Package 'sjSDM'

July 23, 2025

Type Package

Title Scalable Joint Species Distribution Modeling

Version 1.0.6

Description

A scalable and fast method for estimating joint Species Distribution Models (jSDMs) for big community data, including eDNA data. The package estimates a full (i.e. non-latent) jSDM with different response distributions (including the traditional multivariate probit model). The package allows to perform variation partitioning (VP) / ANOVA on the fitted models to separate the contribution of environmental, spatial, and biotic associations. In addition, the total R-squared can be further partitioned per species and site to reveal the internal metacommunity structure, see Leibold et al., <doi:10.1111/oik.08618>. The internal structure can then be regressed against environmental and spatial distinctiveness, richness, and traits to analyze metacommunity assembly processes. The package includes support for accounting for spatial autocorrelation and the option to fit responses using deep neural networks instead of a standard linear predictor. As described in Pichler & Hartig (2021) <doi:10.1111/2041-210X.13687>, scalability is achieved by using a Monte Carlo approximation of the joint likelihood implemented via 'PyTorch' and 'reticulate', which can be run on CPUs or GPUs.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 3.0)

Imports reticulate, stats, mvtnorm, utils, rstudioapi, abind, graphics, grDevices, Metrics, parallel, mgcv, cli, crayon, ggplot2, checkmate, mathjaxr, ggtern, beeswarm, qgam, scales

Suggests testthat, knitr, rmarkdown, iml, fields

RoxygenNote 7.3.2

URL https://github.com/TheoreticalEcology/s-jSDM/

BugReports https://github.com/TheoreticalEcology/s-jSDM/issues

VignetteBuilder knitr

RdMacros mathjaxr

NeedsCompilation no

Author Maximilian Pichler [aut, cre] (ORCID:
<https://orcid.org/0000-0003-2252-8327>),
Florian Hartig [aut] (ORCID: <https://orcid.org/0000-0002-6255-9059>),
Wang Cai [ctb]

Maintainer Maximilian Pichler <maximilian.pichler@biologie.uni-regensburg.de>

Repository CRAN

Date/Publication 2024-08-19 12:00:05 UTC

Contents

`AccSGD` *AccSGD*

Description

accelerated stochastic gradient, see Kidambi et al., 2018 for details

Usage

```
AccSGD(kappa = 1000, xi = 10, small_const = 0.7, weight_decay = 0)
```

Arguments

`kappa`	long step
`xi`	advantage parameter
`small_const`	small constant
`weight_decay`	l2 penalty on weights

Value

Anonymous function that returns optimizer when called.

References

Kidambi, R., Netrapalli, P., Jain, P., & Kakade, S. (2018, February). On the insufficiency of existing momentum schemes for stochastic optimization. In 2018 Information Theory and Applications Workshop (ITA) (pp. 1-9). IEEE.

AdaBound *AdaBound*

Description

adaptive gradient methods with dynamic bound of learning rate, see Luo et al., 2019 for details

Usage

```
AdaBound(
  betas = c(0.9, 0.999),
  final_lr = 0.1,
  gamma = 0.001,
  eps = 1e-08,
  weight_decay = 0,
  amsbound = TRUE
)
```

Arguments

betas	betas
final_lr	eps
gamma	small_const
eps	eps
weight_decay	weight_decay
amsbound	amsbound

Value

Anonymous function that returns optimizer when called.

References

Luo, L., Xiong, Y., Liu, Y., & Sun, X. (2019). Adaptive gradient methods with dynamic bound of learning rate. arXiv preprint arXiv:1902.09843.

Adamax *Adamax*

Description

Adamax optimizer, see Kingma and Ba, 2014

Usage

```
Adamax(betas = c(0.9, 0.999), eps = 1e-08, weight_decay = 0.002)
```

Arguments

betas	exponential decay rates
eps	fuzz factor
weight_decay	l2 penalty on weights

Value

Anonymous function that returns optimizer when called.

References

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

anova.sjSDM *Anova / Variation partitioning*

Description

Compute variance explained by the three fractions env, space, associations

Usage

```
## S3 method for class 'sjSDM'
anova(object, samples = 5000L, verbose = TRUE, ...)
```

Arguments

object	model of object sjSDM
samples	Number of Monte Carlo samples
verbose	TRUE or FALSE, indicating whether progress should be printed or not
...	optional arguments which are passed to the calculation of the logLikelihood

Details

The ANOVA function removes each of the three fractions (Environment, Space, Associations) and measures the drop in variance explained, and thus the importance of the three fractions.

Variance explained is measured by Deviance as well as the pseudo-R2 metrics of Nagelkerke and McFadden

In downstream functions such as `plot.sjSDManova` or `plot.sjSDManova` with `add_shared=TRUE`. The anova can get unstable for many species and few occurrences/observations. We recommend using large numbers for 'samples'.

Value

An S3 class of type 'sjSDManova' including the following components:

`results`	Data frame of results.
`to_print`	Data frame, summarized results for type I anova.
`N`	Number of observations (sites).
`spatial`	Logical, spatial model or not.
`species`	individual species R2s.
`sites`	individual site R2s.
`lls`	individual site by species negative-log-likelihood values.
`model`	model

Implemented S3 methods are `print.sjSDManova` and `plot.sjSDManova`

See Also

`plot.sjSDManova`, `print.sjSDManova`,`summary.sjSDManova`, `plot.sjSDMinternalStructure`

Examples

```
## Not run:
library(sjSDM)
# simulate community:
community = simulate_SDM(env = 3L, species = 10L, sites = 100L)

Occ <- community$response
Env <- community$env_weights
SP <- data.frame(matrix(rnorm(200, 0, 0.3), 100, 2)) # spatial coordinates


# fit model:
model <- sjSDM(Y = Occ,
               env = linear(data = Env, formula = ~X1+X2+X3),
               spatial = linear(data = SP, formula = ~0+X1*X2),
               family=binomial("probit"),
               verbose = FALSE,
               iter = 20) # increase iter for real analysis
```

```
# Calculate ANOVA for env, space, associations, for details see ?anova.sjSDM
an = anova(model, samples = 10, verbose = FALSE) # increase iter for real analysis

# Show anova fractions
plot(an)

# ANOVA tables with different way to handle fractions
summary(an)
summary(an, fractions = "discard")
summary(an, fractions = "proportional")
summary(an, fractions = "equal")

# Internal structure
int = internalStructure(an, fractions = "proportional")

print(int)

plot(int) # default is negative values will be set to 0
plot(int, negatives = "scale") # global rescaling of all values to range 0-1
plot(int, negatives = "raw") # negative values will be discarded

plotAssemblyEffects(int)
plotAssemblyEffects(int, negatives = "floor")
plotAssemblyEffects(int, response = "sites", pred = as.factor(c(rep(1, 50), rep(2, 50))))
plotAssemblyEffects(int, response = "species", pred = runif(10))
plotAssemblyEffects(int, response = "species", pred = as.factor(c(rep(1, 5), rep(2, 5))))

## End(Not run)
```

