing evaluation metric to filter predictions, must be among POD, FAR, POFD, SR, ACCURACY, BIAS, AUCroc, AUCprg, TSS, KAPPA, OR, ORSS, CSI, ETS, BOYCE, MPA

... (*optional, one or several of the following arguments depending on the selected function*)

Value

get_species_data a data.frame combining data.species, coord, data.env.var (and PA.table) slots of BIOMOD.formated.data (or BIOMOD.formated.data.PA) object

get_eval_data a data.frame combining eval.data.species, eval.coord, eval.data.env.var slots of BIOMOD.formated.data or BIOMOD.formated.data.PA object

`get_options` a BIOMOD.stored.options-class object from the `models.options` slot of a BIOMOD.models.out-class object

`get_calib_lines` a BIOMOD.stored.data.frame-class object from the `calib.lines` slot of a BIOMOD.models.out object

`get_projected_models` a `vector` from the `models.projected` slot of a BIOMOD.projection.out object

`get_predictions` a BIOMOD.stored.data object from the `proj.out` slot of a BIOMOD.models.out, BIOMOD.projection.out or BIOMOD.ensemble.models.out object

`get_kept_models` a `vector` containing names of the kept models of a BIOMOD.ensemble.models.out object

`get_formal_data` depending on the `subinfo` parameter :

- `NULL` a BIOMOD.stored.formated.data-class (or BIOMOD.stored.models.out-class) object from the `formated.input.data` (or `models.out`) slot of a BIOMOD.models.out (or BIOMOD.ensemble.models.out) object
- `expl.var.names` a `vector` from the `expl.var.names` slot of a BIOMOD.models.out or BIOMOD.ensemble.models.ou object
- `resp.var` a `vector` from the `data.species` slot of the `formated.input.data` slot of a BIOMOD.models.out or BIOMOD.ensemble.models.out object
- `expl.var` a `data.frame` from the `data.env.var` slot of the `formated.input.data` slot of a BIOMOD.models.out or BIOMOD.ensemble.models.out object
- `MinMax` a `list` of minimum and maximum values (or levels if factorial) of variable contained in the `data.env.var` slot of the `formated.input.data` slot of a BIOMOD.models.out or BIOMOD.ensemble.models.out object
- `eval.resp.var` a `vector` from the `eval.data.species` slot of the `formated.input.data` slot of a BIOMOD.models.out or BIOMOD.ensemble.models.out object
- `eval.expl.var` a `data.frame` from the `eval.data.env.var` slot of the `formated.input.data` slot of a BIOMOD.models.out or BIOMOD.ensemble.models.out object

`get_built_models` a `vector` from the `models.computed` slot (or `em.computed`) of a BIOMOD.models.out (or BIOMOD.ensemble.models.out) object

`get_evaluations` a data.frame from the `models.evaluation` slot (or `model_evaluation` of each model in `em.computed`) of a BIOMOD.models.out (or BIOMOD.ensemble.models.out) object. Contains evaluation metric for different models and dataset. Evaluation metric are calculated on the calibrating data (column `calibration`), on the cross-validation data (column `validation`) or on the evaluation data (column `evaluation`).
For cross-validation data, see `CV.[...]` *parameters in* BIOMOD_Modeling *function ; for evaluation data, see* `eval.[...]` *parameters in* BIOMOD_FormatingData.

`get_variables_importance` a BIOMOD.stored.data.frame-class from the `variables.importance` slot (or `model_variables_importance` of each model in `em.models`) of a BIOMOD.models.out (or BIOMOD.ensemble.models.out) object

Author(s)

Damien Georges

See Also

BIOMOD.models.out, BIOMOD.projection.out, BIOMOD.ensemble.models.out

Other Toolbox functions: getters.bm, load_stored_object(), predict2.bm, predict2.em, setters

---

load_stored_object *Functions to load* BIOMOD.stored.data *objects*

Description

This functions allow the user to load BIOMOD.stored.data objects into memory.

Usage

```
load_stored_object(obj, ...)

## S4 method for signature 'BIOMOD.stored.data'
load_stored_object(obj, layer = 1)

## S4 method for signature 'BIOMOD.stored.SpatRaster'
load_stored_object(obj, layer = 1)
```

Arguments

obj	a BIOMOD.stored.data object
...	additional arguments
layer	an integer corresponding to the layer ID to be extracted when multilayer object considered

Author(s)

Damien Georges

See Also

BIOMOD.stored.data

Other Toolbox functions: getters.bm, getters.out, predict2.bm, predict2.em, setters

`ModelsTable` *Single models package and functions*

Description

A `data.frame` containing for each single model available in **biomod2** the package and functions to be called.

Usage

```
ModelsTable
```

Format

A `data.frame` object with 12 rows and 5 variables:

model all single models that can be computed in **biomod2**

type data type associated to the models

package `R` package used

func function used in the `R` package

train function called by **caret** for the tuning

All single models available are the following :

- ANN (`nnet`)
- CTA (`rpart`)
- DNN (`cito`)
- FDA (`fda`)
- GAM (`gam`, `gam` or `bam`)
- GBM (`gbm`)
- GLM (`glm`)
- MARS (`earth`)
- MAXENT (see Maxent website)
- MAXNET (`maxnet`)
- RF (`randomForest`)
- RFd (`randomForest` `downsampled`)
- SRE (`bm_SRE`)
- XGBOOST (`xgboost`)

ODMAP *ODMAP empty table*

Description

A `data.frame` containing ODMAP (Overview, Data, Model, Assessment, Prediction) protocol components.

