Package ‘SDMtune’

January 20, 2025

Type Package

Title Species Distribution Model Selection

Version 1.3.2

Description User-friendly framework that enables the training and the evaluation of species distribution models (SDMs). The package implements functions for data driven variable selection and model tuning and includes numerous utilities to display the results. All the functions used to select variables or to tune model hyperparameters have an interactive real-time chart displayed in the 'RStudio' viewer pane during their execution.

License GPL-3

URL https://consbiol-unibern.github.io/SDMtune/

BugReports https://github.com/ConsBiol-unibern/SDMtune/issues

SystemRequirements Java (>= 8)

Depends R (>= 4.1.0)

Imports cli (>= 3.4.1), dismo (>= 1.3-9), gbm (>= 2.1.5), ggplot2 (>= 3.4.0), jsonlite (>= 1.6), maxnet (>= 0.1.4), methods, nnet (>= 7.3-12), randomForest (>= 4.6-14), Rcpp (>= 1.0.1), rlang (>= 0.4.5), rstudioapi (>= 0.10), stringr (>= 1.4.0), terra (>= 1.6-47), whisker (>= 0.3-2)

Encoding UTF-8

LazyData true

LinkingTo Rcpp

RoxygenNote 7.3.2

Suggests covr, htmltools (>= 0.3.6), kableExtra (>= 1.1.0), knitr (>= 1.23), maps (>= 3.3.0), pkgdown (>= 2.0.7), plotROC (>= 2.2.1), rasterVis (>= 0.50), reshape2 (>= 1.4.3), rJava (>= 0.9-11), rmarkdown (>= 2.16), scales (>= 1.0.0), testthat (>= 3.1.7), withr (>= 2.5.0), zeallot (>= 0.1.0)

VignetteBuilder knitr

Config/testthat/edition 3

Collate 'ANN-class.R' 'BRT-class.R' 'Maxent-class.R' 'Maxnet-class.R' 'RF-class.R' 'RcppExports.R' 'SWD-class.R' 'SDMmodel-class.R' 'SDMmodel2MaxEnt.R' 'SDMmodelCV-class.R' 'SDMtune-class.R' 'SDMtune-package.R' 'addSamplesToBg.R' 'aicc.R' 'auc.R' 'chart-utils.R' 'checkMaxentInstallation.R' 'combineCV.R' 'confMatrix.R' 'convertFolds.R' 'corVar.R' 'doJk.R' 'getTunableArgs.R' 'gridSearch.R' 'maxentTh.R' 'maxentVarImp.R' 'mergeSWD.R' 'modelReport.R' 'optimizeModel.R' 'plotCor.R' 'plotJk.R' 'plotPA.R' 'plotPred.R' 'plotROC.R' 'plotResponse.R' 'plotVarImp.R' 'predict-ANN.R' 'predict-BRT.R' 'predict-Maxent.R' 'predict-Maxnet.R' 'predict-RF.R' 'predict-SDMmodel.R' 'predict-SDMmodelCV.R' 'prepareSWD.R' 'randomFolds.R' 'randomSearch.R' 'reduceVar.R' 'swd2csv.R' 'thinData.R' 'thresholds.R' 'train.R' 'trainANN.R' 'trainBRT.R' 'trainMaxent.R' 'trainMaxnet.R' 'trainRF.R' 'trainValTest.R' 'tss.R' 'utils.R' 'varImp.R' 'varSel.R' 'virtualSp.R' 'zzz.R'

Language en-US

NeedsCompilation yes

Author Sergio Vignali [aut, cre] (<https://orcid.org/0000-0002-3390-5442>), Arnaud Barras [aut] (<https://orcid.org/0000-0003-0850-6965>), Veronika Braunisch [aut] (<https://orcid.org/0000-0001-7035-4662>), Conservation Biology - University of Bern [fnd]

Maintainer Sergio Vignali <sdmtune@gmail.com>

Repository CRAN

Date/Publication 2024-12-16 16:50:06 UTC

Contents

`addSamplesToBg` *Add Samples to Background*

Description

The function add the presence locations to the background. This is equivalent to the Maxent argument `addsamplestobackground=true`.

Usage

```
addSamplesToBg(x, all = FALSE)
```

Arguments

x	SWD object.
all	logical. If TRUE it adds all the presence locations even if already included in the background locations. This is equivalent to the Maxent argument addallsamplestobackground=true.

Value

An object of class SWD.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")
data

# Add presence locations with values not included in the backgrounds to the
# background locations
new_data <- addSamplesToBg(data)
new_data

# Add all the presence locations to the background locations, even if they
# have values already included in the backgrounds
new_data <- addSamplesToBg(data,
                           all = TRUE)
new_data
```

aicc	*AICc*

Description

Compute the Akaike Information Criterion corrected for small samples size (Warren and Seifert, 2011).

Usage

```
aicc(model, env)
```

Arguments

`model`	SDMmodel object.
`env`	rast containing the environmental variables.

Details

The function is available only for **Maxent** and **Maxnet** methods.

Value

The computed AICc

Author(s)

Sergio Vignali

References

Warren D.L., Seifert S.N., (2011). Ecological niche modeling in Maxent: the importance of model complexity and the performance of model selection criteria. Ecological Applications, 21(2), 335–342.

See Also

auc and tss.

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                   full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
```

```
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Train a model
model <- train(method = "Maxnet",
               data = data,
               fc = "l")

# Compute the AICc
aicc(model,
     env = predictors)
```

ANN-class *Artificial Neural Network*

Description

This Class represents an Artificial Neural Network model object and hosts all the information related to the model.

Usage

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

Arguments

object	ANN object

Details

See nnet for the meaning of the slots.

Slots

- `size` integer. Number of the units in the hidden layer.
- `decay` numeric. Weight decay.
- `rang` numeric. Initial random weights.
- `maxit` integer. Maximum number of iterations.
- `model` nnet. The nnet model object.

Author(s)

Sergio Vignali

---

auc *AUC*

Description

Compute the AUC using the Man-Whitney U Test formula.

Usage

```
auc(model, test = NULL)
```

Arguments

`model`	An SDMmodel or SDMmodelCV object.
`test`	SWD object when `model` is an SDMmodel object; logical or SWD object when `model` is an SDMmodelCV object. If not provided it computes the training AUC, see details.

Details

For SDMmodelCV objects, the function computes the mean of the training AUC values of the k-folds. If `test = TRUE` it computes the mean of the testing AUC values for the k-folds. If test is an SWD object, it computes the mean AUC values for the provided testing dataset.

Value

The value of the AUC.

Author(s)

Sergio Vignali

References

Mason, S. J. and Graham, N. E. (2002), Areas beneath the relative operating characteristics (ROC) and relative operating levels (ROL) curves: Statistical significance and interpretation. Q.J.R. Meteorol. Soc., 128: 2145-2166.

See Also

aicc and tss.

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Split presence locations in training (80%) and testing (20%) datasets
datasets <- trainValTest(data,
                         test = 0.2,
                         only_presence = TRUE)
train <- datasets[[1]]
test <- datasets[[2]]

# Train a model
model <- train(method = "Maxnet",
               data = train,
               fc = "l")

# Compute the training AUC
auc(model)

# Compute the testing AUC
auc(model,
    test = test)

# Same example but using cross validation instead of training and testing
# datasets
folds <- randomFolds(data,
                     k = 4,
                     only_presence = TRUE)

model <- train(method = "Maxnet",
               data = data,
               fc = "l",
               folds = folds)

# Compute the training AUC
auc(model)
```

```
# Compute the testing AUC
auc(model,
    test = TRUE)

# Compute the AUC for the held apart testing dataset
auc(model,
    test = test)
```

BRT-class *Boosted Regression Tree*

Description

This Class represents a Boosted Regression Tree model objects and hosts all the information related to the model.

Usage

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

Arguments

`object`	BRT object

Details

See gbm for the meaning of the slots.

Slots

`distribution` character. Name of the used distribution.

`n.trees` integer. Maximum number of grown trees.

`interaction.depth` integer. Maximum depth of each tree.

`shrinkage` numeric. The shrinkage parameter.

`bag.fraction` numeric. Random fraction of data used in the tree expansion.

`model` gbm. The Boosted Regression Tree model object.

Author(s)

Sergio Vignali

checkMaxentInstallation

Check Maxent Installation

Description

The function checks if Maxent is correctly installed.

Usage

```
checkMaxentInstallation(verbose = TRUE)
```

Arguments

verbose	logical, if TRUE the function provides useful messages to understand what is not correctly installed.

Details

In order to have Maxent correctly configured is necessary that:

- Java is installed;
- the package "rJava" is installed;
- the file "maxent.jar" is in the correct folder.

Value

TRUE if Maxent is correctly installed, FALSE otherwise.

Author(s)

Sergio Vignali

Examples

```
checkMaxentInstallation()
```

combineCV *Combine Cross Validation models*

Description

This function combines cross-validation models by retraining a new model with all presence and absence/background locations and the same hyperparameters.

Usage

```
combineCV(model)
```

Arguments

`model`	SDMmodelCV object.

Details

This is an utility function to retrain a model with all data after, for example, the hyperparameters tuning (gridSearch, randomSearch or optimizeModel) to avoid manual setting of the hyperparameters in the train function.

Value

An SDMmodel object.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")
```

```
# Create 4 random folds splitting only the presence data
folds <- randomFolds(data,
                     k = 4,
                     only_presence = TRUE)

model <- train(method = "Maxnet",
               data = data,
               folds = folds)

# Define the hyperparameters to test
h <- list(reg = 1:2,
          fc = c("lqp", "lqph"))

# Run the function using the AUC as metric
output <- gridSearch(model,
                     hypers = h,
                     metric = "auc")
output@results
output@models

# Order results by highest test AUC
output@results[order(-output@results$test_AUC), ]

# Combine cross validation models for output with highest test AUC
idx <- which.max(output@results$test_AUC)
combined_model <- combineCV(output@models[[idx]])
combined_model
```

`confMatrix` *Confusion Matrix*

Description

Computes Confusion Matrixes for threshold values varying from 0 to 1.