`bioticStruct` *biotic structure*

Description

define biotic (species-species) association (interaction) structure

Usage

```
bioticStruct(
  df = NULL,
  lambda = 0,
  alpha = 0.5,
  on_diag = FALSE,
  reg_on_Cov = TRUE,
  inverse = FALSE,
  diag = FALSE
)
```

Arguments

df	degree of freedom for covariance parametrization, if NULL df is set to ncol(Y)/2
lambda	lambda penalty, strength of regularization: $\lambda * (lasso + ridge)$
alpha	weighting between lasso and ridge: $(1-\alpha)*\|covariances\|+\alpha\|\|covariances\|\|^2$
on_diag	regularization on diagonals
reg_on_Cov	regularization on covariance matrix
inverse	regularization on the inverse covariance matrix
diag	use diagonal matrix with zeros (internal usage)

Value

An S3 class of type 'bioticStruct' including the following components:

l1_cov	L1 regularization strength.
l2_cov	L2 regularization strength.
inverse	Logical, use inverse covariance matrix or not.
diag	Logical, use diagonal matrix or not.
reg_on_Cov	Logical, regularize covariance matrix or not.
on_diag	Logical, regularize diagonals or not.

Implemented S3 methods include print.bioticStruct

See Also

sjSDM

Examples

```
## Not run:

# Basic workflow:
## simulate community:
com = simulate_SDM(env = 3L, species = 7L, sites = 100L)

## fit model:
model = sjSDM(Y = com$response,env = com$env_weights, iter = 50L,
              verbose = FALSE)
# increase iter for your own data

# Default distribution is binomial("probit"). Alternatively, you can use
# binomial(logit), poisson("log"), "nbinom" (with log, still somewhat
# experimental) and gaussian("identity")

coef(model)
summary(model)
getCov(model)
```

```
## plot results
species=c("sp1","sp2","sp3","sp4","sp5","sp6","sp7")
group=c("mammal","bird","fish","fish","mammal","amphibian","amphibian")
group = data.frame(species=species,group=group)
plot(model,group=group)

## calculate post-hoc p-values:
p = getSe(model)
summary(p)

## or turn on the option in the sjSDM function:
model = sjSDM(Y = com$response, env = com$env_weights, se = TRUE,
              family = binomial("probit"),
              iter = 2L,
              verbose = FALSE)
summary(model)

## fit model with interactions:
model = sjSDM(Y = com$response,
              env = linear(data = com$env_weights, formula = ~X1:X2 + X3),
              se = TRUE,
              iter = 2L,
              verbose = FALSE) # increase iter for your own data
summary(model)

## without intercept:
model = update(model, env_formula = ~0+X1:X2 + X3,
               verbose = FALSE)

summary(model)

## predict with model:
preds = predict(model, newdata = com$env_weights)

## calculate R-squared:
R2 = Rsquared(model)
print(R2)

# With spatial terms:
## linear spatial model
XY = matrix(rnorm(200), 100, 2)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(XY, ~0+X1:X2),
              iter = 50L,
              verbose = FALSE) # increase iter for your own data
summary(model)
predict(model, newdata = com$env_weights, SP = XY)
R2 = Rsquared(model)
print(R2)

## Using spatial eigenvectors as predictors to account
## for spatial autocorrelation is a common approach:
SPV = generateSpatialEV(XY)
```

```
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(SPV, ~0+., lambda = 0.1),
              iter = 50L,
              verbose = FALSE) # increase iter for your own data
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)

## Visualize internal meta-community structure
an = anova(model,
           verbose = FALSE)

internal = internalStructure(an)
plot(internal)

## Visualize community assemlby effects

plotAssemblyEffects(internal)

### see ?anova.sjSDM for mroe details


## non-linear(deep neural network) model
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = DNN(SPV,hidden = c(5L, 5L), ~0+.),
              iter = 2L,# increase iter for your own data
              verbose = FALSE)
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)


# Regularization
## lambda is the regularization strength
## alpha weights the lasso or ridge penalty:
## - alpha = 0 --> pure lasso
## - alpha = 1.0 --> pure ridge
model = sjSDM(Y = com$response,
              # mix of lasso and ridge
              env = linear(com$env_weights, lambda = 0.01, alpha = 0.5),
              # we can do the same for the species-species associations
              biotic = bioticStruct(lambda = 0.01, alpha = 0.5),
              iter = 2L,# increase iter for your own data
              verbose = FALSE)
summary(model)
coef(model)
getCov(model)



# Anova
com = simulate_SDM(env = 3L, species = 15L, sites = 200L, correlation = TRUE)

XY = matrix(rnorm(400), 200, 2)
SPV = generateSpatialEV(XY)
```

```
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(SPV, ~0+.),
              verbose = FALSE,
              iter = 50L) # increase iter for your own data
result = anova(model, verbose = FALSE)
print(result)
plot(result)

## visualize internal meta-community structure
internal = internalStructure(an)
plot(internal)



# Deep neural networks
## we can fit also a deep neural network instead of a linear model:
model = sjSDM(Y = com$response,
              env = DNN(com$env_weights, hidden = c(10L, 10L, 10L)),
              verbose = FALSE,
              iter = 2L) # increase iter for your own data
summary(model)
getCov(model)
pred = predict(model, newdata = com$env_weights)

## extract weights
weights = getWeights(model)

## we can also assign weights:
setWeights(model, weights)

## with regularization:
model = sjSDM(Y = com$response,
              # mix of lasso and ridge
              env = DNN(com$env_weights, lambda = 0.01, alpha = 0.5),
              # we can do the same for the species-species associations
              biotic = bioticStruct(lambda = 0.01, alpha = 0.5),
              verbose = FALSE,
              iter = 2L) # increase iter for your own data
getCov(model)
getWeights(model)

## End(Not run)
```

`butterflies` *butterflies*

Description

This dataset is from doi:10.1111/2041210X.13106.

Usage

```
butterflies
```

Format

A 'list': List of 3.

env data.frame with 4 environmental covariates

PA Presence-absence data for 55 butterfly species

lat_lon Coordinates for the sites

Details

This is a dataset about butterfly communities. It consists of 2609 sites and 55 species.

Author(s)

Maximilian Pichler

Source

doi:10.1111/2041210X.13106

Examples

```
## Not run:
  PA = butterflies$PA
  E = butterflies$env
  LatLon = butterflies$lat_lon

  m = sjSDM(PA,
            scale(E),
            spatial = DNN(scale(LatLon), formula = ~0+.),
            se = TRUE,
            iter = 20L, # increase to 100
            step_size = 200L,
            verbose = FALSE)
  summary(m)
  plot(m)


## End(Not run)
```

checkModel *check model check model and rebuild if necessary*

Description

check model check model and rebuild if necessary

Usage

```
checkModel(object)
```

Arguments

object	of class sjSDM

check_module *check module*

Description

check if module is loaded

Usage

```
check_module()
```

coef.sjSDM *Return coefficients from a fitted sjSDM model*

Description

Return coefficients from a fitted sjSDM model

Usage

```
## S3 method for class 'sjSDM'
coef(object, ...)
```

Arguments

object	a model fitted by sjSDM
...	optional arguments for compatibility with the generic function, no function implemented

Value

Matrix of environmental coefficients or list of environmental and spatial coefficients for spatial models.