Usage

```
ODMAP
```

Format

A `data.frame` object with 84 rows and 4 variables:

section Overview, Data, Model, Assessment or Prediction step

subsection corresponding field

element corresponding field

value to be filled with `BIOMOD_Report` function

OptionsBigboss *Bigboss pre-defined parameter values for single models*

Description

A `BIOMOD.models.options` object containing for each single model available in **biomod2** the parameter values pre-defined by **biomod2** team.

Usage

```
OptionsBigboss
```

Format

A `BIOMOD.models.options` object with some changed values :

`ANN.nnet.nnet`
- `size = 5`
- `decay = 0.1`
- `trace = FALSE`
- `rang = 0.1`
- `maxit = 200`

`CTA.rpart.rpart` • `control = list(xval = 5, minbucket = 5, minsplit = 5, cp = 0.001, maxdepth = 10)`

- cost = NULL

DNN.cito.dnn • batchsize = 100L
- epochs = 150L
- hidden = c(100L, 100L)
- lr = 0.05
- optimizer = "adam"
- lambda = 0.001
- alpha = 1.0
- validation = 0.2
- lr_scheduler = config_lr_scheduler("reduce_on_plateau", patience = 7)
- early_stopping = 14

FDA.mda.fda

GAM.gam.gam

GAM.mgcv.bam

GAM.mgcv.gam • method = 'GCV.Cp'
- control = list(epsilon = 1e-06, trace = FALSE, maxit = 100)

GBM.gbm.gbm • n.trees = 2500
- interaction.depth = 7
- n.minobsinnode = 5
- shrinkage = 0.001
- cv.folds = 3
- keep.data = FALSE
- n.cores = 1

GLM.stats.glm • mustart = 0.5
- control = glm.control(maxit = 50)

MARS.earth.earth • ncross = 0
- nk = NULL
- penalty = 2
- thresh = 0.001
- nprune = NULL
- pmethod = 'backward'

MAXENT.MAXENT.MAXENT • path_to_maxent.jar = '.'

RF.randomForest.randomForest • ntree = 500
- mtry = 2
- sampsize = NULL
- nodesize = 5
- maxnodes = NULL

RFd.randomForest.randomForest • type = 'classification'
- ntree = 500
- mtry = 2
- strata = factor(c(0, 1))

- sampsize = NULL
- nodesize = 5
- maxnodes = NULL

SRE.biomod2.bm_SRE
- do.extrem = TRUE

XGBOOST.xgboost.xgb_train
- params = list(max_depth = 2, eta = 1, nthread = 2)
- nrounds = 4

---

plot,BIOMOD.formated.data,missing-method *plot method for* BIOMOD.formated.data *and* BIOMOD.formated.data.PA *object class*

Description

Plot the spatial distribution of presences, absences and pseudo-absences among the different potential dataset (calibration, validation and evaluation). Available only if coordinates were given to BIOMOD_FormatingData.

Usage

```
## S4 method for signature 'BIOMOD.formated.data,missing'
plot(
  x,
  calib.lines = NULL,
  PA,
  run,
  plot.type,
  point.size = 1.5,
  plot.output,
  plot.valid = TRUE,
  plot.eval = TRUE,
  do.plot = TRUE,
  has.mask = FALSE,
  has.mask.eval = FALSE
)
```

Arguments

x	a BIOMOD.formated.data or BIOMOD.formated.data.PA object. Coordinates must be available to be able to use plot
calib.lines	(*optional, default* NULL) A data.frame object returned by get_calib_lines or bm_CrossValidation functions, to explore the distribution of calibration and validation datasets
PA	(*optional, default* 'all') If x is a BIOMOD.formated.data.PA object, a vector containing pseudo-absence set to be represented

run	(*optional, default* 'all') If calib.lines is provided, a vector containing repetition set to be represented
plot.type	a character defining how occurrences will be represented. Must be 'points' (*default*, better when using fine-grained data) or 'raster' (*if environmental variables were given as a raster*, better when using coarse-grained data)
point.size	(*optional, default* 1.5) If plot.type = 'points', a numeric to adjust the size of points
plot.output	a character defining whether to return a single facet with all plots or a list of individual plots, must be 'facet' (*default*) or list
plot.valid	(*optional, default* TRUE) A logical defining whether validation data should be separated on plot or not
plot.eval	(*optional, default* TRUE) A logical defining whether evaluation data should be added to the plot or not
do.plot	(*optional, default* TRUE) A logical defining whether the plot is to be rendered or not
has.mask	*optional, default* FALSE) A logical defining whether a mask for calibration data is available or not
has.mask.eval	(*optional, default* FALSE) A logical defining whether a mask for evaluation data is available or not

Value

a list with the data used to generate the plot and a ggplot2 object

Author(s)

Rémi Lemaire-Patin, Hélène Blancheteau

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)
```

```
## ------------------------------------------------------------------------ #
# Format Data with true absences
myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                     resp.var = myResp,
                                     resp.xy = myRespXY,
                                     expl.var = myExpl)
myBiomodData
plot(myBiomodData)
```

`setters` *Functions to change the place of the different biomod2 objects*

Description

This function allow the user to change the folder where the modelling of biomod2 have been done.

Usage

```
set_new_dirname(obj, new.dir.name)