Usage

```
confMatrix(model, test = NULL, th = NULL, type = NULL)
```

Arguments

`model`	SDMmodel object.
`test`	SWD testing locations, if not provided it uses the training dataset.
`th`	numeric vector. If provided it computes the evaluation at the given thresholds. Default is `NULL` and it computes the evaluation for the unique predicted values at presence and absence/background locations.
`type`	character. The output type used for "Maxent" and "Maxnet" methods, possible values are "cloglog" and "logistic".

Details

- For models trained with the **Maxent** method the argument `type` can be: "raw", "logistic" and "cloglog".
- For models trained with the **Maxnet** method the argument `type` can be: "link", "exponential", "logistic" and "cloglog", see maxnet for more details.

Value

The Confusion Matrix for all the used thresholds.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Train a model
model <- train(method = "Maxnet",
               data = data,
               fc = "l")

# Get the confusion matrix for thresholds ranging from 0 to 1
cm <- confMatrix(model,
                 type = "cloglog")
head(cm)
tail(cm)

# Get the confusion matrix for a specific threshold
confMatrix(model,
           type = "logistic",
           th = 0.6)
```

corVar *Print Correlated Variables*

Description

Utility that prints the name of correlated variables and the relative correlation coefficient value.

Usage

```
corVar(
  bg,
  method = "spearman",
  cor_th = NULL,
  order = TRUE,
  remove_diagonal = TRUE
)
```

Arguments

bg	SWD object with the locations used to compute the correlation between environmental variables.
method	character. The method used to compute the correlation matrix.
cor_th	numeric. If provided it prints only the variables whose correlation coefficient is higher or lower than the given threshold.
order	logical. If TRUE the variable are ordered from the most to the less highly correlated.
remove_diagonal	logical. If TRUE the values in the diagonal are removed.

Value

A data.frame with the variables and their correlation.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare background locations
```

```
bg_coords <- terra::spatSample(predictors,
                               size = 10000,
                               method = "random",
                               na.rm = TRUE,
                               xy = TRUE,
                               values = FALSE)

# Create SWD object
bg <- prepareSWD(species = "Virtual species",
                 a = bg_coords,
                 env = predictors,
                 categorical = "biome")

# Get the correlation among all the environmental variables
corVar(bg,
       method = "spearman")

# Get the environmental variables that have a correlation greater or equal to
# the given threshold
corVar(bg,
       method = "pearson",
       cor_th = 0.8)
```

---

doJk *Jackknife Test*

Description

Run the Jackknife test for variable importance removing one variable at time.

Usage

```
doJk(
  model,
  metric,
  variables = NULL,
  test = NULL,
  with_only = TRUE,
  env = NULL,
  return_models = FALSE,
  progress = TRUE
)
```

Arguments

`model`	SDMmodel or SDMmodelCV object.
`metric`	character. The metric used to evaluate the models, possible values are: "auc", "tss" and "aicc".

`variables`	vector. Variables used for the test, if not provided it takes all the variables used to train the model.
`test`	SWD. If provided it reports the result also for the testing dataset. Not used for **aicc** and SDMmodelCV.
`with_only`	logical. If `TRUE` it runs the test also for each variable in isolation.
`env`	rast containing the environmental variables, used only with "aicc".
`return_models`	logical. If `TRUE` returns all the models together with the test result.
`progress`	logical If `TRUE` shows a progress bar.

Value

A data frame with the test results. If `return_model = TRUE` it returns a list containing the test results together with the models.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Split presence locations in training (80%) and testing (20%) datasets
datasets <- trainValTest(data,
                         test = 0.2,
                         only_presence = TRUE)
train <- datasets[[1]]
test <- datasets[[2]]

# Train a model
model <- train(method = "Maxnet",
               data = train,
               fc = "lq")
```

```
# Execute the Jackknife test only for the environmental variables "bio1" and
# "bio12", using the metric AUC
doJk(model,
     metric = "auc",
    variables = c("bio1", "bio12"),
    test = test)

# The same without testing dataset
doJk(model,
     metric = "auc",
     variables = c("bio1", "bio12"))

# Execute the Jackknife test only for the environmental variables "bio1" and
# "bio12", using the metric TSS but without running the test for one single
# variable
doJk(model,
     metric = "tss",
     variables = c("bio1", "bio12"),
     test = test,
     with_only = FALSE)

# Execute the Jackknife test only for the environmental variables "bio1" and
# "bio12", using the metric AICc but without running the test for one single
# variable
doJk(model,
     metric = "aicc",
     variables = c("bio1", "bio12"),
     with_only = FALSE,
     env = predictors)

# Execute the Jackknife test for all the environmental variables using the
# metric AUC and returning all the trained models
jk <- doJk(model,
           metric = "auc",
           test = test,
           return_models = TRUE)

jk$results
jk$models_without
jk$models_withonly
```

`getTunableArgs` *Get Tunable Arguments*

Description

Returns the name of all function arguments that can be tuned for a given model.

Usage

```
getTunableArgs(model)
```

Arguments

`model`	SDMmodel or SDMmodelCV object.

Value

character vector.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Train a Maxnet model and get tunable hyperparameters
model <- train(method = "Maxnet",
               data = data,
               fc = "l")

getTunableArgs(model)
```

`gridSearch` *Grid Search*

Description

Given a set of possible hyperparameter values, the function trains models with all the possible combinations of hyperparameters.

Usage

```
gridSearch(
  model,
  hypers,
  metric,
  test = NULL,
  env = NULL,
  save_models = TRUE,
  interactive = TRUE,
  progress = TRUE
)
```

Arguments

`model`	SDMmodel or SDMmodelCV object.
`hypers`	named list containing the values of the hyperparameters that should be tuned, see details.
`metric`	character. The metric used to evaluate the models, possible values are: "auc", "tss" and "aicc".
`test`	SWD object. Testing dataset used to evaluate the model, not used with aicc and SDMmodelCV objects.
`env`	rast containing the environmental variables, used only with "aicc".
`save_models`	logical. If `FALSE` the models are not saved and the output contains only a data frame with the metric values for each hyperparameter combination. Set it to `FALSE` when there are many combinations to avoid R crashing for memory overload.
`interactive`	logical. If `FALSE` the interactive chart is not created.
`progress`	logical. If `TRUE` shows a progress bar.

Details

To know which hyperparameters can be tuned you can use the output of the function getTunableArgs. Hyperparameters not included in the `hypers` argument take the value that they have in the passed model.

An interactive chart showing in real-time the steps performed by the algorithm is displayed in the Viewer pane.

Value

SDMtune object.

Author(s)

Sergio Vignali

See Also

randomSearch and optimizeModel.

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Split presence locations in training (80%) and testing (20%) datasets
datasets <- trainValTest(data,
                         test = 0.2,
                         only_presence = TRUE)
train <- datasets[[1]]
test <- datasets[[2]]

# Train a model
model <- train(method = "Maxnet",
               data = train,
               fc = "l")

# Define the hyperparameters to test
h <- list(reg = 1:2,
          fc = c("lqp", "lqph"))

# Run the function using the AUC as metric
output <- gridSearch(model,
                     hypers = h,
                     metric = "auc",
                     test = test)
output@results
output@models

# Order results by highest test AUC
output@results[order(-output@results$test_AUC), ]

# Run the function using the AICc as metric and without saving the trained
# models, helpful when numerous hyperparameters are tested to avoid memory
```

```
# problems
output <- gridSearch(model,
                     hypers = h,
                     metric = "aicc",
                     env = predictors,
                     save_models = FALSE)
output@results
```

`Maxent-class` *Maxent*

Description

This Class represents a MaxEnt model objects and hosts all the information related to the model.

Usage

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

Arguments

`object`	Maxent object

Slots

`results` matrix. The result that usually MaxEnt provide as a csv file.

`reg` numeric. The value of the regularization multiplier used to train the model.

`fc` character. The feature class combination used to train the model.

`iter` numeric. The number of iterations used to train the model.

`extra_args` character. Extra arguments used to run MaxEnt.

`lambdas` vector. The lambdas parameters of the model.

`coeff` data.frame. The lambda coefficients of the model.

`formula` formula. The formula used to make prediction.

`lpn` numeric. Linear Predictor Normalizer.

`dn` numeric. Density Normalizer.

`entropy` numeric. The entropy value.

`min_max` data.frame. The minimum and maximum values of the continuous variables, used for clamping.

Author(s)

Sergio Vignali

maxentTh *MaxEnt Thresholds*

Description

Returns the value of the thresholds generated by the MaxEnt software.

Usage

```
maxentTh(model)
```

Arguments

model	SDMmodel object trained using the "Maxent" method.

Value

data.frame with the thresholds.

Author(s)

Sergio Vignali

See Also

maxentVarImp.

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Train a Maxent model
model <- train(method = "Maxent",
               data = data,
```

```
               fc = "l")

maxentTh(model)
```

---

maxentVarImp *Maxent Variable Importance*

Description

Shows the percent contribution and permutation importance of the environmental variables used to train the model.