`DiffGrad` *DiffGrad*

Description

DiffGrad

Usage

```
DiffGrad(betas = c(0.9, 0.999), eps = 1e-08, weight_decay = 0)
```

Arguments

`betas`	betas
`eps`	eps
`weight_decay`	weight_decay

Value

Anonymous function that returns optimizer when called.

`DNN` *Non-linear model (deep neural network) of environmental responses*

Description

specify the model to be fitted

Usage

```
DNN(
  data = NULL,
  formula = NULL,
  hidden = c(10L, 10L, 10L),
  activation = "selu",
  bias = TRUE,
  lambda = 0,
  alpha = 0.5,
  dropout = 0
)
```

Arguments

data	matrix of environmental predictors
formula	formula object for predictors
hidden	hidden units in layers, length of hidden corresponds to number of layers
activation	activation functions, can be of length one, or a vector of activation functions for each layer. Currently supported: tanh, relu, leakyrelu, selu, or sigmoid
bias	whether use biases in the layers, can be of length one, or a vector (number of hidden layers including (last layer) but not first layer (intercept in first layer is specified by formula)) of logicals for each layer.
lambda	lambda penalty, strength of regularization: $\lambda * (lasso + ridge)$
alpha	weighting between lasso and ridge: $(1 - \alpha) * \|weights\| + \alpha \|\|weights\|\|^2$
dropout	probability of dropout rate

Value

An S3 class of type 'DNN' including the following components:

formula	Model matrix formula
X	Model matrix of covariates
data	Raw data
l1_coef	L1 regularization strength, can be -99 if lambda = 0.0
l2_coef	L2 regularization strength, can be -99 if lambda = 0.0
hidden	Integer vector of hidden neurons in the deep neural network. Length of vector corresponds to the number of hidden layers.
activation	Character vector of activation functions.
bias	Logical vector whether to use bias or not in each hidden layer.

Implemented S3 methods include print.DNN

See Also

linear, sjSDM

Examples

```
## Not run:

# Basic workflow:
## simulate community:
com = simulate_SDM(env = 3L, species = 7L, sites = 100L)

## fit model:
model = sjSDM(Y = com$response,env = com$env_weights, iter = 50L,
              verbose = FALSE)
# increase iter for your own data
```

```
# Default distribution is binomial("probit"). Alternatively, you can use
# binomial(logit), poisson("log"), "nbinom" (with log, still somewhat
# experimental) and gaussian("identity")

coef(model)
summary(model)
getCov(model)

## plot results
species=c("sp1","sp2","sp3","sp4","sp5","sp6","sp7")
group=c("mammal","bird","fish","fish","mammal","amphibian","amphibian")
group = data.frame(species=species,group=group)
plot(model,group=group)

## calculate post-hoc p-values:
p = getSe(model)
summary(p)

## or turn on the option in the sjSDM function:
model = sjSDM(Y = com$response, env = com$env_weights, se = TRUE,
              family = binomial("probit"),
              iter = 2L,
              verbose = FALSE)
summary(model)

## fit model with interactions:
model = sjSDM(Y = com$response,
              env = linear(data = com$env_weights, formula = ~X1:X2 + X3),
              se = TRUE,
              iter = 2L,
              verbose = FALSE) # increase iter for your own data
summary(model)

## without intercept:
model = update(model, env_formula = ~0+X1:X2 + X3,
               verbose = FALSE)

summary(model)

## predict with model:
preds = predict(model, newdata = com$env_weights)

## calculate R-squared:
R2 = Rsquared(model)
print(R2)

# With spatial terms:
## linear spatial model
XY = matrix(rnorm(200), 100, 2)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(XY, ~0+X1:X2),
              iter = 50L,
              verbose = FALSE) # increase iter for your own data
```

```
summary(model)
predict(model, newdata = com$env_weights, SP = XY)
R2 = Rsquared(model)
print(R2)

## Using spatial eigenvectors as predictors to account
## for spatial autocorrelation is a common approach:
SPV = generateSpatialEV(XY)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(SPV, ~0+., lambda = 0.1),
              iter = 50L,
              verbose = FALSE) # increase iter for your own data
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)

## Visualize internal meta-community structure
an = anova(model,
           verbose = FALSE)

internal = internalStructure(an)
plot(internal)

## Visualize community assemlby effects

plotAssemblyEffects(internal)

### see ?anova.sjSDM for mroe details


## non-linear(deep neural network) model
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = DNN(SPV,hidden = c(5L, 5L), ~0+.),
              iter = 2L,# increase iter for your own data
              verbose = FALSE)
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)


# Regularization
## lambda is the regularization strength
## alpha weights the lasso or ridge penalty:
## - alpha = 0 --> pure lasso
## - alpha = 1.0 --> pure ridge
model = sjSDM(Y = com$response,
              # mix of lasso and ridge
              env = linear(com$env_weights, lambda = 0.01, alpha = 0.5),
              # we can do the same for the species-species associations
              biotic = bioticStruct(lambda = 0.01, alpha = 0.5),
              iter = 2L,# increase iter for your own data
              verbose = FALSE)
summary(model)
coef(model)
getCov(model)
```

```
# Anova
com = simulate_SDM(env = 3L, species = 15L, sites = 200L, correlation = TRUE)

XY = matrix(rnorm(400), 200, 2)
SPV = generateSpatialEV(XY)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(SPV, ~0+.),
              verbose = FALSE,
              iter = 50L) # increase iter for your own data
result = anova(model, verbose = FALSE)
print(result)
plot(result)

## visualize internal meta-community structure
internal = internalStructure(an)
plot(internal)



# Deep neural networks
## we can fit also a deep neural network instead of a linear model:
model = sjSDM(Y = com$response,
              env = DNN(com$env_weights, hidden = c(10L, 10L, 10L)),
              verbose = FALSE,
              iter = 2L) # increase iter for your own data
summary(model)
getCov(model)
pred = predict(model, newdata = com$env_weights)

## extract weights
weights = getWeights(model)

## we can also assign weights:
setWeights(model, weights)

## with regularization:
model = sjSDM(Y = com$response,
              # mix of lasso and ridge
              env = DNN(com$env_weights, lambda = 0.01, alpha = 0.5),
              # we can do the same for the species-species associations
              biotic = bioticStruct(lambda = 0.01, alpha = 0.5),
              verbose = FALSE,
              iter = 2L) # increase iter for your own data
getCov(model)
getWeights(model)

## End(Not run)
```

eucalypts *eucalypts*

Description

This dataset is from doi:10.1111/2041210x.12180.

Usage

```
eucalypts
```

Format

A 'list': List of 3.

env data.frame with 7 environmental covariates

PA Presence-absence data for 12 eucalypts species

lat_lon Coordinates for the sites

Details

This is a dataset about butterfly communities. It consists of 458 sites and 12 species.