## S4 method for signature 'BIOMOD.models.out'
set_new_dirname(obj, new.dir.name)

## S4 method for signature 'BIOMOD.ensemble.models.out'
set_new_dirname(obj, new.dir.name)

## S4 method for signature 'BIOMOD.projection.out'
set_new_dirname(obj, new.dir.name)
```

Arguments

`obj`	a `BIOMOD.formated.data`, `BIOMOD.formated.data.PA`, `BIOMOD.models.out`, `BIOMOD.projection.out` or `BIOMOD.ensemble.models.out` object
`new.dir.name`	a `character` corresponding to the new folder path

Author(s)

Hélène Blancheteau

See Also

`BIOMOD.models.out`, `BIOMOD.projection.out`, `BIOMOD.ensemble.models.out`

Other Toolbox functions: `getters.bm`, `getters.out`, `load_stored_object()`, `predict2.bm`, `predict2.em`

summary,BIOMOD.formated.data-method
summary *method for* BIOMOD.formated.data *object class*

Description

Summarize the number of presences, absences and pseudo-absences among the different potential dataset (calibration, validation and evaluation).

Usage

```
## S4 method for signature 'BIOMOD.formated.data'
summary(object, calib.lines = NULL)
```

Arguments

object	a BIOMOD.formated.data or BIOMOD.formated.data.PA object returned by the BIOMOD_FormatingData function
calib.lines	(*optional, default* NULL) an array object returned by get_calib_lines or bm_CrossValidation functions, to explore the distribution of calibration and validation datasets

Value

a data.frame

Author(s)

Rémi Lemaire-Patin

Examples

```
library(terra)

# Load species occurrences (6 species available)
data(DataSpecies)
head(DataSpecies)

# Select the name of the studied species
myRespName <- 'GuloGulo'

# Get corresponding presence/absence data
myResp <- as.numeric(DataSpecies[, myRespName])

# Get corresponding XY coordinates
myRespXY <- DataSpecies[, c('X_WGS84', 'Y_WGS84')]

# Load environmental variables extracted from BIOCLIM (bio_3, bio_4, bio_7, bio_11 & bio_12)
```

```
data(bioclim_current)
myExpl <- terra::rast(bioclim_current)