Usage

```
maxentVarImp(model)
```

Arguments

model	SDMmodel or SDMmodelCV object trained using the "Maxent" method.

Details

When an SDMmodelCV object is passed to the function, the output is the average of the variable importance of each model trained during the cross validation.

Value

A data frame with the variable importance.

Author(s)

Sergio Vignali

See Also

maxentTh.

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background
```

```
# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Train a Maxent model
# The next line checks if Maxent is correctly configured but you don't need
# to run it in your script
model <- train(method = "Maxent",
               data = data,
               fc = "l")

maxentVarImp(model)
```

Maxnet-class *Maxnet*

Description

This Class represents a Maxnet model objects and hosts all the information related to the model.

Usage

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

Arguments

object	Maxnet object

Slots

`reg` numeric. The value of the regularization multiplier used to train the model.

`fc` character. The feature class combination used to train the model.

`model` maxnet. The maxnet model object.

Author(s)

Sergio Vignali

mergeSWD — *Merge SWD Objects*

Description

Merge two SWD objects.

Usage

```
mergeSWD(swd1, swd2, only_presence = FALSE)
```

Arguments

swd1	SWD object.
swd2	SWD object.
only_presence	logical If TRUE only for the presence locations are merged and the absence/background locations are taken only from the swd1 object.

Details

- In case the two SWD objects have different columns, only the common columns are used in the merged object.
- The SWD object is created in a way that the presence locations are always before than the absence/background locations.

Value

The merged SWD object.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
```

```
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Split only presence locations in training (80%) and testing (20%) datasets
datasets <- trainValTest(data,
                         test = 0.2,
                         only_presence = TRUE)
train <- datasets[[1]]
test <- datasets[[2]]

# Merge the training and the testing datasets together
merged <- mergeSWD(train,
                   test,
                   only_presence = TRUE)

# Split presence and absence locations in training (80%) and testing (20%)
datasets
datasets <- trainValTest(data,
                         test = 0.2)
train <- datasets[[1]]
test <- datasets[[2]]

# Merge the training and the testing datasets together
merged <- mergeSWD(train, test)
```

`modelReport` *Model Report*

Description

Make a report that shows the main results.

Usage

```
modelReport(
  model,
  folder,
  test = NULL,
  type = NULL,
  response_curves = FALSE,
  only_presence = FALSE,
  jk = FALSE,
  env = NULL,
  clamp = TRUE,
  permut = 10,
  verbose = TRUE
)
```

Arguments

model	SDMmodel object.
folder	character. The name of the folder in which to save the output. The folder is created in the working directory.
test	SWD object with the test locations.
type	character. The output type used for "Maxent" and "Maxnet" methods, possible values are "cloglog" and "logistic".
response_curves	logical, if TRUE it plots the response curves in the html output.
only_presence	logical, if TRUE it uses only the range of the presence location for the marginal response.
jk	logical, if TRUE it runs the jackknife test.
env	rast. If provided it computes and adds a prediction map to the output.
clamp	logical for clumping during prediction, used for response curves and for the prediction map.
permut	integer. Number of permutations.
verbose	logical, if TRUE prints informative messages.

Details

The function produces a report similar to the one created by MaxEnt software. See **terra** documentation to see how to pass factors.

Author(s)

Sergio Vignali

Examples

```
# If you run the following examples with the function example(),
# you may want to set the argument ask like following: example("modelReport",
# ask = FALSE)
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
```

```
                   env = predictors,
                   categorical = "biome")

# Split presence locations in training (80%) and testing (20%) datasets
datasets <- trainValTest(data,
                         test = 0.2,
                         only_presence = TRUE)
train <- datasets[[1]]
test <- datasets[[2]]

# Train a model
model <- train(method = "Maxnet",
               data = train,
               fc = "lq")

# Create the report
## Not run:
modelReport(model,
            type = "cloglog",
            folder = "my_folder",
            test = test,
            response_curves = TRUE,
            only_presence = TRUE,
            jk = TRUE,
            env = predictors,
            permut = 2)
## End(Not run)
```

`optimizeModel` *Optimize Model*

Description

The function uses a Genetic Algorithm implementation to optimize the model hyperparameter configuration according to the chosen metric.

Usage

```
optimizeModel(
  model,
  hypers,
  metric,
  test = NULL,
  pop = 20,
  gen = 5,
  env = NULL,
  keep_best = 0.4,
  keep_random = 0.2,
  mutation_chance = 0.4,
```

```
  interactive = TRUE,
  progress = TRUE,
  seed = NULL
)
```

Arguments

model	SDMmodel or SDMmodelCV object.
hypers	named list containing the values of the hyperparameters that should be tuned, see details.
metric	character. The metric used to evaluate the models, possible values are: "auc", "tss" and "aicc".
test	SWD object. Testing dataset used to evaluate the model, not used with aicc and SDMmodelCV objects.
pop	numeric. Size of the population.
gen	numeric. Number of generations.
env	rast containing the environmental variables, used only with "aicc".
keep_best	numeric. Percentage of the best models in the population to be retained during each iteration, expressed as decimal number.
keep_random	numeric. Probability of retaining the excluded models during each iteration, expressed as decimal number.
mutation_chance	numeric. Probability of mutation of the child models, expressed as decimal number.
interactive	logical. If `FALSE` the interactive chart is not created.
progress	logical. If `TRUE` shows a progress bar.
seed	numeric. The value used to set the seed to have consistent results.

Details

To know which hyperparameters can be tuned you can use the output of the function getTunableArgs. Hyperparameters not included in the `hypers` argument take the value that they have in the passed model.

An interactive chart showing in real-time the steps performed by the algorithm is displayed in the Viewer pane.

Part of the code is inspired by this post.

Value

SDMtune object.

Author(s)

Sergio Vignali

See Also

gridSearch and randomSearch.

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Split presence locations in training (80%) and testing (20%) datasets
datasets <- trainValTest(data,
                         val = 0.2,
                         test = 0.2,
                         only_presence = TRUE,
                         seed = 61516)
train <- datasets[[1]]
val <- datasets[[2]]

# Train a model
model <- train("Maxnet",
               data = train)

# Define the hyperparameters to test
h <- list(reg = seq(0.2, 5, 0.2),
          fc = c("l", "lq", "lh", "lp", "lqp", "lqph"))

# Run the function using as metric the AUC
## Not run:
output <- optimizeModel(model,
                        hypers = h,
                        metric = "auc",
                        test = val,
                        pop = 15,
                        gen = 2,
                        seed = 798)
output@results
output@models
output@models[[1]]  # Best model
```

```
## End(Not run)
```

plotCor *Plot Correlation*

Description

Plot a correlation matrix heat map with the value of the correlation coefficients according with the given method. If cor_th is passed then it prints only the coefficients that are higher or lower than the given threshold.

Usage

```
plotCor(bg, method = "spearman", cor_th = NULL, text_size = 3)
```

Arguments

bg	SWD object used to compute the correlation matrix.
method	character. The method used to compute the correlation matrix.
cor_th	numeric. If provided it prints only the coefficients that are higher or lower than the given threshold.
text_size	numeric, used to change the size of the text.

Value

A ggplot object.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare background locations
bg_coords <- terra::spatSample(predictors,
                               size = 9000,
                               method = "random",
                               na.rm = TRUE,
                               xy = TRUE,
                               values = FALSE)
```

```
# Create SWD object
bg <- prepareSWD(species = "Virtual species",
                 a = bg_coords,
                 env = predictors,
                 categorical = "biome")

# Plot heat map
plotCor(bg,
        method = "spearman")

# Plot heat map showing only values higher than given threshold and change
# text size
plotCor(bg,
        method = "spearman",
        cor_th = 0.8,
        text_size = 4)
```

---

plotJk *Plot Jackknife Test*

Description

Plot the Jackknife Test for variable importance.

Usage

```
plotJk(jk, type = c("train", "test"), ref = NULL)
```

Arguments

jk	data.frame with the output of the doJk function.
type	character, "train" or "test" to plot the result of the test on the train or testing dataset.
ref	numeric. The value of the chosen metric for the model trained using all the variables. If provided it plots a vertical line showing the reference value.