Author(s)

Maximilian Pichler

Source

doi:10.1111/2041210x.12180

Examples

```
## Not run:
  PA = eucalypts$PA
  E = eucalypts$env
  LatLon = eucalypts$lat_lon

  m = sjSDM(PA,
            scale(E),
            spatial = DNN(scale(LatLon), formula = ~0+.),
            se = TRUE,
            verbose = FALSE)
  summary(m)
  plot(m)


## End(Not run)
```

generateSpatialEV *Generate spatial eigenvectors*

Description

Generates a Moran's eigenvector map of the distance matrix. See Dray, Legendre, and Peres-Neto, 2006 for more information.

Usage

```
generateSpatialEV(coords = NULL, threshold = 0)
```

Arguments

coords	matrix or data.frame of coordinates
threshold	ignore distances greater than threshold

Value

Matrix of spatial eigenvectors.

References

Dray, S., Legendre, P., & Peres-Neto, P. R. (2006). Spatial modelling: a comprehensive framework for principal coordinate analysis of neighbour matrices (PCNM). Ecological modelling, 196(3-4), 483-493.

getCor *getCor*

Description

get species-species association correlation matrix

Usage

```
getCor(object)

## S3 method for class 'sjSDM'
getCor(object)
```

Arguments

object	a model fitted by sjSDM, or sjSDM with DNN object

Value

Matrix of dimensions species by species corresponding to the covariance (occurrence) matrix.

See Also

sjSDM,DNN

---

getCov *getCov*

Description

get species-species association (covariance) matrix

Usage

```
getCov(object)

## S3 method for class 'sjSDM'
getCov(object)
```

Arguments

object	a model fitted by sjSDM, or sjSDM with DNN object

Value

Matrix of dimensions species by species corresponding to the covariance (occurrence) matrix.

See Also

sjSDM,DNN

---

getImportance *getImportance*

Description

variation partitioning with coefficients

Usage

```
getImportance(beta, sp = NULL, association, covX, covSP = NULL)
```

Arguments

beta	abiotic weights
sp	spatial weights
association	species associations
covX	environmental covariance matrix
covSP	spatial covariance matrix

Author(s)

Maximilian Pichler

---

getSe *Post hoc calculation of standard errors*

Description

Post hoc calculation of standard errors

Usage

```
getSe(object, step_size = NULL, parallel = 0L)
```

Arguments

object	a model fitted by sjSDM
step_size	batch size for stochastic gradient descent
parallel	number of cpu cores for the data loader, only necessary for large datasets

Value

The object passed to this function but the object$se field contains the standard errors now

getWeights *Get weights*

Description

return weights of each layer

Usage

```
getWeights(object)

## S3 method for class 'sjSDM'
getWeights(object)
```

Arguments

object	object of class sjSDM with DNN

Value

- layers - list of layer weights
- sigma - weight to construct covariance matrix

importance *Importance of environmental, spatial and association components*

Description

Computes standardized variance components with respect to abiotic, biotic, and spatial effect groups.

Usage

```
importance(x, save_memory = TRUE, ...)
```

Arguments

x	object fitted by sjSDM or a list with beta, the association matrix, and the correlation matrix of the predictors, see details below
save_memory	use torch backend to calculate importance with single precision floats
...	additional arguments

Details

This approach is based on Ovaskainen et al., 2017, and also used in Leibold et al., 2021. Unlike the anova.sjSDM function in the sjSDM package, importance is not calculated by explicitly switching a particular model component of and refitting the model, but essentially by setting it ineffective.

Although we have no hard reasons to discourage the use of this function, we have decided in sjSDM to measure importance maninly based on a traditional ANOVA approach. We therefore recommend users to use the anova.sjSDM.

This function is maintained hidden for comparison / benchmarking purpose, and in case there is a need to use it in the future. If you want to access it, use sjSDM:::importance.

Value

An S3 class of type 'sjSDMimportance' including the following components:

names	Character vector, species names.
res	Data frame of results.
spatial	Logical, spatial model or not.

Implemented S3 methods include print.sjSDMimportance and plot.sjSDMimportance

Author(s)

Maximilian Pichler

References

Ovaskainen, O., Tikhonov, G., Norberg, A., Guillaume Blanchet, F., Duan, L., Dunson, D., ... & Abrego, N. (2017). How to make more out of community data? A conceptual framework and its implementation as models and software. Ecology letters, 20(5), 561-576.

Leibold, M. A., Rudolph, F. J., Blanchet, F. G., De Meester, L., Gravel, D., Hartig, F., ... & Chase, J. M. (2021). The internal structure of metacommunities. Oikos.

See Also

print.sjSDMimportance, plot.sjSDMimportance

Examples

```
## Not run:
library(sjSDM)
com = simulate_SDM(sites = 300L, species = 12L,
                   link = "identical", response = "identical")
Raw = com$response
SP = matrix(rnorm(300*2), 300, 2)
SPweights = matrix(rnorm(12L), 1L)
SPweights[1,1:6] = 0
Y = Raw + (SP[,1,drop=FALSE]*SP[,2,drop=FALSE]) %*% SPweights
Y = ifelse(Y > 0, 1, 0)
```

```
model = sjSDM(Y = Y,env = linear(com$env_weights, lambda = 0.001),
              spatial = linear(SP,formula = ~0+X1:X2, lambda = 0.001),
              biotic = bioticStruct(lambda = 0.001),iter = 40L, verbose = FALSE)
imp = importance(model)
plot(imp)

## End(Not run)
```

`installation_help` *Installation help*

Description

Trouble shooting guide for the installation of the sjSDM package

We provide a function `install_sjSDM` to install automatically all necessary python dependencies but it can fail sometimes because of individual system settings or if other python/conda installations get into the way.

'PyTorch' Installation - Before you start

A few notes before you start with the installation (skip this point if you do not know 'conda'):

- existing 'conda' installations: make sure you have the latest conda3/miniconda3 version and remove unnecessary 'conda' installations.
- existing 'conda'/'virtualenv' environments (skip this point if you do not know 'conda'): we currently enforce the usage of a specific environment called 'r-sjsdm', so if you want use a custom environment it should be named 'r-sjsdm'

Windows - automatic installation

Sometimes the automatic 'miniconda' installation (via `install_sjSDM`) doesn't work because of white spaces in the user's name. But you can easily download and install 'conda' on your own:

Download and install the latest 'conda' version

Afterwards run:
`install_sjSDM(version = c("gpu")) # or "cpu" if you do not have a proper gpu device`

Reload the package and run the example , if this doesn't work:

- Restart RStudio
- Install manually 'pytorch', see the following section

Windows - manual installation

Download and install the latest 'conda' version:

- Install the latest 'conda' version
- Open the command window (cmd.exe - hit windows key + r and write cmd)

Run in cmd.exe:

```
$ conda create --name r-sjsdm python=3.7
$ conda activate r-sjsdm
$ conda install pytorch torchvision cpuonly -c pytorch # cpu
$ conda install pytorch torchvision cudatoolkit=11.3 -c pytorch #gpu
$ python -m pip install pyro-ppl torch_optimizer madgrad
```