## ------------------------------------------------------------------------ #
# Format Data with true absences
myBiomodData <- BIOMOD_FormatingData(resp.name = myRespName,
                                     resp.var = myResp,
                                     resp.xy = myRespXY,
                                     expl.var = myExpl)
myBiomodData
summary(myBiomodData)
```

Index

∗ **ANN**
bm_RunModelsLoop, 111
∗ **CTA**
bm_RunModelsLoop, 111
∗ **DNN**
bm_RunModelsLoop, 111
∗ **FDA**
bm_RunModelsLoop, 111
∗ **GAM**
bm_RunModelsLoop, 111
∗ **GBM**
bm_RunModelsLoop, 111
∗ **GLM**
bm_RunModelsLoop, 111
∗ **MARS**
bm_RunModelsLoop, 111
∗ **MAXENT**
bm_RunModelsLoop, 111
∗ **Main functions**
BIOMOD_EnsembleForecasting, 30
BIOMOD_EnsembleModeling, 35
BIOMOD_FormatingData, 42
BIOMOD_LoadModels, 49
BIOMOD_Modeling, 51
BIOMOD_Projection, 58
BIOMOD_RangeSize, 62
∗ **ODMAP**
BIOMOD_Report, 65
∗ **Pearson**
bm_VariablesImportance, 124
∗ **Plot functions**
bm_ModelAnalysis, 81
bm_PlotEvalBoxplot, 88
bm_PlotEvalMean, 90
bm_PlotRangeSize, 93
bm_PlotResponseCurves, 96
bm_PlotVarImpBoxplot, 100
∗ **Primary functions**
BIOMOD_Report, 65
∗ **RF**
bm_RunModelsLoop, 111
∗ **SRE**
bm_PseudoAbsences, 102
bm_RunModelsLoop, 111
∗ **Secondary functions**
bm_BinaryTransformation, 69
bm_CrossValidation, 71
bm_FindOptimStat, 76
bm_MakeFormula, 80
bm_ModelAnalysis, 81
bm_ModelingOptions, 84
bm_PlotEvalBoxplot, 88
bm_PlotEvalMean, 90
bm_PlotRangeSize, 93
bm_PlotResponseCurves, 96
bm_PlotVarImpBoxplot, 100
bm_PseudoAbsences, 102
bm_RangeSize, 107
bm_RunModelsLoop, 111
bm_SampleBinaryVector, 114
bm_SampleFactorLevels, 115
bm_SRE, 117
bm_Tuning, 119
bm_VariablesImportance, 124
∗ **Spearman**
bm_VariablesImportance, 124
∗ **Toolbox functions**
getters.bm, 128
getters.out, 129
load_stored_object, 135
setters, 141
∗ **Toolbox objects**
BIOMOD.ensemble.models.out, 4
BIOMOD.formated.data, 7
BIOMOD.formated.data.PA, 11
BIOMOD.models.options, 16
BIOMOD.models.out, 17
BIOMOD.options.dataset, 19

BIOMOD.options.default, 21
BIOMOD.projection.out, 22
BIOMOD.rangesize.out, 25
BIOMOD.stored.data, 26
biomod2_ensemble_model, 27
biomod2_model, 29
∗ **XGBOOST**
bm_RunModelsLoop, 111
∗ **analyze**
bm_ModelAnalysis, 81
∗ **auc**
bm_FindOptimStat, 76
∗ **binary**
bm_BinaryTransformation, 69
bm_SampleBinaryVector, 114
∗ **boxplot**
bm_PlotEvalBoxplot, 88
bm_PlotVarImpBoxplot, 100
∗ **boyce**
bm_FindOptimStat, 76
∗ **convert**
bm_BinaryTransformation, 69
∗ **curve**
bm_PlotResponseCurves, 96
∗ **datasets**
bioclim_current, 3
bioclim_future, 4
DataSpecies, 126
DataSTOC, 127
ModelsTable, 136
ODMAP, 137
OptionsBigboss, 137
∗ **dataset**
BIOMOD_FormatingData, 42
∗ **disk**
bm_PseudoAbsences, 102
∗ **ensemble**
BIOMOD_EnsembleModeling, 35
∗ **evaluation**
BIOMOD_FormatingData, 42
bm_FindOptimStat, 76
bm_PlotEvalBoxplot, 88
bm_PlotEvalMean, 90
bm_PlotVarImpBoxplot, 100
∗ **factor**
bm_SampleFactorLevels, 115
∗ **filter**
bm_BinaryTransformation, 69
∗ **format**
BIOMOD_FormatingData, 42
∗ **formula**
bm_MakeFormula, 80
bm_RunModelsLoop, 111
∗ **gain**
BIOMOD_RangeSize, 62
bm_PlotRangeSize, 93
bm_RangeSize, 107
∗ **ggplot**
bm_PlotEvalBoxplot, 88
bm_PlotEvalMean, 90
bm_PlotRangeSize, 93
bm_PlotResponseCurves, 96
bm_PlotVarImpBoxplot, 100
∗ **html**
BIOMOD_Report, 65
∗ **importance**
bm_VariablesImportance, 124
∗ **loss**
BIOMOD_RangeSize, 62
bm_PlotRangeSize, 93
bm_RangeSize, 107
∗ **markdown**
BIOMOD_Report, 65
∗ **models**
BIOMOD_EnsembleForecasting, 30
BIOMOD_EnsembleModeling, 35
BIOMOD_Modeling, 51
BIOMOD_Projection, 58
bm_FindOptimStat, 76
bm_MakeFormula, 80
bm_ModelAnalysis, 81
bm_ModelingOptions, 84
bm_RunModelsLoop, 111
bm_SRE, 117
∗ **mpa**
bm_FindOptimStat, 76
∗ **multivariate**
BIOMOD_Modeling, 51
∗ **nonlinear**
BIOMOD_Modeling, 51
∗ **nonparametric**
BIOMOD_Modeling, 51
∗ **options**
bm_FindOptimStat, 76
bm_MakeFormula, 80
bm_ModelingOptions, 84

bm_RunModelsLoop, 111
∗ **projections**
BIOMOD_RangeSize, 62
bm_PlotRangeSize, 93
bm_RangeSize, 107
∗ **projection**
BIOMOD_EnsembleForecasting, 30
BIOMOD_Projection, 58
∗ **pseudo-absence**
BIOMOD_FormatingData, 42
bm_PseudoAbsences, 102
∗ **quantile**
bm_SRE, 117
∗ **random**
bm_PseudoAbsences, 102
bm_VariablesImportance, 124