Value

A ggplot object.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Split presence locations in training (80%) and testing (20%) datasets
datasets <- trainValTest(data,
                         test = 0.2,
                         only_presence = TRUE)
train <- datasets[[1]]
test <- datasets[[2]]

# Train a model
model <- train(method = "Maxnet",
               data = train,
               fc = "lq")

# Execute the Jackknife test for all the environmental variables using the
# metric AUC
jk <- doJk(model,
           metric = "auc",
           test = test)

# Plot Jackknife test result for training
plotJk(jk,
       type = "train",
       ref = auc(model))

#' # Plot Jackknife test result for testing
plotJk(jk,
       type = "test",
       ref = auc(model, test = test))
```

`plotPA` *Plot Presence Absence Map*

Description

Plot a presence absence map using the given threshold.

Usage

```
plotPA(
  map,
  th,
  colors = NULL,
  hr = FALSE,
  filename = "",
  overwrite = FALSE,
  wopt = list(),
  ...
)
```

Arguments

`map`	rast object with the prediction.
`th`	numeric. The threshold used to convert the output in a presence/absence map.
`colors`	vector. Colors to be used, default is `NULL` and it uses red and blue.
`hr`	logical. If `TRUE` it produces an output with high resolution.
`filename`	character. If provided the raster map is saved in a file. It must include the extension.
`overwrite`	logical. If `TRUE` an existing file is overwritten.
`wopt`	list. Writing options passed to writeRaster.
`...`	Unused arguments.

Value

A ggplot object.

Author(s)

Sergio Vignali

See Also

plotPred.

Examples

```
map <- terra::rast(matrix(runif(400, 0, 1),
                                nrow = 20,
                                ncol = 20))
plotPA(map,
       th = 0.8)
```

```
# Custom colors
plotPA(map,
       th = 0.5,
       colors = c("#d8b365", "#018571"))

## Not run:
# Save the file. The following command will save the map in the working
# directory. Note that `filename` must include the extension.
plotPA(map,
       th = 0.7,
       filename = "my_map.tif")
## End(Not run)
```

---

plotPred *Plot Prediction*

Description

Plot Prediction output.

Usage

```
plotPred(map, lt = "", colorramp = NULL, hr = FALSE)
```

Arguments

map	rast object with the prediction.
lt	character. Legend title.
colorramp	vector. A custom colour ramp given as a vector of colours (see example), default is NULL and uses a blue/red colour ramp.
hr	logical. If TRUE produces an output with high resolution.

Value

A ggplot object.

Author(s)

Sergio Vignali

See Also

plotPA.

Examples

```
map <- terra::rast(matrix(runif(400, 0, 1),
                                    nrow = 20,
                                    ncol= 20))

plotPred(map,
         lt = "Habitat suitability \ncloglog")

# Custom colors
plotPred(map,
         lt = "Habitat suitability",
         colorramp = c("#2c7bb6", "#ffffbf", "#d7191c"))
```

`plotResponse` *Plot Response Curve*

Description

Plot the Response Curve of the given environmental variable.

Usage

```
plotResponse(
  model,
  var,
  type = NULL,
  only_presence = FALSE,
  marginal = FALSE,
  fun = mean,
  rug = FALSE,
  color = "red"
)
```

Arguments

`model`	SDMmodel or SDMmodelCV object.
`var`	character. Name of the variable to be plotted.
`type`	character. The output type used for "Maxent" and "Maxnet" methods, possible values are "cloglog" and "logistic".
`only_presence`	logical. If `TRUE` it uses only the presence locations when applying the function for the marginal response.
`marginal`	logical. If `TRUE` it plots the marginal response curve.
`fun`	function used to compute the level of the other variables for marginal curves.
`rug`	logical. If `TRUE` it adds the rug plot for the presence and absence/background locations, available only for continuous variables.
`color`	The color of the curve, default is "red".

Details

- Note that `fun` is not a character argument, you must use `mean` and not `"mean"`.
- If you want to modify the plot, first you have to assign the output of the function to a variable, and then you have two options:
  - Modify the `ggplot` object by editing the theme or adding additional elements
  - Get the data with `ggplot2::ggplot_build()` and then build your own plot (see examples)

Value

A ggplot object.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Train a model
model <- train(method = "Maxnet",
               data = data,
               fc = "lq")

# Plot cloglog response curve for a continuous environmental variable (bio1)
plotResponse(model,
             var = "bio1",
             type = "cloglog")

# Plot marginal cloglog response curve for a continuous environmental
# variable (bio1)
plotResponse(model,
             var = "bio1",
             type = "cloglog",
```

```
             marginal = TRUE)

# Plot logistic response curve for a continuous environmental variable
# (bio12) adding the rugs and giving a custom color
plotResponse(model,
             var = "bio12",
             type = "logistic",
             rug = TRUE,
             color = "blue")

# Plot response curve for a categorical environmental variable (biome) giving
# a custom color
plotResponse(model,
             var = "biome",
             type = "logistic",
             color = "green")

# Modify plot
# Change y axes limits
my_plot <- plotResponse(model,
                        var = "bio1",
                        type = "cloglog")
my_plot +
  ggplot2::scale_y_continuous(limits = c(0, 1))

# Get the data and create your own plot:
df <- ggplot2::ggplot_build(my_plot)$data[[1]]
plot(df$x, df$y,
     type = "l",
     lwd = 3,
     col = "blue",
     xlab = "bio1",
     ylab = "cloglog output")

# Train a model with cross validation
folds <- randomFolds(data,
                     k = 4,
                     only_presence = TRUE)

model <- train(method = "Maxnet",
               data = data,
               fc = "lq",
               folds = folds)

# Plot cloglog response curve for a continuous environmental variable (bio17)
plotResponse(model,
             var = "bio1",
             type = "cloglog")

# Plot logistic response curve for a categorical environmental variable
# (biome) giving a custom color
plotResponse(model,
             var = "biome",
```

```
              type = "logistic",
              color = "green")
```

plotROC *Plot ROC curve*

Description

Plot the ROC curve of the given model and print the AUC value.

Usage

```
plotROC(model, test = NULL)
```

Arguments

`model`	SDMmodel object.
`test`	SWD object. The testing dataset.

Value

A ggplot object.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Split presence locations in training (80%) and testing (20%) datasets
datasets <- trainValTest(data,
```

```
                         test = 0.2,
                         only_presence = TRUE)
train <- datasets[[1]]
test <- datasets[[2]]

# Train a model
model <- train(method = "Maxnet",
               data = train,
               fc = "l")

# Plot the training ROC curve
plotROC(model)

# Plot the training and testing  ROC curves
plotROC(model,
        test = test)
```

`plotVarImp` *Plot Variable Importance*

Description

Plot the variable importance as a bar plot.

Usage

```
plotVarImp(df, color = "grey")
```

Arguments

`df`	data.frame. A data.frame containing the the name of the variables as first column and the value of the variable importance as second column.
`color`	character. The colour of the bar plot.

Value

A ggplot object.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)
```

```
predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Train a model
model <- train(method = "Maxnet",
               data = data,
               fc = "l")

# Compute variable importance
vi <- varImp(model,
             permut = 1)

# Plot variable importance
plotVarImp(vi)

# Plot variable importance with custom color
plotVarImp(vi,
           color = "red")
```

`predict,ANN-method` *Predict ANN*

Description

Predict the output for a new dataset from a trained ANN model.

Usage

```
## S4 method for signature 'ANN'
predict(object, data, type, clamp)
```

Arguments

`object`	ANN object.
`data`	data.frame with the data for the prediction.
`type`	Not used.
`clamp`	Not used.

Details

Used by the predict,SDMmodel-method, not exported.

Value

A vector with the predicted values.

Author(s)

Sergio Vignali

---

predict,BRT-method *Predict BRT*

Description

Predict the output for a new dataset from a trained BRT model.

Usage

```
## S4 method for signature 'BRT'
predict(object, data, type, clamp)
```

Arguments

object	BRT object.
data	data.frame with the data for the prediction.
type	Not used.
clamp	Not used.

Details

Used by the predict,SDMmodel-method, not exported.

The function uses the number of tree defined to train the model and the "response" type output.

Value

A vector with the predicted values.

Author(s)

Sergio Vignali

predict,Maxent-method *Predict Maxent*

Description

Predict the output for a new dataset from a trained Maxent model.

Usage

```
## S4 method for signature 'Maxent'
predict(object, data, type = c("cloglog", "logistic", "raw"), clamp = TRUE)
```

Arguments

`object`	Maxent object.
`data`	data.frame with the data for the prediction.
`type`	character. MaxEnt output type, possible values are "cloglog", "logistic" and "raw".
`clamp`	logical for clumping during prediction.

Details

Used by the predict,SDMmodel-method, not exported.

The function performs the prediction in **R** without calling the **MaxEnt** Java software. This results in a faster computation for large datasets and might result in a slightly different output compared to the Java software.

Value

A vector with the prediction

Author(s)

Sergio Vignali

References

Wilson P.D., (2009). Guidelines for computing MaxEnt model output values from a lambdas file.

predict,Maxnet-method *Predict Maxnet*

Description

Predict the output for a new dataset from a trained Maxnet model.

Usage

```
## S4 method for signature 'Maxnet'
predict(
  object,
  data,
  type = c("link", "exponential", "cloglog", "logistic"),
  clamp = TRUE
)
```

Arguments

object	Maxnet object.
data	data.frame with the data for the prediction.
type	character. Maxnet output type, possible values are "link", "exponential", "cloglog" and "logistic".
clamp	logical for clumping during prediction.

Details

Used by the predict,SDMmodel-method, not exported.

Value

A vector with the predicted values.

Author(s)

Sergio Vignali

predict,RF-method *Predict RF*

Description

Predict the output for a new dataset from a trained RF model.

Usage

```
## S4 method for signature 'RF'
predict(object, data, type, clamp)
```

Arguments

`object`	RF object.
`data`	data.frame with the data for the prediction.
`type`	Not used.
`clamp`	Not used.

Details

Used by the predict,SDMmodel-method, not exported.

Value

A vector with the predicted probabilities of class 1.

Author(s)

Sergio Vignali

predict,SDMmodel-method *Predict*

Description

Predict the output for a new dataset given a trained SDMmodel model.