Restart R, try to run the example, and if this doesn't work:

- Restart RStudio
- See the 'Help and bugs' section

Linux - automatic installation

Run in R:
`install_sjSDM(version = c("gpu")) # or "cpu" if you do not have a proper 'gpu' device`

Restart R try to run the example, if this doesn't work:

- Restart RStudio
- Install manually 'PyTorch', see the following section

Linux - manual installation

We strongly advise to use a 'conda' environment but a virtual env should also work. The only requirement is that it is named 'r-sjsdm'

Download and install the latest 'conda' version:

- Install the latest 'conda' version
- Open your terminal

Run in your terminal:

```
$ conda create --name r-sjsdm python=3.7
$ conda activate r-sjsdm
$ conda install pytorch torchvision cpuonly -c pytorch # cpu
$ conda install pytorch torchvision cudatoolkit=11.3 -c pytorch #gpu
$ python -m pip install pyro-ppl torch_optimizer madgrad
```

Restart R try to run the example, if this doesn't work:

- Restart RStudio
- See the 'Help and bugs' section

MacOS - automatic installation

Run in R:
`install_sjSDM(version = c("cpu"))`

Restart R try to run the example, if this doesn't work:

- Restart RStudio
- Install manually 'PyTorch', see the following section

MacOS - manual installation

Download and install the latest 'conda' version:

- Install the latest 'conda' version
- Open your terminal

Run in your terminal:

```
$ conda create --name r-sjsdm python=3.7
$ conda activate r-sjsdm
$ python -m pip install torch torchvision torchaudio
$ python -m pip install pyro-ppl torch_optimizer madgrad
```

Restart R try to run the example from, if this doesn't work:

- Restart RStudio
- See the 'Help and bugs' section

Help and bugs

To report bugs or ask for help, post a reproducible example via the sjSDM issue tracker with a copy of the `install_diagnostic` output as a quote.

Author(s)

Maintainer: Maximilian Pichler <maximilian.pichler@biologie.uni-regensburg.de> (ORCID)

Authors:

- Florian Hartig <florian.hartig@biologie.uni-regensburg.de> (ORCID)

Other contributors:

- Wang Cai <caiwang0503@163.com> [contributor]

See Also

Useful links:

- https://github.com/TheoreticalEcology/s-jSDM/
- Report bugs at https://github.com/TheoreticalEcology/s-jSDM/issues

install_diagnostic *install diagnostic*

Description

Print information about available conda environments, python configs, and pytorch versions.

Usage

```
install_diagnostic()
```

Details

If the trouble shooting guide `installation_help` did not help with the installation, please create an issue on issue tracker with the output of this function as a quote.

Value

No return value, called to extract dependency information.

See Also

`installation_help`, `install_sjSDM`

install_sjSDM *Install sjSDM and its dependencies*

Description

Install sjSDM and its dependencies

Usage

```
install_sjSDM(
  conda = "auto",
  version = c("cpu", "gpu"),
  restart_session = TRUE,
  ...
)
```

Arguments

conda	path to conda
version	version = "cpu" for CPU version, or "gpu" for GPU version. (note MacOS users have to install 'cuda' binaries by themselves)
restart_session	Restart R session after installing (note this will only occur within RStudio).
...	not supported

Value

No return value, called for side effects (installation of 'python' dependencies).

`internalStructure` *Plot internal metacommunity structure*

Description

Plot internal metacommunity structure

Usage

```
internalStructure(
  object,
  Rsquared = c("McFadden", "Nagelkerke"),
  fractions = c("discard", "proportional", "equal"),
  negatives = c("floor", "scale", "raw"),
  plot = FALSE
)
```

Arguments

`object`	anova object from `anova.sjSDM`
`Rsquared`	which R squared should be used, McFadden or Nagelkerke (McFadden is default)
`fractions`	how to handle shared fractions
`negatives`	how to handle negative R squareds
`plot`	should the plots be suppressed or not.
	Plots and returns the internal metacommunity structure of species and sites (see Leibold et al., 2022). Plots were heavily inspired by Leibold et al., 2022

Value

An object of class sjSDMinternalStructure consisting of a list of data.frames with the internal structure.

References

Leibold, M. A., Rudolph, F. J., Blanchet, F. G., De Meester, L., Gravel, D., Hartig, F., ... & Chase, J. M. (2022). The internal structure of metacommunities. Oikos, 2022(1).

See Also

plot.sjSDMinternalStructure, print.sjSDMinternalStructure

Examples

```
## Not run:
library(sjSDM)
# simulate community:
community = simulate_SDM(env = 3L, species = 10L, sites = 100L)

Occ <- community$response
Env <- community$env_weights
SP <- data.frame(matrix(rnorm(200, 0, 0.3), 100, 2)) # spatial coordinates


# fit model:
model <- sjSDM(Y = Occ,
               env = linear(data = Env, formula = ~X1+X2+X3),
               spatial = linear(data = SP, formula = ~0+X1*X2),
               family=binomial("probit"),
               verbose = FALSE,
               iter = 20) # increase iter for real analysis

# Calculate ANOVA for env, space, associations, for details see ?anova.sjSDM
an = anova(model, samples = 10, verbose = FALSE) # increase iter for real analysis

# Show anova fractions
plot(an)

# ANOVA tables with different way to handle fractions
summary(an)
summary(an, fractions = "discard")
summary(an, fractions = "proportional")
summary(an, fractions = "equal")

# Internal structure
int = internalStructure(an, fractions = "proportional")

print(int)

plot(int) # default is negative values will be set to 0
plot(int, negatives = "scale") # global rescaling of all values to range 0-1
plot(int, negatives = "raw") # negative values will be discarded

plotAssemblyEffects(int)
plotAssemblyEffects(int, negatives = "floor")
plotAssemblyEffects(int, response = "sites", pred = as.factor(c(rep(1, 50), rep(2, 50))))
plotAssemblyEffects(int, response = "species", pred = runif(10))
plotAssemblyEffects(int, response = "species", pred = as.factor(c(rep(1, 5), rep(2, 5))))

## End(Not run)
```

`is_torch_available` *is_torch_available*

Description

is_torch_available

Usage

```
is_torch_available()
```

Details

check whether torch is available

Value

Logical, is torch module available or not.