∗ **range**
BIOMOD_RangeSize, 62
bm_PlotRangeSize, 93
bm_RangeSize, 107
bm_SRE, 117
∗ **regression**
BIOMOD_Modeling, 51
∗ **report**
BIOMOD_Report, 65
∗ **residuals**
bm_ModelAnalysis, 81
∗ **response**
bm_PlotResponseCurves, 96
∗ **sample**
bm_SampleBinaryVector, 114
bm_SampleFactorLevels, 115
∗ **shuffle**
bm_VariablesImportance, 124
∗ **species**
BIOMOD_RangeSize, 62
bm_PlotRangeSize, 93
bm_RangeSize, 107
∗ **sre**
bm_SRE, 117
∗ **surface**
bm_SRE, 117
∗ **threshold**
bm_BinaryTransformation, 69
∗ **tree**
BIOMOD_Modeling, 51
∗ **tss**
bm_FindOptimStat, 76
∗ **weights**
BIOMOD_EnsembleModeling, 35

ANN_biomod2_model-class
*(*biomod2_model*)*, 29

bam, *54*, *56*, *136*
bioclim_current, 3
bioclim_future, 4
BIOMOD.ensemble.models.out, 4, *5*, *10*, *15*, *16*, *18*, *20*, *22*, *23*, *25*, *26*, *28*, *30*, *31*, *39*, *49*, *66*, *88*, *89*, *91*, *92*, *96–98*, *100*, *101*, *129*, *132*, *134*, *135*, *141*
BIOMOD.ensemble.models.out-class
*(*BIOMOD.ensemble.models.out*)*, 4
BIOMOD.formated.data, 7, *9*, *20*, *26*, *46*, *52*, *66*, *72*, *84*, *112*, *121*, *132*, *133*, *139*, *141*, *142*
BIOMOD.formated.data(), *5*, *15*, *16*, *18*, *20*, *22*, *23*, *25*, *26*, *28*, *30*
BIOMOD.formated.data,data.frame,ANY-method
*(*BIOMOD.formated.data*)*, 7
BIOMOD.formated.data,numeric,data.frame-method
*(*BIOMOD.formated.data*)*, 7
BIOMOD.formated.data,numeric,matrix-method
*(*BIOMOD.formated.data*)*, 7
BIOMOD.formated.data,numeric,SpatRaster-method
*(*BIOMOD.formated.data*)*, 7
BIOMOD.formated.data-class
*(*BIOMOD.formated.data*)*, 7
BIOMOD.formated.data.PA, 11, *20*, *46*, *52*, *66*, *72*, *84*, *105*, *112*, *121*, *132*, *133*, *139*, *141*, *142*
BIOMOD.formated.data.PA(), *5*, *10*, *16*, *18*, *20*, *22*, *23*, *25*, *26*, *28*, *30*
BIOMOD.formated.data.PA,numeric,data.frame-method
*(*BIOMOD.formated.data.PA*)*, 11
BIOMOD.formated.data.PA,numeric,SpatRaster-method
*(*BIOMOD.formated.data.PA*)*, 11
BIOMOD.formated.data.PA-class
*(*BIOMOD.formated.data.PA*)*, 11
BIOMOD.models.options, *5*, *10*, *15*, *16*, 16, *18*, *20*, *22*, *23*, *25*, *26*, *28*, *30*, *53*, *55*, *85*, *86*, *112*, *122*, *137*
BIOMOD.models.options-class
*(*BIOMOD.models.options*)*, 16
BIOMOD.models.out, *5*, *10*, *15–17*, 17, *20*, *22*, *23*, *25*, *26*, *28*, *30*, *35*, *49*, *56*, *59*, *66*,

81, *88, 89*, *91, 92*, *96–98*, *100, 101*, *129*, *132*, *134, 135*, *141*
BIOMOD.models.out-class *(*BIOMOD.models.out*)*, 17
BIOMOD.options.dataset, *16*, 19, *20*, *22*, *55*, *120*
BIOMOD.options.dataset(), *5*, *10*, *15, 16*, *18*, *22, 23*, *25, 26*, *28*, *30*
BIOMOD.options.dataset,character-method *(*BIOMOD.options.dataset*)*, 19
BIOMOD.options.dataset-class *(*BIOMOD.options.dataset*)*, 19
BIOMOD.options.default, *16*, *20*, 21, *120*
BIOMOD.options.default(), *5*, *10*, *15, 16*, *18*, *20*, *23*, *25, 26*, *28*, *30*
BIOMOD.options.default,character,character-method *(*BIOMOD.options.default*)*, 21
BIOMOD.options.default-class *(*BIOMOD.options.default*)*, 21
BIOMOD.projection.out, *5*, *10*, *15, 16*, *18*, *20*, *22*, 22, *23*, *25, 26*, *28*, *30–32*, *60*, *129*, *132*, *134, 135*, *141*
BIOMOD.projection.out-class *(*BIOMOD.projection.out*)*, 22
BIOMOD.rangesize.out, *5*, *10*, *15, 16*, *18*, *20*, *22, 23*, 25, *26*, *28*, *30*
BIOMOD.rangesize.out-class *(*BIOMOD.rangesize.out*)*, 25
BIOMOD.stored.data, *5*, *10*, *15, 16*, *18*, *20*, *22, 23*, *25*, 26, *28*, *30*, *134, 135*
BIOMOD.stored.data-class *(*BIOMOD.stored.data*)*, 26
BIOMOD.stored.data.frame-class *(*BIOMOD.stored.data*)*, 26
BIOMOD.stored.files-class *(*BIOMOD.stored.data*)*, 26
BIOMOD.stored.formated.data-class *(*BIOMOD.stored.data*)*, 26
BIOMOD.stored.models.out-class *(*BIOMOD.stored.data*)*, 26
BIOMOD.stored.options-class *(*BIOMOD.stored.data*)*, 26
BIOMOD.stored.SpatRaster-class *(*BIOMOD.stored.data*)*, 26
biomod2_ensemble_model, *5*, *10*, *15, 16*, *18*, *20*, *22, 23*, *25–27*, 27, *30*
biomod2_ensemble_model-class *(*biomod2_ensemble_model*)*, 27
biomod2_model, *5*, *10*, *15, 16*, *18*, *20*, *22, 23*, *25, 26*, *28, 29*, 29, *128, 129*
biomod2_model-class *(*biomod2_model*)*, 29
BIOMOD_EnsembleForecasting, *4*, *22, 23*, *26*, 30, *40*, *60*, *64*, *70*, *109*