Usage

```
## S4 method for signature 'SDMmodel'
predict(
  object,
  data,
  type = NULL,
  clamp = TRUE,
  filename = "",
  overwrite = FALSE,
  wopt = list(),
  extent = NULL,
  ...
)
```

Arguments

`object`	SDMmodel object.
`data`	data.frame, SWD or rast with the data for the prediction.
`type`	character. Output type, see details, used only for **Maxent** and **Maxnet** methods.
`clamp`	logical for clumping during prediction, used only for **Maxent** and **Maxnet** methods.
`filename`	character. If provided the raster map is saved in a file. It must include the extension.
`overwrite`	logical. If `TRUE` an existing file is overwritten.
`wopt`	list. Writing options passed to writeRaster.
`extent`	ext object, if provided it restricts the prediction to the given extent.
`...`	Additional arguments to pass to the predict function.

Details

- filename, and extent are arguments used only when the prediction is run for a rast object.
- For models trained with the **Maxent** method the argument `type` can be: "raw", "logistic" and "cloglog". The function performs the prediction in **R** without calling the **MaxEnt** Java software. This results in a faster computation for large datasets and might result in a slightly different output compared to the Java software.
- For models trained with the **Maxnet** method the argument `type` can be: "link", "exponential", "logistic" and "cloglog", see maxnet for more details.
- For models trained with the **ANN** method the function uses the "raw" output type.
- For models trained with the **RF** method the output is the probability of class 1.
- For models trained with the **BRT** method the function uses the number of trees defined to train the model and the "response" output type.

Value

A vector with the prediction or a rast object if data is a raster rast.

Author(s)

Sergio Vignali

References

Wilson P.D., (2009). Guidelines for computing MaxEnt model output values from a lambdas file.

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Split presence locations in training (80%) and testing (20%) datasets
datasets <- trainValTest(data,
                         test = 0.2,
                         only_presence = TRUE)
train <- datasets[[1]]
test <- datasets[[2]]

# Train a model
model <- train(method = "Maxnet",
               data = train,
               fc = "l")

# Make cloglog prediction for the test dataset
predict(model,
        data = test,
        type = "cloglog")

# Make logistic prediction for the whole study area
predict(model,
        data = predictors,
        type = "logistic")

## Not run:
# Make logistic prediction for the whole study area and save it in a file.
# Note that the filename must include the extension. The function saves the
```

```
# file in your working directory
predict(model,
        data = predictors,
        type = "logistic",
        filename = "my_map.tif")
## End(Not run)
```

---

predict,SDMmodelCV-method
Predict for Cross Validation

Description

Predict the output for a new dataset given a trained SDMmodelCV model. The output is given as the provided function applied to the prediction of the k models.

Usage

```
## S4 method for signature 'SDMmodelCV'
predict(
  object,
  data,
  fun = "mean",
  type = NULL,
  clamp = TRUE,
  filename = "",
  overwrite = FALSE,
  wopt = list(),
  extent = NULL,
  progress = TRUE,
  ...
)
```

Arguments

`object`	SDMmodelCV object.
`data`	data.frame, SWD or raster rast with the data for the prediction.
`fun`	character. Function used to combine the output of the k models. Note that fun is a character argument, you must use `"mean"` and not `mean`. You can also pass a vector of character containing multiple function names, see details.
`type`	character. Output type, see details, used only for **Maxent** and **Maxnet** methods.
`clamp`	logical for clumping during prediction, used only for **Maxent** and **Maxnet** methods.
`filename`	character. If provided the raster map is saved in a file. It must include the extension.
`overwrite`	logical. If `TRUE` an existing file is overwritten.

`wopt`	list. Writing options passed to writeRaster.
`extent`	ext object, if provided it restricts the prediction to the given extent.
`progress`	logical. If `TRUE` shows a progress bar during prediction.
`...`	Additional arguments to pass to the predict function.

Details

- filename, and extent are arguments used only when the prediction is run for a rast object.
- When a character vector is passed to the `fun` argument, than all the given functions are applied and a named list is returned, see examples.
- When `filename` is provided and the `fun` argument contains more than one function name, the saved files are named as `filename_fun`, see example.
- For models trained with the **Maxent** method the argument `type` can be: "raw", "logistic" and "cloglog". The function performs the prediction in **R** without calling the **MaxEnt** Java software. This results in a faster computation for large datasets and might result in a slightly different output compared to the Java software.
- For models trained with the **Maxnet** method the argument `type` can be: "link", "exponential", "logistic" and "cloglog", see maxnet for more details.
- For models trained with the **ANN** method the function uses the "raw" output type.
- For models trained with the **RF** method the output is the probability of class 1.
- For models trained with the **BRT** method the function uses the number of trees defined to train the model and the "response" output type.

Value

A vector with the prediction or a rast object if data is a rast or a list in the case of multiple functions.

Author(s)

Sergio Vignali

References

Wilson P.D., (2009). Guidelines for computing MaxEnt model output values from a lambdas file.

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background
```

```
# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Create 4 random folds splitting only the presence data
folds <- randomFolds(data,
                     k = 4,
                     only_presence = TRUE)

model <- train(method = "Maxnet",
               data = data,
               fc = "l",
               folds = folds)

# Make cloglog prediction for the whole study area and get the result as
# average of the k models
predict(model,
        data = predictors,
        fun = "mean",
        type = "cloglog")

# Make cloglog prediction for the whole study area, get the average, standard
# deviation, and maximum values of the k models, and save the output in three
# files.
# The following commands save the output in the working directory. Note that
# the `filename` must include the extension
## Not run:
maps <- predict(model,
                data = predictors,
                fun = c("mean", "sd", "max"),
                type = "cloglog",
                filename = "prediction.tif")

# In this case three files are created: prediction_mean.tif,
# prediction_sd.tif and prediction_max.tif
plotPred(maps$mean)
plotPred(maps$sd)
plotPred(maps$max)

# Make logistic prediction for the whole study area, given as standard
# deviation of the k models, and save it in a file
predict(model,
        data = predictors,
        fun = "sd",
        type = "logistic",
        filename = "my_map.tif")
## End(Not run)
```

prepareSWD *Prepare an SWD object*

Description

Given the coordinates, the species' name and the environmental variables, the function creates an SWD object (sample with data).

Usage

```
prepareSWD(
  species,
  env,
  p = NULL,
  a = NULL,
  categorical = NULL,
  verbose = TRUE
)
```

Arguments

species	character. The name of the species.
env	rast containing the environmental variables used to extract the values at coordinate locations.
p	data.frame. The coordinates of the presence locations.
a	data.frame. The coordinates of the absence/background locations.
categorical	vector indicating which of the environmental variable are categorical.
verbose	logical, if TRUE prints informative messages.

Details

The SWD object is created in a way that the presence locations are always before than the absence/background locations.

Value

An SWD object.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create the SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")
data
```

---

randomFolds *Create Random Folds*

Description

Create random folds for cross validation.

Usage

```
randomFolds(data, k, only_presence = FALSE, seed = NULL)
```

Arguments

`data`	SWD object that will be used to train the model.
`k`	integer. Number of fold used to create the partition.
`only_presence`	logical, if `TRUE` the random folds are created only for the presence locations and all the background locations are included in each fold, used manly for presence-only methods.
`seed`	integer. The value used to set the seed for the fold partition.

Details

When `only_presence = FALSE`, the proportion of presence and absence is preserved.

Value

list with two matrices, the first for the training and the second for the testing dataset. Each column of one matrix represents a fold with `TRUE` for the locations included in and `FALSE` excluded from the partition.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd", full.names = TRUE)
predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

data <- prepareSWD(species = "Virtual species", p = p_coords, a = bg_coords,
                   env = predictors, categorical = "biome")

# Create 4 random folds splitting presence and absence locations
folds <- randomFolds(data, k = 4)

# Create 4 random folds splitting only the presence locations
folds <- randomFolds(data, k = 4, only_presence = TRUE)
```

---

randomSearch *Random Search*

Description

The function performs a random search in the hyperparameters space, creating a population of random models each one with a random combination of the provided hyperparameters values.

Usage

```
randomSearch(
  model,
  hypers,
  metric,
  test = NULL,
  pop = 20,
  env = NULL,
  interactive = TRUE,
  progress = TRUE,
  seed = NULL
)
```

Arguments

`model`	SDMmodel or SDMmodelCV object.
`hypers`	named list containing the values of the hyperparameters that should be tuned, see details.
`metric`	character. The metric used to evaluate the models, possible values are: "auc", "tss" and "aicc".
`test`	SWD object. Test dataset used to evaluate the model, not used with aicc and SDMmodelCV objects.
`pop`	numeric. Size of the population.
`env`	rast containing the environmental variables, used only with "aicc".
`interactive`	logical. If `FALSE` the interactive chart is not created.
`progress`	logical. If `TRUE` shows a progress bar.
`seed`	numeric. The value used to set the seed to have consistent results.

Details

To know which hyperparameters can be tuned you can use the output of the function getTunableArgs. Hyperparameters not included in the `hypers` argument take the value that they have in the passed model.

An interactive chart showing in real-time the steps performed by the algorithm is displayed in the Viewer pane.