`linear` *Linear model of environmental response*

Description

specify the model to be fitted

Usage

```
linear(data = NULL, formula = NULL, lambda = 0, alpha = 0.5)
```

Arguments

`data`	matrix of environmental predictors
`formula`	formula object for predictors
`lambda`	lambda penalty, strength of regularization: $\lambda * (lasso + ridge)$
`alpha`	weighting between lasso and ridge: $(1-\alpha)*\|coefficients\|+\alpha\|\|coefficients\|\|^2$

Value

An S3 class of type 'linear' including the following components:

`formula`	Model matrix formula
`X`	Model matrix of covariates
`data`	Raw data
`l1_coef`	L1 regularization strength, can be -99 if `lambda = 0.0`
`l2_coef`	L2 regularization strength, can be -99 if `lambda = 0.0`

Implemented S3 methods include `print.linear`

See Also

DNN, sjSDM

Examples

```
## Not run:

# Basic workflow:
## simulate community:
com = simulate_SDM(env = 3L, species = 7L, sites = 100L)

## fit model:
model = sjSDM(Y = com$response,env = com$env_weights, iter = 50L,
              verbose = FALSE)
# increase iter for your own data

# Default distribution is binomial("probit"). Alternatively, you can use
# binomial(logit), poisson("log"), "nbinom" (with log, still somewhat
# experimental) and gaussian("identity")

coef(model)
summary(model)
getCov(model)

## plot results
species=c("sp1","sp2","sp3","sp4","sp5","sp6","sp7")
group=c("mammal","bird","fish","fish","mammal","amphibian","amphibian")
group = data.frame(species=species,group=group)
plot(model,group=group)

## calculate post-hoc p-values:
p = getSe(model)
summary(p)

## or turn on the option in the sjSDM function:
model = sjSDM(Y = com$response, env = com$env_weights, se = TRUE,
              family = binomial("probit"),
              iter = 2L,
              verbose = FALSE)
summary(model)

## fit model with interactions:
model = sjSDM(Y = com$response,
              env = linear(data = com$env_weights, formula = ~X1:X2 + X3),
              se = TRUE,
              iter = 2L,
              verbose = FALSE) # increase iter for your own data
summary(model)

## without intercept:
model = update(model, env_formula = ~0+X1:X2 + X3,
               verbose = FALSE)
```

```
summary(model)

## predict with model:
preds = predict(model, newdata = com$env_weights)

## calculate R-squared:
R2 = Rsquared(model)
print(R2)

# With spatial terms:
## linear spatial model
XY = matrix(rnorm(200), 100, 2)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(XY, ~0+X1:X2),
              iter = 50L,
              verbose = FALSE) # increase iter for your own data
summary(model)
predict(model, newdata = com$env_weights, SP = XY)
R2 = Rsquared(model)
print(R2)

## Using spatial eigenvectors as predictors to account
## for spatial autocorrelation is a common approach:
SPV = generateSpatialEV(XY)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(SPV, ~0+., lambda = 0.1),
              iter = 50L,
              verbose = FALSE) # increase iter for your own data
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)

## Visualize internal meta-community structure
an = anova(model,
           verbose = FALSE)

internal = internalStructure(an)
plot(internal)

## Visualize community assemlby effects

plotAssemblyEffects(internal)

### see ?anova.sjSDM for mroe details


## non-linear(deep neural network) model
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = DNN(SPV,hidden = c(5L, 5L), ~0+.),
              iter = 2L,# increase iter for your own data
              verbose = FALSE)
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)
```

```
# Regularization
## lambda is the regularization strength
## alpha weights the lasso or ridge penalty:
## - alpha = 0 --> pure lasso
## - alpha = 1.0 --> pure ridge
model = sjSDM(Y = com$response,
              # mix of lasso and ridge
              env = linear(com$env_weights, lambda = 0.01, alpha = 0.5),
              # we can do the same for the species-species associations
              biotic = bioticStruct(lambda = 0.01, alpha = 0.5),
              iter = 2L,# increase iter for your own data
              verbose = FALSE)
summary(model)
coef(model)
getCov(model)



# Anova
com = simulate_SDM(env = 3L, species = 15L, sites = 200L, correlation = TRUE)

XY = matrix(rnorm(400), 200, 2)
SPV = generateSpatialEV(XY)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(SPV, ~0+.),
              verbose = FALSE,
              iter = 50L) # increase iter for your own data
result = anova(model, verbose = FALSE)
print(result)
plot(result)

## visualize internal meta-community structure
internal = internalStructure(an)
plot(internal)



# Deep neural networks
## we can fit also a deep neural network instead of a linear model:
model = sjSDM(Y = com$response,
              env = DNN(com$env_weights, hidden = c(10L, 10L, 10L)),
              verbose = FALSE,
              iter = 2L) # increase iter for your own data
summary(model)
getCov(model)
pred = predict(model, newdata = com$env_weights)

## extract weights
weights = getWeights(model)

## we can also assign weights:
```

```
setWeights(model, weights)

## with regularization:
model = sjSDM(Y = com$response,
              # mix of lasso and ridge
              env = DNN(com$env_weights, lambda = 0.01, alpha = 0.5),
              # we can do the same for the species-species associations
              biotic = bioticStruct(lambda = 0.01, alpha = 0.5),
              verbose = FALSE,
              iter = 2L) # increase iter for your own data
getCov(model)
getWeights(model)

## End(Not run)
```

`logLik.sjSDM` *Extract negative-log-Likelihood from a fitted sjSDM model*

Description

Extract negative-log-Likelihood from a fitted sjSDM model

Usage

```
## S3 method for class 'sjSDM'
logLik(object, individual = FALSE, ...)
```

Arguments

`object`	a model fitted by `sjSDM`
`individual`	returns internal ll structure, mostly for internal useage
`...`	optional arguments passed to internal logLik function (only used if `individual=TRUE`)

Value

Numeric value or numeric matrix if individual is true.

`madgrad` *madgrad*

Description

stochastic gradient descent optimizer

Usage

```
madgrad(momentum = 0.9, weight_decay = 0, eps = 1e-06)
```

Arguments

momentum	strength of momentum
weight_decay	l2 penalty on weights
eps	epsilon

Value

Anonymous function that returns optimizer when called.

References

Defazio, A., & Jelassi, S. (2021). Adaptivity without Compromise: A Momentumized, Adaptive, Dual Averaged Gradient Method for Stochastic Optimization. arXiv preprint arXiv:2101.11075.

---

new_image *new_image function*

Description

new_image function

Usage

```
new_image(
  z,
  cols = (grDevices::colorRampPalette(c("white", "#24526E"), bias = 1.5))(10),
  range = c(0.5, 1)
)
```

Arguments

z	z matrix
cols	cols for gradient
range	rescale to range

plot.sjSDM *Coefficients plot*

Description

Plotting coefficients returned by sjSDM model. This function only for model fitted by linear, fitted by DNN is not yet supported.

Usage

```
## S3 method for class 'sjSDM'
plot(x, ...)
```

Arguments

x	a model fitted by sjSDM
...	Additional arguments to pass to plotsjSDMcoef.

Value

ggplot2 object for linear sjSDM model and nothing for DNN sjSDM model.

Author(s)

CAI Wang

See Also

plotsjSDMcoef

Examples

```
## Not run:
library(sjSDM)
# simulate community:
com = simulate_SDM(env = 6L, species = 7L, sites = 100L)

# fit model:
model = sjSDM(Y = com$response,env = com$env_weights, iter = 2L, se = TRUE,
              verbose = FALSE)

# normal plot
plot(model)

# colored by groups
species=c("sp1","sp2","sp3","sp4","sp5","sp6","sp7")
group=c("mammal","bird","fish","fish","mammal","amphibian","amphibian")
group = data.frame(species=species,group=group)
```

```
plot(model,group=group)

## End(Not run)
```

`plot.sjSDM.DNN` *Training history*

Description

Plot training loss history

Usage

```
## S3 method for class 'sjSDM.DNN'
plot(x, ...)
```

Arguments

x	a model fitted by `sjSDM` with `DNN` object
...	passed to plot

Value

No return value, called for side effects.