BIOMOD_EnsembleForecasting(), *40*, *46*, *50*, *57*, *61*, *64*
BIOMOD_EnsembleModeling, *4, 5*, *17, 18*, *26–28*, *30–33*, 35, *49, 50*, *56, 57*, *61*, *66*, *79*, *88, 89*, *91, 92*, *96–98*, *100, 101*, *125*
BIOMOD_EnsembleModeling(), *33*, *46*, *50*, *57*, *61*, *64*
BIOMOD_FormatingData, *7*, *10, 11*, *15*, *17*, *20*, *31*, *33*, *37*, *40*, 42, *52*, *54–56*, *59*, *66*, *72*, *75*, *84, 85*, *97*, *105*, *112*, *118*, *121*, *134*, *139*, *142*
BIOMOD_FormatingData(), *33*, *40*, *50*, *57*, *61*, *64*
BIOMOD_LoadModels, *4, 5*, *17, 18*, 49
BIOMOD_LoadModels(), *33*, *40*, *46*, *57*, *61*, *64*
BIOMOD_Modeling, *5*, *7*, *10, 11*, *15–22*, *26*, *29, 30*, *33*, *35*, *37*, *40*, *46*, *49, 50*, 51, *56*, *58, 59*, *61*, *66*, *75*, *79*, *81*, *85, 86*, *88, 89*, *91, 92*, *96–98*, *100, 101*, *111*, *114*, *118*, *122*, *125*, *134*
BIOMOD_Modeling(), *33*, *40*, *46*, *50*, *61*, *64*
BIOMOD_PresenceOnly, *4, 5*, *17, 18*
BIOMOD_Projection, *17, 18*, *22, 23*, *26*, *31*, *56, 57*, 58, *64*, *70*, *85*, *109*
BIOMOD_Projection(), *33*, *40*, *46*, *50*, *57*, *64*
BIOMOD_RangeSize, *25*, *33*, *61*, 62, *93, 94*
BIOMOD_RangeSize(), *33*, *40*, *46*, *50*, *57*, *61*
BIOMOD_Report, 65, *137*
bm_BinaryTransformation, 69
bm_BinaryTransformation(), *75*, *79*, *81, 82*, *86*, *89*, *92*, *94*, *98*, *101*, *105*, *109*, *114–116*, *118*, *122*, *125*
bm_BinaryTransformation,data.frame-method *(*bm_BinaryTransformation*)*, 69
bm_BinaryTransformation,matrix-method *(*bm_BinaryTransformation*)*, 69
bm_BinaryTransformation,numeric-method *(*bm_BinaryTransformation*)*, 69
bm_BinaryTransformation,SpatRaster-method *(*bm_BinaryTransformation*)*, 69
bm_CalculateStatAbun *(*bm_FindOptimStat*)*, 76

bm_CalculateStatAbund
(bm_FindOptimStat), 76
bm_CalculateStatBin (bm_FindOptimStat), 76
bm_CrossValidation, 7, 10, 11, 15, 20, 40, 46, 55, 56, 71, 84, 112, 116, 121, 139, 142
bm_CrossValidation(), 70, 79, 81, 82, 86, 89, 92, 94, 98, 101, 105, 109, 114–116, 118, 122, 125
bm_CrossValidation_block
(bm_CrossValidation), 71
bm_CrossValidation_block,BIOMOD.formated.data-method
(bm_CrossValidation), 71
bm_CrossValidation_block,BIOMOD.formated.data.PA-method
(bm_CrossValidation), 71
bm_CrossValidation_env
(bm_CrossValidation), 71
bm_CrossValidation_env,BIOMOD.formated.data-method
(bm_CrossValidation), 71
bm_CrossValidation_env,BIOMOD.formated.data.PA-method
(bm_CrossValidation), 71
bm_CrossValidation_kfold
(bm_CrossValidation), 71
bm_CrossValidation_kfold,BIOMOD.formated.data-method
(bm_CrossValidation), 71
bm_CrossValidation_kfold,BIOMOD.formated.data.PA-method
(bm_CrossValidation), 71
bm_CrossValidation_random
(bm_CrossValidation), 71
bm_CrossValidation_random,BIOMOD.formated.data-method
(bm_CrossValidation), 71
bm_CrossValidation_random,BIOMOD.formated.data.PA-method
(bm_CrossValidation), 71
bm_CrossValidation_strat
(bm_CrossValidation), 71
bm_CrossValidation_strat,BIOMOD.formated.data-method
(bm_CrossValidation), 71
bm_CrossValidation_strat,BIOMOD.formated.data.PA-method
(bm_CrossValidation), 71
bm_CrossValidation_user.defined
(bm_CrossValidation), 71
bm_CrossValidation_user.defined,BIOMOD.formated.data-method
(bm_CrossValidation), 71
bm_CrossValidation_user.defined,BIOMOD.formated.data.PA-method
(bm_CrossValidation), 71
bm_FindOptimStat, 76, 113, 114
bm_FindOptimStat(), 70, 75, 81, 82, 86, 89, 92, 94, 98, 101, 105, 109, 114–116, 118, 122, 125
bm_MakeFormula, 80, 114
bm_MakeFormula(), 70, 75, 79, 82, 86, 89, 92, 94, 98, 101, 105, 109, 114–116, 118, 122, 125
bm_ModelAnalysis, 81
bm_ModelAnalysis(), 70, 75, 79, 81, 86, 89, 92, 94, 98, 101, 105, 109, 114–116, 118, 122, 125
bm_ModelingOptions, 16, 17, 19–22, 33, 40, 53–56, 81, 84, 112, 114, 118, 120, 122
bm_ModelingOptions(), 70, 75, 79, 81, 82, 86, 89, 92, 94, 98, 101, 105, 109, 114–116, 118, 122, 125
bm_PlotEvalBoxplot, 5, 18, 40, 57, 88
bm_PlotEvalBoxplot(), 70, 75, 79, 81, 82, 86, 92, 94, 98, 101, 105, 109, 114–116, 118, 122, 125
bm_PlotEvalMean, 5, 18, 40, 57, 90
bm_PlotEvalMean(), 70, 75, 79, 81, 82, 86, 89, 94, 98, 101, 105, 109, 114–116, 118, 122, 125
bm_PlotRangeSize, 25, 64, 93, 109
bm_PlotRangeSize(), 70, 75, 79, 81, 82, 86, 89, 92, 98, 101, 105, 109, 114–116, 118, 122, 125