Value

SDMtune object.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
```

```
                    env = predictors,
                    categorical = "biome")

# Split presence locations in training (80%) and testing (20%) datasets
datasets <- trainValTest(data,
                         test = 0.2,
                         only_presence = TRUE)
train <- datasets[[1]]
test <- datasets[[2]]

# Train a model
model <- train(method = "Maxnet",
               data = train,
               fc = "l")

# Define the hyperparameters to test
h <- list(reg = seq(0.2, 3, 0.2),
          fc = c("lqp", "lqph", "lh"))

# Run the function using as metric the AUC
output <- randomSearch(model,
                       hypers = h,
                       metric = "auc",
                       test = test,
                       pop = 10,
                       seed = 25)
output@results
output@models

# Order results by highest test AUC
output@results[order(-output@results$test_AUC), ]
```

`reduceVar` *Reduce Variables*

Description

Remove variables whose importance is less than the given threshold. The function removes one variable at time and after trains a new model to get the new variable contribution rank. If use_jk is TRUE the function checks if after removing the variable the model performance decreases (according to the given metric and based on the starting model). In this case the function stops removing the variable even if the contribution is lower than the given threshold.

Usage

```
reduceVar(
  model,
  th,
  metric,
  test = NULL,
```

```
  env = NULL,
  use_jk = FALSE,
  permut = 10,
  use_pc = FALSE,
  interactive = TRUE,
  verbose = TRUE
)
```

Arguments

model	SDMmodel or SDMmodelCV object.
th	numeric. The contribution threshold used to remove variables.
metric	character. The metric used to evaluate the models, possible values are: "auc", "tss" and "aicc", used only if use_jk is TRUE.
test	SWD object containing the test dataset used to evaluate the model, not used with aicc, and if use_jk = FALSE.
env	rast containing the environmental variables, used only with "aicc".
use_jk	Flag to use the Jackknife AUC test during the variable selection, if FALSE the function uses the percent variable contribution.
permut	integer. Number of permutations, used if use_pc = FALSE.
use_pc	logical. If TRUE and the model is trained using the Maxent method, the algorithm uses the percent contribution computed by Maxent software to score the variable importance.
interactive	logical. If FALSE the interactive chart is not created.
verbose	logical. If TRUE prints informative messages.

Details

An interactive chart showing in real-time the steps performed by the algorithm is displayed in the Viewer pane.

Value

The model trained using the selected variables.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)
```

```
# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Split presence locations in training (80%) and testing (20%) datasets
datasets <- trainValTest(data,
                         test = 0.2,
                         only_presence = TRUE)
train <- datasets[[1]]
test <- datasets[[2]]

# Train a Maxnet model
model <- train(method = "Maxnet",
               data = train,
               fc = "lq")

# Remove all variables with permuation importance lower than 2%
output <- reduceVar(model,
                    th = 2,
                    metric = "auc",
                    test = test,
                    permut = 1)

# Remove variables with permuation importance lower than 3% only if testing
# TSS doesn't decrease
## Not run:
output <- reduceVar(model,
                    th = 3,
                    metric = "tss",
                    test = test,
                    permut = 1,
                    use_jk = TRUE)

# Remove variables with permuation importance lower than 2% only if AICc
# doesn't increase
output <- reduceVar(model,
                    th = 2,
                    metric = "aicc",
                    permut = 1,
                    use_jk = TRUE,
                    env = predictors)

# Train a Maxent model
model <- train(method = "Maxent",
               data = train,
```

```
                 fc = "lq")

# Remove all variables with percent contribution lower than 2%
output <- reduceVar(model,
                    th = 2,
                    metric = "auc",
                    test = test,
                    use_pc = TRUE)
## End(Not run)
```

RF-class *Random Forest*

Description

This Class represents a Random Forest model objects and hosts all the information related to the model.

Usage

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

Arguments

object	RF object

Details

See randomForest for the meaning of the slots.

Slots

`mtry` integer. Number of variable randomly sampled.

`ntree` integer. Number of grown trees.

`nodesize` integer. Minimum size of terminal nodes.

`model` randomForest. The randomForest model object.

Author(s)

Sergio Vignali

`SDMmodel-class` *SDMmodel*

Description

This Class represents an SDMmodel object and hosts all the information related to the model.

Usage

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

Arguments

`object`	SDMmodel object

Slots

`data` SWD object. The data used to train the model.

`model` An object of class ANN, BRT, RF, Maxent or Maxnet.

Author(s)

Sergio Vignali

`SDMmodel2MaxEnt` *SDMmodel2MaxEnt*

Description

Converts an SDMmodel object containing a MaxEnt model into a dismo MaxEnt object.

Usage

```
SDMmodel2MaxEnt(model)
```

Arguments

`model`	SDMmodel object to be converted.

Value

The converted dismo MaxEnt object.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Train a Maxent model
model <- train(method = "Maxent",
               data = data,
               fc = "l")

dismo_model <- SDMmodel2MaxEnt(model)
dismo_model
```

`SDMmodelCV-class` *SDMmodelCV*

Description

This Class represents an SDMmodel model object with replicates and hosts all the models trained during the cross validation.

Usage

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

Arguments

`object`	SDMmodelCV object

Slots

`models` list. A list containing all the models trained during the cross validation.

`data` SWD object. Full dataset used to make the partitions.

`folds` list with two matrices, the first for the training and the second for the testing dataset. Each column of one matrix represents a fold with `TRUE` for the locations included in and `FALSE` excluded from the partition.

Author(s)

Sergio Vignali

`SDMtune-class` *SDMtune class*

Description

Class used to save the results of one of the following functions: gridSearch, randomSearch or optimizeModel.

Plot an SDMtune object. Use the interactive argument to create an interactive chart.

Usage

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

## S4 method for signature 'SDMtune,missing'
plot(x, title = "", interactive = FALSE)
```

Arguments

`object`	SDMtune object
`x`	SDMtune object.
`title`	character. The title of the plot.
`interactive`	logical, if TRUE plot an interactive chart.

Value

If `interactive` = `FALSE` the function returns a `ggplot` object otherwise it returns an SDMtuneChart object that contains the path of the temporary folder where the necessary files to create the chart are saved. In both cases the objects are returned as invisible.

Slots

`results` data.frame. Results with the evaluation of the models.

`models` list. List of SDMmodel or SDMmodelCV objects.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd", full.names = TRUE)
predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species", p = p_coords, a = bg_coords,
                   env = predictors, categorical = "biome")

# Split presence locations in training (80%) and testing (20%) datasets
datasets <- trainValTest(data, test = 0.2, only_presence = TRUE)
train <- datasets[[1]]
test <- datasets[[2]]

# Train a model
model <- train(method = "Maxnet", data = train, fc = "l")

# Define the hyperparameters to test
h <- list(reg = 1:5, fc = c("lqp", "lqph"))

# Run the gridSearch function using as metric the AUC
output <- gridSearch(model, hypers = h, metric = "auc", test = test)
output

# Plot the output
plot(output, title = "My experiment")

# Plot the interactive chart
p <- plot(output, title = "My experiment", interactive = TRUE)
# Print the temporary folder that stores the files used to create the chart
str(p)
```

`SWD-class` *Sample With Data*

Description

Object similar to the MaxEnt SWD format that hosts the species name, the coordinates of the locations and the value of the environmental variables at the location places.

Usage

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

Arguments

`object`	SWD object

Details

The object can contains presence/absence, presence/background, presence only or absence/background only data. Use the prepareSWD function to create the object.

Slots

`species` character. Name of the species.

`coords` data.frame. Coordinates of the locations.

`data` data.frame. Value of the environmental variables at location sites.

`pa` numeric. Vector with `1` for presence and `0` for absence/background locations.

Author(s)

Sergio Vignali

`swd2csv` *SWD to csv*

Description

Save an SWD object as csv file.

Usage

```
swd2csv(swd, file_name)
```

Arguments

`swd`	SWD object.
`file_name`	character. The name of the file in which to save the object, see details.

Details

- The `file_name` argument should include the extension (i.e. my_file.csv).
- If `file_name` is a single name the function saves the presence absence/background locations in a single file, adding the column **pa** with 1s for presence and 0s for absence/background locations. If `file_name` is a vector with two names, it saves the object in two files: the first name is used for the presence locations and the second for the absence/background locations.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")
## Not run:
# The following commands save the output in the working directory
# Save the SWD object as a single csv file
swd2csv(data,
        file_name = "train_data.csv")

# Save the SWD object in two separate csv files
swd2csv(data,
        file_name = c("presence.csv", "absence.csv"))
## End(Not run)
```

---

thinData *Thin Data*

Description

Remove all but one location per raster cell. The function removes NAs and if more than one location falls within the same raster cell it selects randomly one.

Usage

```
thinData(coords, env, x = "x", y = "y", verbose = TRUE, progress = TRUE)
```

Arguments

`coords`	data.frame or matrix with the coordinates, see details.
`env`	rast containing the environmental variables.
`x`	character. Name of the column containing the x coordinates.
`y`	character. Name of the column containing the y coordinates.
`verbose`	logical, if `TRUE` prints an informative message.
`progress`	logical, if `TRUE` shows a progress bar.

Details

- **coords** and **env** must have the same coordinate reference system.
- The coords argument can contain several columns. This is useful if the user has information related to the coordinates that doesn't want to loose with the thinning procedure. The function expects to have the x coordinates in a column named "x", and the y coordinates in a column named "y". If this is not the case, the name of the columns containing the coordinates can be specified using the arguments x and y.