Examples

```
## Not run:
library(sjSDM)
# simulate community:
com = simulate_SDM(env = 6L, species = 7L, sites = 100L)

# fit model:
model = sjSDM(Y = com$response,env = com$env_weights, iter = 2L, se = TRUE,
              verbose = FALSE)

# normal plot
plot(model)

# colored by groups
species=c("sp1","sp2","sp3","sp4","sp5","sp6","sp7")
group=c("mammal","bird","fish","fish","mammal","amphibian","amphibian")
group = data.frame(species=species,group=group)

plot(model,group=group)

## End(Not run)
```

`plot.sjSDManova` *Plot anova results*

Description

Plot anova results

Usage

```
## S3 method for class 'sjSDManova'
plot(
  x,
  y,
  type = c("McFadden", "Deviance", "Nagelkerke"),
  fractions = c("discard", "proportional", "equal"),
  cols = c("#7FC97F", "#BEAED4", "#FDC086"),
  alpha = 0.15,
  env_deviance = NULL,
  ...
)
```

Arguments

`x`	anova object from `anova.sjSDM`
`y`	unused argument
`type`	deviance, Nagelkerke or McFadden R-squared
`fractions`	how to handle shared fractions
`cols`	colors for the groups
`alpha`	alpha for colors
`env_deviance`	environmental deviance
`...`	Additional arguments to pass to `plot()`

Value

List with the following components:

`VENN`	Matrix of shown results.

References

Leibold, M. A., Rudolph, F. J., Blanchet, F. G., De Meester, L., Gravel, D., Hartig, F., ... & Chase, J. M. (2022). The internal structure of metacommunities. Oikos, 2022(1).

plot.sjSDMimportance *Plot importance*

Description

Plot importance

Usage

```
## S3 method for class 'sjSDMimportance'
plot(x, y, col.points = "#24526e", cex.points = 1.2, ...)
```

Arguments

x	a model fitted by importance
y	unused argument
col.points	point color
cex.points	point size
...	Additional arguments to pass to plot()

Value

The visualized matrix is silently returned.

plot.sjSDMinternalStructure *Plot internal structure*

Description

Creates a ternary diagram of an object of class

Usage

```
## S3 method for class 'sjSDMinternalStructure'
plot(
  x,
  alpha = 0.15,
  env_deviance = NULL,
  negatives = c("floor", "scale", "raw"),
  ...
)
```

Arguments

x	and object of class sjSDMinternalStructure create by anova object from internalStructure
alpha	alpha of points
env_deviance	environmental deviance/gradient (points will be colored)
negatives	how to handle negative R squareds
...	no function

Examples

```
## Not run:
library(sjSDM)
# simulate community:
community = simulate_SDM(env = 3L, species = 10L, sites = 100L)

Occ <- community$response
Env <- community$env_weights
SP <- data.frame(matrix(rnorm(200, 0, 0.3), 100, 2)) # spatial coordinates


# fit model:
model <- sjSDM(Y = Occ,
               env = linear(data = Env, formula = ~X1+X2+X3),
               spatial = linear(data = SP, formula = ~0+X1*X2),
               family=binomial("probit"),
               verbose = FALSE,
               iter = 20) # increase iter for real analysis

# Calculate ANOVA for env, space, associations, for details see ?anova.sjSDM
an = anova(model, samples = 10, verbose = FALSE) # increase iter for real analysis

# Show anova fractions
plot(an)

# ANOVA tables with different way to handle fractions
summary(an)
summary(an, fractions = "discard")
summary(an, fractions = "proportional")
summary(an, fractions = "equal")

# Internal structure
int = internalStructure(an, fractions = "proportional")

print(int)

plot(int) # default is negative values will be set to 0
plot(int, negatives = "scale") # global rescaling of all values to range 0-1
plot(int, negatives = "raw") # negative values will be discarded

plotAssemblyEffects(int)
plotAssemblyEffects(int, negatives = "floor")
```

```
plotAssemblyEffects(int, response = "sites", pred = as.factor(c(rep(1, 50), rep(2, 50))))
plotAssemblyEffects(int, response = "species", pred = runif(10))
plotAssemblyEffects(int, response = "species", pred = as.factor(c(rep(1, 5), rep(2, 5))))

## End(Not run)
```

`plot.sjSDM_cv` *Plot elastic net tuning*

Description

Plot elastic net tuning

Usage

```
## S3 method for class 'sjSDM_cv'
plot(x, y, perf = c("logLik", "AUC", "AUC_macro"), resolution = 6, k = 3, ...)
```

Arguments

`x`	a model fitted by `sjSDM_cv`
`y`	unused argument
`perf`	performance measurement to plot
`resolution`	resolution of grid
`k`	number of knots for the gm
`...`	Additional arguments to pass to `plot()`

Value

Named vector of optimized regularization parameters.

Without space:

`lambda_cov`	Regularization strength in the `bioticStruct` object.
`alpha_cov`	Weigthing between L1 and L2 in the `bioticStruct` object.
`lambda_coef`	Regularization strength in the `linear` or `DNN` object.
`alpha_coef`	Weigthing between L1 and L2 in the `linear` or `DNN` object.

With space:

`lambda_cov`	Regularization strength in the `bioticStruct` object.
`alpha_cov`	Weigthing between L1 and L2 in the `bioticStruct` object.
`lambda_coef`	Regularization strength in the `linear` or `DNN` object.
`alpha_coef`	Weigthing between L1 and L2 in the `linear` or `DNN` object.
`lambda_spatial`	Regularization strength in the `linear` or `DNN` object for the spatial component.
`alpha_spatial`	Weigthing between L1 and L2 in the`linear` or `DNN` object for the spatial component.

plotAssemblyEffects *Plot predictors of assembly processes*

Description

The function plots correlations between assembly processes and predictors or traits

Usage

```
plotAssemblyEffects(
  object,
  response = c("sites", "species"),
  pred = NULL,
  cols = c("#A38310", "#B42398", "#20A382"),
  negatives = c("raw", "scale", "floor")
)
```

Arguments

object	An sjSDManova object from the anova.sjSDM function.
response	whether to use sites or species. Default is sites
pred	predictor variable. If NULL, environment uniqueness, spatial uniqueness, and richness is calculated from the fitted object and used as predictor.
cols	Colors for the three assembly processes.
negatives	how to handle negative R squareds

Details

Correlation and plots of the three assembly processes (environment, space, and codist) against environmental and spatial uniqueness and richness. The importance of the three assembly processes is measured by the partial R-squared (shown in the internal structure plots).

Importances are available for species and sites. Custom environmental predictors or traits can be specified. Environmental predictors are plotted against site R-squared and traits are plotted against species R-squared. Regression lines are estimated by 50\

Value

A list with the following components:

env	A list of summary tables for env, space, and codist R-squared.
space	A list of summary tables for env, space, and codist R-squared.
codist	A list of summary tables for env, space, and codist R-squared.