bm_PlotResponseCurves, 5, 18, 40, 57, 96
bm_PlotResponseCurves(), 70, 75, 79, 81, 82, 86, 89, 92, 94, 101, 105, 109, 114–116, 118, 122, 125
bm_PlotVarImpBoxplot, 5, 18, 40, 57, 100, 125
bm_PlotVarImpBoxplot(), 70, 75, 79, 81, 82, 86, 89, 92, 94, 98, 105, 109, 114–116, 118, 122, 125
bm_PseudoAbsences, 14, 15, 45, 46, 102, 116, 118
bm_PseudoAbsences(), 70, 75, 79, 81, 82, 86, 89, 92, 94, 98, 101, 109, 114–116, 118, 122, 125
bm_PseudoAbsences_disk
(bm_PseudoAbsences), 102
bm_PseudoAbsences_disk,ANY,SpatRaster-method
(bm_PseudoAbsences), 102
bm_PseudoAbsences_disk,ANY,SpatVector-method
(bm_PseudoAbsences), 102

bm_PseudoAbsences_random (bm_PseudoAbsences), 102
bm_PseudoAbsences_random,ANY,SpatRaster-method (bm_PseudoAbsences), 102
bm_PseudoAbsences_random,ANY,SpatVector-method (bm_PseudoAbsences), 102
bm_PseudoAbsences_sre (bm_PseudoAbsences), 102
bm_PseudoAbsences_sre,ANY,SpatRaster-method (bm_PseudoAbsences), 102
bm_PseudoAbsences_sre,ANY,SpatVector-method (bm_PseudoAbsences), 102
bm_PseudoAbsences_user.defined (bm_PseudoAbsences), 102
bm_PseudoAbsences_user.defined,ANY,SpatRaster-method (bm_PseudoAbsences), 102
bm_PseudoAbsences_user.defined,ANY,SpatVector-method (bm_PseudoAbsences), 102
bm_RangeSize, *94*, 107
bm_RangeSize(), *70*, *75*, *79*, *81, 82*, *86*, *89*, *92*, *94*, *98*, *101*, *105*, *114–116*, *119*, *122*, *125*
bm_RangeSize,data.frame,data.frame-method (bm_RangeSize), 107
bm_RangeSize,SpatRaster,SpatRaster-method (bm_RangeSize), 107
bm_RunModel, *29, 30*
bm_RunModel (bm_RunModelsLoop), 111
bm_RunModelsLoop, *10*, *15*, *20*, *22*, *79*, *81*, 111, *118*, *125*
bm_RunModelsLoop(), *70*, *75*, *79*, *81, 82*, *86*, *89*, *92*, *94*, *98*, *101*, *105*, *109*, *115, 116*, *119*, *122*, *125*
bm_SampleBinaryVector, 114
bm_SampleBinaryVector(), *70*, *75*, *79*, *81, 82*, *86*, *89*, *92*, *94*, *98*, *101*, *105*, *109*, *114*, *116*, *119*, *122*, *125*
bm_SampleFactorLevels, *114*, 115
bm_SampleFactorLevels(), *70*, *75*, *79*, *81, 82*, *86*, *89*, *92*, *94*, *98*, *101*, *105*, *109*, *114, 115*, *119*, *122*, *125*
bm_SRE, *46*, *54*, *104, 105*, 117, *122*, *136*
bm_SRE(), *70*, *75*, *79*, *81, 82*, *86*, *89*, *92*, *94*, *98*, *101*, *105*, *109*, *114–116*, *122*, *125*
bm_Tuning, *7*, *10, 11*, *15, 16*, *20*, *22*, *55, 56*, *81*, *85, 86*, *118*, 119
bm_Tuning(), *70*, *75*, *79*, *81, 82*, *86*, *89*, *92*, *94*, *98*, *101*, *105*, *109*, *114–116*, *119*, *125*
bm_VariablesImportance, *5*, *18*, *40*, *56*, *113, 114*, 124
bm_VariablesImportance(), *70*, *75*, *79*, *81, 82*, *86*, *89*, *92*, *94*, *98*, *101*, *105*, *109*, *114–116*, *119*, *122*

cito, *54*, *56*, *136*
CTA_biomod2_model-class (biomod2_model), 29

DataSpecies, 126
DataSTOC, 127
DNN_biomod2_model-class (biomod2_model), 29

earth, *54*, *56*, *114*, *136*
EMca_biomod2_model-class (biomod2_ensemble_model), 27
EMci_biomod2_model-class (biomod2_ensemble_model), 27
EMcv_biomod2_model-class (biomod2_ensemble_model), 27
EMfreq_biomod2_model-class (biomod2_ensemble_model), 27
EMmean_biomod2_model-class (biomod2_ensemble_model), 27
EMmedian_biomod2_model-class (biomod2_ensemble_model), 27
EMmode_biomod2_model-class (biomod2_ensemble_model), 27
EMwmean_biomod2_model-class (biomod2_ensemble_model), 27
ENMevaluate, *122*
expand.grid, *122*

facet_wrap, *23*, *89*
fda, *54*, *56*, *114*, *136*
FDA_biomod2_model-class (biomod2_model), 29
formalArgs, *85*
formals, *85*
formula, *80, 81*
free (getters.out), 129
free,BIOMOD.projection.out-method (getters.out), 129

gam, *54*, *56*, *136*
GAM_biomod2_model-class (biomod2_model), 29

gbm, *54*, *56*, *114*, *136*
GBM_biomod2_model-class (biomod2_model), 29
geom_point, *23*
get.block, *75*
get_built_models, *31*, *36*, *37*, *49*, *59*, *81*, *97*, *132*
get_built_models (getters.out), 129
get_built_models,BIOMOD.ensemble.models.out-method (getters.out), 129
get_built_models,BIOMOD.models.out-method (getters.out), 129
get_calib_lines, *20*, *84*, *121*, *139*, *142*
get_calib_lines (getters.out), 129
get_calib_lines,BIOMOD.models.out-method (getters.out), 129
get_eval_data (getters.out), 129
get_eval_data,BIOMOD.formated.data-method (getters.out), 129