Value

a matrix or a data frame with the thinned locations.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare background locations, by sampling  also on areas with NA values
bg_coords <- terra::spatSample(predictors,
                               size = 9000,
                               method = "random",
                               xy = TRUE,
                               values = FALSE)
nrow(bg_coords)

# Thin the locations
# The function will remove the coordinates that have NA values for some
# predictors. Note that the function expects to have the coordinates in two
# columns named "x" and "y"

colnames(bg_coords)
thinned_bg <- thinData(bg_coords,
                       env = predictors)
nrow(thinned_bg)

# Here we sample only on areas without NA values and then we double the
# coordinates
bg_coords <- terra::spatSample(predictors,
                               size = 9000,
                               method = "random",
                               na.rm = TRUE,
                               xy = TRUE,
                               values = FALSE)
```

```
thinned_bg <- thinData(rbind(bg_coords, bg_coords),
                       env = predictors)

nrow(thinned_bg)

# In case of a dataframe containing more than two columns (e.g. a dataframe
# with the coordinates plus an additional column with the age of the species)
# and custom column names, use the function in this way
age <- sample(c(1, 2),
              size = nrow(bg_coords),
              replace = TRUE)

data <- cbind(age, bg_coords)
colnames(data) <- c("age", "X", "Y")

thinned_bg <- thinData(data,
                       env = predictors,
                       x = "X",
                       y = "Y")
head(data)
```

---

thresholds *Thresholds*

Description

Compute three threshold values: minimum training presence, equal training sensitivity and specificity and maximum training sensitivity plus specificity together with fractional predicted area and the omission rate. If a test dataset is provided it returns also the equal test sensitivity and specificity and maximum test sensitivity plus specificity thresholds and the p-values of the one-tailed binomial exact test.

Usage

```
thresholds(model, type = NULL, test = NULL)
```

Arguments

model	SDMmodel object.
type	character. The output type used for "Maxent" and "Maxnet" methods, possible values are "cloglog" and "logistic".
test	SWD testing locations, if not provided it returns the training and test thresholds.

Details

The equal training sensitivity and specificity minimizes the difference between sensitivity and specificity. The one-tailed binomial test checks that test points are predicted no better than by a random prediction with the same fractional predicted area.

Value

data.frame with the thresholds.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Split presence locations in training (80%) and testing (20%) datasets
datasets <- trainValTest(data,
                         test = 0.2,
                         only_presence = TRUE)
train <- datasets[[1]]
test <- datasets[[2]]

# Train a model
model <- train(method = "Maxnet",
               data = train,
               fc = "l")

# Get the cloglog thresholds
thresholds(model,
           type = "cloglog")

# Get the logistic thresholds passing the test dataset
thresholds(model,
           type = "logistic",
           test = test)
```

`train` *Train*

Description

Train a model using one of the following methods: Artificial Neural Networks, Boosted Regression Trees, Maxent, Maxnet or Random Forest.

Usage

```
train(method, data, folds = NULL, progress = TRUE, ...)
```

Arguments

`method`	character or character vector. Method used to train the model, possible values are "ANN", "BRT", "Maxent", "Maxnet" or "RF", see details.
`data`	SWD object with presence and absence/background locations.
`folds`	list. Output of the function randomFolds or folds object created with other packages, see details.
`progress`	logical. If `TRUE` shows a progress bar during cross validation.
`...`	Arguments passed to the relative method, see details.

Details

- For the ANN method possible arguments are (for more details see nnet):
  - size: integer. Number of the units in the hidden layer.
  - decay numeric. Weight decay, default is 0.
  - rang numeric. Initial random weights, default is 0.7.
  - maxit integer. Maximum number of iterations, default is 100.
- For the BRT method possible arguments are (for more details see gbm):
  - distribution: character. Name of the distribution to use, default is "bernoulli".
  - n.trees: integer. Maximum number of tree to grow, default is 100.
  - interaction.depth: integer. Maximum depth of each tree, default is 1.
  - shrinkage: numeric. The shrinkage parameter, default is 0.1.
  - bag.fraction: numeric. Random fraction of data used in the tree expansion, default is 0.5.
- For the RF method the model is trained as classification. Possible arguments are (for more details see randomForest):
  - mtry: integer. Number of variable randomly sampled at each split, default is `floor(sqrt(number of variables))`
  - ntree: integer. Number of tree to grow, default is 500.
  - nodesize: integer. Minimum size of terminal nodes, default is 1.
- Maxent models are trained using the arguments `"removeduplicates=false"` and `"addsamplestobackground=false"`. Use the function thinData to remove duplicates and the function addSamplesToBg to add presence locations to background locations. For the Maxent method, possible arguments are:

  - reg: numeric. The value of the regularization multiplier, default is 1.
  - fc: character. The value of the feature classes, possible values are combinations of "l", "q", "p", "h" and "t", default is "lqph".
  - iter: numeric. Number of iterations used by the MaxEnt algorithm, default is 500.
- Maxnet models are trained using the argument `"addsamplestobackground = FALSE"`, use the function addSamplesToBg to add presence locations to background locations. For the Maxnet method, possible arguments are (for more details see maxnet):
  - reg: numeric. The value of the regularization intensity, default is 1.
  - fc: character. The value of the feature classes, possible values are combinations of "l", "q", "p", "h" and "t", default is "lqph".

The folds argument accepts also objects created with other packages: **ENMeval** or **blockCV**. In this case the function converts internally the folds into a format valid for **SDMtune**.

When multiple methods are given as `method` argument, the function returns a named list of model object, with the name corresponding to the used method, see examples.

Value

An SDMmodel or SDMmodelCV or a list of model objects.

Author(s)

Sergio Vignali

References

Venables, W. N. & Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0.

Brandon Greenwell, Bradley Boehmke, Jay Cunningham and GBM Developers (2019). gbm: Generalized Boosted Regression Models. `https://CRAN.R-project.org/package=gbm`.

A. Liaw and M. Wiener (2002). Classification and Regression by randomForest. R News 2(3), 18–22.

Hijmans, Robert J., Steven Phillips, John Leathwick, and Jane Elith. 2017. dismo: Species Distribution Modeling. `https://cran.r-project.org/package=dismo`.

Steven Phillips (2017). maxnet: Fitting 'Maxent' Species Distribution Models with 'glmnet'. `https://CRAN.R-project.org/package=maxnet`.

Muscarella, R., Galante, P.J., Soley-Guardia, M., Boria, R.A., Kass, J., Uriarte, M. and R.P. Anderson (2014). ENMeval: An R package for conducting spatially independent evaluations and estimating optimal model complexity for ecological niche models. Methods in Ecology and Evolution.

Roozbeh Valavi, Jane Elith, José Lahoz-Monfort and Gurutzeta Guillera-Arroita (2018). blockCV: Spatial and environmental blocking for k-fold cross-validation. `https://github.com/rvalavi/blockCV`.

See Also

randomFolds.

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

## Train a Maxent model
model <- train(method = "Maxent",
               data = data,
               fc = "l",
               reg = 1.5,
               iter = 700)

# Add samples to background. This should be done preparing the data before
# training the model without using
data <- addSamplesToBg(data)
model <- train("Maxent",
               data = data)

## Train a Maxnet model
model <- train(method = "Maxnet",
               data = data,
               fc = "lq",
               reg = 1.5)

## Cross Validation
# Create 4 random folds splitting only the presence data
folds <- randomFolds(data,
                     k = 4,
                     only_presence = TRUE)

model <- train(method = "Maxnet",
               data = data,
               fc = "l",
               reg = 0.8,
               folds = folds)

## Not run:
# Run only if you have the package ENMeval installed
```

```
## Block partition using the ENMeval package
require(ENMeval)
block_folds <- get.block(occ = data@coords[data@pa == 1, ],
                         bg.coords = data@coords[data@pa == 0, ])

model <- train(method = "Maxnet",
               data = data,
               fc = "l",
               reg = 0.8,
               folds = block_folds)

## Checkerboard1 partition using the ENMeval package
cb_folds <- get.checkerboard1(occ = data@coords[data@pa == 1, ],
                              env = predictors,
                              bg.coords = data@coords[data@pa == 0, ],
                              aggregation.factor = 4)

model <- train(method = "Maxnet",
               data = data,
               fc = "l",
               reg = 0.8,
               folds = cb_folds)

## Environmental block using the blockCV package
# Run only if you have the package blockCV
require(blockCV)
# Create sf object
sf_df <- sf::st_as_sf(cbind(data@coords, pa = data@pa),
                      coords = c("X", "Y"),
                      crs = terra::crs(predictors,
                                       proj = TRUE))

# Spatial blocks
spatial_folds <- cv_spatial(x = sf_df,
                            column = "pa",
                            rows_cols = c(8, 10),
                            k = 5,
                            hexagon = FALSE,
                            selection = "systematic")

model <- train(method = "Maxnet",
               data = data,
               fc = "l",
               reg = 0.8,
               folds = spatial_folds)
## End(Not run)

## Train presence absence models
# Prepare presence and absence locations
p_coords <- virtualSp$presence
a_coords <- virtualSp$absence
# Create SWD object
data <- prepareSWD(species = "Virtual species",
```

```
                   p = p_coords,
                   a = a_coords,
                   env = predictors[[1:5]])

## Train an Artificial Neural Network model
model <- train("ANN",
               data = data,
               size = 10)

## Train a Random Forest model
model <- train("RF",
               data = data,
               ntree = 300)

## Train a Boosted Regression Tree model
model <- train("BRT",
               data = data,
               n.trees = 300,
               shrinkage = 0.001)

## Multiple methods trained together with default arguments
output <- train(method = c("ANN", "BRT", "RF"),
                data = data,
                size = 10)
output$ANN
output$BRT
output$RF

## Multiple methods trained together passing extra arguments
output <- train(method = c("ANN", "BRT", "RF"),
                data = data,
                size = 10,
                ntree = 300,
                n.trees = 300,
                shrinkage = 0.001)
output
```

`trainValTest` *Train, Validation and Test datasets*

Description

Split a dataset randomly in training and testing datasets or training, validation and testing datasets.

Usage

```
trainValTest(x, test, val = 0, only_presence = FALSE, seed = NULL)
```

Arguments

x	SWD object containing the data that have to be split in training, validation and testing datasets.
test	numeric. The percentage of data withhold for testing.
val	numeric. The percentage of data withhold for validation, default is 0.
only_presence	logical. If TRUE the split is done only for the presence locations and all the background locations are included in each partition, used manly for presence-only methods, default is FALSE.
seed	numeric. The value used to set the seed in order to have consistent results, default is NULL.

Details

When only_presence = FALSE, the proportion of presence and absence is preserved.

Value

A list with the training, validation and testing or training and testing SWD objects accordingly.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Split presence locations in training (80%) and testing (20%) datasets
# and splitting only the presence locations
datasets <- trainValTest(data,
                         test = 0.2,
                         only_presence = TRUE)
train <- datasets[[1]]
test <- datasets[[2]]
```

```
# Split presence locations in training (60%), validation (20%) and testing
# (20%) datasets and splitting the presence and the absence locations
datasets <- trainValTest(data,
                         val = 0.2,
                         test = 0.2)
train <- datasets[[1]]
val <- datasets[[2]]
test <- datasets[[3]]
```

---

tss *True Skill Statistics*

Description

Compute the max TSS of a given model.

Usage

```
tss(model, test = NULL)
```

Arguments

model	SDMmodel or SDMmodelCV object.
test	SWD object when `model` is an SDMmodel object; logical or SWD object when `model` is an SDMmodelCV object. If not provided it computes the training TSS, see details.

Details

For SDMmodelCV objects, the function computes the mean of the training TSS values of the k-folds. If `test = TRUE` it computes the mean of the testing TSS values for the k-folds. If test is an SWD object, it computes the mean TSS values for the provided testing dataset.

Value

The value of the TSS of the given model.

Author(s)

Sergio Vignali

References

Allouche O., Tsoar A., Kadmon R., (2006). Assessing the accuracy of species distribution models: prevalence, kappa and the true skill statistic (TSS). Journal of Applied Ecology, 43(6), 1223–1232.

See Also

aicc and auc.

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Split presence locations in training (80%) and testing (20%) datasets
datasets <- trainValTest(data,
                         test = 0.2,
                         only_presence = TRUE)
train <- datasets[[1]]
test <- datasets[[2]]

# Train a model
model <- train(method = "Maxnet",
               data = train,
               fc = "l")

# Compute the training TSS
tss(model)

# Compute the testing TSS
tss(model,
    test = test)

# Same example but using cross validation instead of training and
# testing datasets. Create 4 random folds splitting only the presence
# locations
folds = randomFolds(train,
                    k = 4,
                    only_presence = TRUE)

model <- train(method = "Maxnet",
               data = train,
               fc = "l",
```

```
                folds = folds)

# Compute the training TSS
tss(model)

# Compute the testing TSS
tss(model,
    test = TRUE)

# Compute the TSS for the held apart testing dataset
tss(model,
    test = test)
```

---

varImp *Variable Importance*

Description

The function randomly permutes one variable at time (using training and absence/background datasets) and computes the decrease in training AUC. The result is normalized to percentages. Same implementation of MaxEnt java software but with the additional possibility of running several permutations to obtain a better estimate of the permutation importance. In case of more than one permutation (default is 10) the average of the decrease in training AUC is computed.

Usage

```
varImp(model, permut = 10, progress = TRUE)
```

Arguments

`model`	SDMmodel or SDMmodelCV object.
`permut`	integer. Number of permutations.
`progress`	logical. If `TRUE` shows a progress bar.

Details

Note that it could return values slightly different from MaxEnt Java software due to a different random permutation.

For SDMmodelCV objects the function returns the average and the standard deviation of the permutation importances of the single models.

Value

data.frame with the ordered permutation importance.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Split presence locations in training (80%) and testing (20%) datasets
datasets <- trainValTest(data,
                         test = 0.2,
                         only_presence = TRUE)
train <- datasets[[1]]
test <- datasets[[2]]

# Train a model
model <- train(method = "Maxnet",
               data = train,
               fc = "l")

# Compute variable importance
vi <- varImp(model,
             permut = 5)
vi

# Same example but using cross validation instead of training and testing
# datasets
# Create 4 random folds splitting only the presence locations
folds = randomFolds(data,
                    k = 4,
                    only_presence = TRUE)

model <- train(method = "Maxnet",
               data = data,
               fc = "l",
               folds = folds)

# Compute variable importance
vi <- varImp(model,
             permut = 5)
vi
```

varSel *Variable Selection*

Description

The function performs a data-driven variable selection. Starting from the provided model it iterates through all the variables starting from the one with the highest contribution (permutation importance or maxent percent contribution). If the variable is correlated with other variables (according to the given method and threshold) it performs a Jackknife test and among the correlated variables it removes the one that results in the best performing model when removed (according to the given metric for the training dataset). The process is repeated until the remaining variables are not highly correlated anymore.

Usage

```
varSel(
  model,
  metric,
  bg4cor,
  test = NULL,
  env = NULL,
  method = "spearman",
  cor_th = 0.7,
  permut = 10,
  use_pc = FALSE,
  interactive = TRUE,
  progress = TRUE,
  verbose = TRUE
)
```

Arguments

`model`	SDMmodel or SDMmodelCV object.
`metric`	character. The metric used to evaluate the models, possible values are: "auc", "tss" and "aicc".
`bg4cor`	SWD object. Background locations used to test the correlation between environmental variables.
`test`	SWD. Test dataset used to evaluate the model, not used with aicc and SDMmodelCV objects.
`env`	rast containing the environmental variables, used only with "aicc".
`method`	character. The method used to compute the correlation matrix.
`cor_th`	numeric. The correlation threshold used to select highly correlated variables.
`permut`	integer. Number of permutations.

use_pc	logical, use percent contribution. If TRUE and the model is trained using the Maxent method, the algorithm uses the percent contribution computed by Maxent software to score the variable importance.
interactive	logical. If FALSE the interactive chart is not created.
progress	logical. If TRUE shows a progress bar.
verbose	logical. If TRUE prints informative messages.

Details

An interactive chart showing in real-time the steps performed by the algorithm is displayed in the Viewer pane.

To find highly correlated variables the following formula is used:

$$|coeff| \le cor_th$$

Value

The SDMmodel or SDMmodelCV object trained using the selected variables.

Author(s)

Sergio Vignali

Examples

```
# Acquire environmental variables
files <- list.files(path = file.path(system.file(package = "dismo"), "ex"),
                    pattern = "grd",
                    full.names = TRUE)

predictors <- terra::rast(files)

# Prepare presence and background locations
p_coords <- virtualSp$presence
bg_coords <- virtualSp$background

# Create SWD object
data <- prepareSWD(species = "Virtual species",
                   p = p_coords,
                   a = bg_coords,
                   env = predictors,
                   categorical = "biome")

# Split presence locations in training (80%) and testing (20%) datasets
datasets <- trainValTest(data,
                         test = 0.2,
                         only_presence = TRUE)
train <- datasets[[1]]
test <- datasets[[2]]

# Train a model
```

```
model <- train(method = "Maxnet",
               data = train,
               fc = "l")

# Prepare background locations to test autocorrelation, this usually gives a
# warning message given that less than 10000 points can be randomly sampled
bg_coords <- terra::spatSample(predictors,
                               size = 9000,
                               method = "random",
                               na.rm = TRUE,
                               xy = TRUE,
                               values = FALSE)

bg <- prepareSWD(species = "Virtual species",
                 a = bg_coords,
                 env = predictors,
                 categorical = "biome")

## Not run:
# Remove variables with correlation higher than 0.7 accounting for the AUC,
# in the following example the variable importance is computed as permutation
# importance
vs <- varSel(model,
             metric = "auc",
             bg4cor = bg,
             test = test,
             cor_th = 0.7,
             permut = 1)
vs

# Remove variables with correlation higher than 0.7 accounting for the TSS,
# in the following example the variable importance is the MaxEnt percent
# contribution
# Train a model
model <- train(method = "Maxent",
               data = train,
               fc = "l")

vs <- varSel(model,
             metric = "tss",
             bg4cor = bg,
             test = test,
             cor_th = 0.7,
             use_pc = TRUE)
vs

# Remove variables with correlation higher than 0.7 accounting for the aicc,
# in the following example the variable importance is the MaxEnt percent
# contribution
vs <- varSel(model,
             metric = "aicc",
             bg4cor = bg,
             cor_th = 0.7,
```

```
            use_pc = TRUE,
            env = predictors)
vs
## End(Not run)
```

---

virtualSp *Virtual Species*

Description

Dataset containing a random generated virtual species. The purpose of this dataset is to demonstrate the use of the functions included in the package.

Usage

```
virtualSp
```

Format

A list with five elements:

presence 400 random generated coordinates for the presence locations.

absence 300 random generated coordinates for the absence locations.

background 5000 random generated coordinates for the background locations.

pa_map Presence absence map used to extract the presence and absence locations.

prob_map Probability map of the random generated virtual species.

Details

The random species has been generated using the package **virtualspecies**.

References

Leroy, B. , Meynard, C. N., Bellard, C. and Courchamp, F. (2016), virtualspecies, an R package to generate virtual species distributions. Ecography, 39: 599-607. doi:10.1111/ecog.01388

Index