get_evaluations, *56*, *89*, *91*, *92*
get_evaluations (getters.out), 129
get_evaluations,BIOMOD.ensemble.models.out-method (getters.out), 129
get_evaluations,BIOMOD.models.out-method (getters.out), 129
get_formal_data, *133*
get_formal_data (getters.out), 129
get_formal_data,BIOMOD.ensemble.models.out-method (getters.out), 129
get_formal_data,BIOMOD.models.out-method (getters.out), 129
get_formal_model, *124*
get_formal_model (getters.bm), 128
get_formal_model,biomod2_model-method (getters.bm), 128
get_kept_models (getters.out), 129
get_kept_models,BIOMOD.ensemble.models.out-method (getters.out), 129
get_optim_value, *55*, *77*
get_optim_value (bm_FindOptimStat), 76
get_options (getters.out), 129
get_options,BIOMOD.models.out-method (getters.out), 129
get_predictions, *23*, *133*
get_predictions (getters.out), 129
get_predictions,BIOMOD.ensemble.models.out-method (getters.out), 129
get_predictions,BIOMOD.models.out-method (getters.out), 129
get_predictions,BIOMOD.projection.out-method (getters.out), 129
get_projected_models, *63*
get_projected_models (getters.out), 129
get_projected_models,BIOMOD.projection.out-method (getters.out), 129
get_scaling_model (getters.bm), 128
get_scaling_model,biomod2_model-method (getters.bm), 128
get_species_data (getters.out), 129
get_species_data,BIOMOD.formated.data-method (getters.out), 129
get_species_data,BIOMOD.formated.data.PA-method (getters.out), 129
get_variables_importance, *101*, *125*
get_variables_importance (getters.out), 129
get_variables_importance,BIOMOD.ensemble.models.out-method (getters.out), 129
get_variables_importance,BIOMOD.models.out-method (getters.out), 129
getModelInfo, *121*
getters.bm, 128, *135*, *141*
getters.out, *129*, 129, *135*, *141*
glm, *54*, *56*, *136*
GLM_biomod2_model-class (biomod2_model), 29

kfold, *75*

load, *40*, *56*
load_stored_object, 135
load_stored_object(), *129*, *135*, *141*
load_stored_object,BIOMOD.stored.data-method (load_stored_object), 135
load_stored_object,BIOMOD.stored.SpatRaster-method (load_stored_object), 135

mars, *114*
MARS_biomod2_model-class (biomod2_model), 29
match.call, *67*
MAXENT_biomod2_model-class (biomod2_model), 29
maxnet, *54*, *56*, *114*, *136*
MAXNET_biomod2_model-class (biomod2_model), 29
ModelsTable, *85*, *86*, *120*, *122*, 136

nnet, *54*, *56*, *114*, *136*

ODMAP, *67*, 137
OptionsBigboss, *55*, *85*, *86*, 137

PackedSpatRaster, *26*
plot, *23*
plot,BIOMOD.formated.data,missing-method, 139
plot,BIOMOD.projection.out,missing-method *(*BIOMOD.projection.out*)*, 22
predict2.bm, *129*, *135*, *141*
predict2.em, *129*, *135*, *141*
print,BIOMOD.models.options-method *(*BIOMOD.models.options*)*, 16
print,BIOMOD.options.dataset-method *(*BIOMOD.options.dataset*)*, 19
prune, *114*

randomForest, *54*, *56*, *114*, *125*, *136*
RasterLayer, *108*
RF_biomod2_model-class *(*biomod2_model*)*, 29
RFd_biomod2_model-class *(*biomod2_model*)*, 29
rpart, *54*, *56*, *114*, *136*

s, *80*, *81*
set_new_dirname *(*setters*)*, 141
set_new_dirname,BIOMOD.ensemble.models.out-method *(*setters*)*, 141
set_new_dirname,BIOMOD.models.out-method *(*setters*)*, 141
set_new_dirname,BIOMOD.projection.out-method *(*setters*)*, 141
setters, *129*, *135*, 141
show,BIOMOD.ensemble.models.out-method *(*BIOMOD.ensemble.models.out*)*, 4
show,BIOMOD.formated.data-method *(*BIOMOD.formated.data*)*, 7
show,BIOMOD.models.options-method *(*BIOMOD.models.options*)*, 16
show,BIOMOD.models.out-method *(*BIOMOD.models.out*)*, 17
show,BIOMOD.options.dataset-method *(*BIOMOD.options.dataset*)*, 19
show,BIOMOD.projection.out-method *(*BIOMOD.projection.out*)*, 22
show,biomod2_ensemble_model-method *(*biomod2_ensemble_model*)*, 27
show,biomod2_model-method *(*biomod2_model*)*, 29
SpatialPoints, *103*
SpatialPointsDataFrame, *103*, *104*
SpatRaster, *3*, *4*, *9*, *10*, *12–14*, *31–33*, *43*, *44*, *46*, *59–61*, *69*, *70*, *77*, *97*, *104*, *108*, *115–118*
SpatVector, *8*, *9*, *12*, *13*, *42–45*, *77*, *117*
SRE_biomod2_model-class *(*biomod2_model*)*, 29
summary,BIOMOD.formated.data-method, 142

train, *20*, *120*, *122*
trainControl, *121*, *122*
tune, *122*

xgboost, *54*, *56*, *114*, *136*
XGBOOST_biomod2_model-class *(*biomod2_model*)*, 29