Note

Defaults for negative values are different than for plot.sjSDMinternalStructure

References

Leibold, M. A., Rudolph, F. J., Blanchet, F. G., De Meester, L., Gravel, D., Hartig, F., ... & Chase, J. M. (2022). The internal structure of metacommunities. *Oikos*, 2022(1).

Examples

```
## Not run:
library(sjSDM)
# simulate community:
community = simulate_SDM(env = 3L, species = 10L, sites = 100L)

Occ <- community$response
Env <- community$env_weights
SP <- data.frame(matrix(rnorm(200, 0, 0.3), 100, 2)) # spatial coordinates


# fit model:
model <- sjSDM(Y = Occ,
               env = linear(data = Env, formula = ~X1+X2+X3),
               spatial = linear(data = SP, formula = ~0+X1*X2),
               family=binomial("probit"),
               verbose = FALSE,
               iter = 20) # increase iter for real analysis

# Calculate ANOVA for env, space, associations, for details see ?anova.sjSDM
an = anova(model, samples = 10, verbose = FALSE) # increase iter for real analysis

# Show anova fractions
plot(an)

# ANOVA tables with different way to handle fractions
summary(an)
summary(an, fractions = "discard")
summary(an, fractions = "proportional")
summary(an, fractions = "equal")

# Internal structure
int = internalStructure(an, fractions = "proportional")

print(int)

plot(int) # default is negative values will be set to 0
plot(int, negatives = "scale") # global rescaling of all values to range 0-1
plot(int, negatives = "raw") # negative values will be discarded

plotAssemblyEffects(int)
plotAssemblyEffects(int, negatives = "floor")
plotAssemblyEffects(int, response = "sites", pred = as.factor(c(rep(1, 50), rep(2, 50))))
plotAssemblyEffects(int, response = "species", pred = runif(10))
plotAssemblyEffects(int, response = "species", pred = as.factor(c(rep(1, 5), rep(2, 5))))

## End(Not run)
```

plotsjSDMcoef *Internal coefficients plot*

Description

Plotting coefficients returned by sjSDM model. This function only for model fitted by linear, fitted by DNN is not yet supported.

Usage

```
plotsjSDMcoef(object, wrap_col = NULL, group = NULL, col = NULL, slist = NULL)
```

Arguments

object	a model fitted by sjSDM
wrap_col	Scales argument passed to wrap_col
group	Define the taxonomic characteristics of a species, you need to provide a dataframe with column1 named "species" and column2 named "group", default is NULL. For example, group[1,1]== "sp1", group[1,2]== "Mammal".
col	Define colors for groups, default is NULL.
slist	Select the species you want to plot, default is all, parameter is not supported yet.

Value

ggplot2 object

Author(s)

CAI Wang

Examples

```
## Not run:
library(sjSDM)
# simulate community:
com = simulate_SDM(env = 6L, species = 7L, sites = 100L)

# fit model:
model = sjSDM(Y = com$response,env = com$env_weights, iter = 2L, se = TRUE,
              verbose = FALSE)

# normal plot
plot(model)

# colored by groups
species=c("sp1","sp2","sp3","sp4","sp5","sp6","sp7")
group=c("mammal","bird","fish","fish","mammal","amphibian","amphibian")
group = data.frame(species=species,group=group)
```

```
plot(model,group=group)

## End(Not run)
```

---

predict.sjSDM *Predict from a fitted sjSDM model*

Description

Predict from a fitted sjSDM model

Usage

```
## S3 method for class 'sjSDM'
predict(
  object,
  newdata = NULL,
  SP = NULL,
  Y = NULL,
  type = c("link", "raw"),
  dropout = FALSE,
  ...
)
```

Arguments

object	a model fitted by sjSDM
newdata	newdata for predictions
SP	spatial predictors (e.g. X and Y coordinates)
Y	Known occurrences of species, must be a matrix of the original size, species to be predicted must consist of NAs
type	raw or link
dropout	use dropout for predictions or not, only supported for DNNs
...	optional arguments for compatibility with the generic function, no function implemented

Value

Matrix of predictions (sites by species)

Examples

```
## Not run:

## Conditional predictions based on focal species
com = simulate_SDM(sites = 200L)
## first 100 observations are the training data
model = sjSDM(com$response[1:100, ], com$env_weights[1:100,])
## Assume that for the other 100 observations, only the first species is missing
## and we want to use the other 4 species to improve the predictions:
Y_focal = com$response[101:200, ]
Y_focal[,1] = NA # set to NA because occurrences are unknown

pred_conditional = predict(model, newdata = com$env_weights[101:200,], Y = Y_focal)
pred_unconditional = predict(model, newdata = com$env_weights[101:200,])[,1]

## Compare performance:
Metrics::auc(com$response[101:200, 1], pred_conditional)
Metrics::auc(com$response[101:200, 1], pred_unconditional)

## Conditional predictions are better, however, it only works if occurrences of
## other species for new sites are known!


## End(Not run)
```

---

print.bioticStruct *Print a bioticStruct object*

Description

Print a bioticStruct object

Usage

```
## S3 method for class 'bioticStruct'
print(x, ...)
```

Arguments

x	object created by bioticStruct
...	optional arguments for compatibility with the generic function, no function implemented

print.DNN *Print a DNN object*

Description

Print a DNN object

Usage

```
## S3 method for class 'DNN'
print(x, ...)
```

Arguments

x	object created by DNN
...	optional arguments for compatibility with the generic function, no function implemented

print.linear *Print a linear object*

Description

Print a linear object

Usage

```
## S3 method for class 'linear'
print(x, ...)
```

Arguments

x	object created by linear
...	optional arguments for compatibility with the generic function, no function implemented

Value

Invisible formula object

print.sjSDM *Print a fitted sjSDM model*

Description

Print a fitted sjSDM model

Usage

```
## S3 method for class 'sjSDM'
print(x, ...)
```

Arguments

x	a model fitted by sjSDM
...	optional arguments for compatibility with the generic function, no function implemented

Value

No return value

print.sjSDManova *Print sjSDM anova object*

Description

This is a wrapper for summary.sjSDManova, maintained for backwards compatibility - prefer to use summary() instead

Usage

```
## S3 method for class 'sjSDManova'
print(x, ...)
```

Arguments

x	an object of type sjSDManova created by anova.sjSDM
...	additional arguments to summary.sjSDManova

Examples

```
## Not run:
library(sjSDM)
# simulate community:
community = simulate_SDM(env = 3L, species = 10L, sites = 100L)

Occ <- community$response
Env <- community$env_weights
SP <- data.frame(matrix(rnorm(200, 0, 0.3), 100, 2)) # spatial coordinates


# fit model:
model <- sjSDM(Y = Occ,
               env = linear(data = Env, formula = ~X1+X2+X3),
               spatial = linear(data = SP, formula = ~0+X1*X2),
               family=binomial("probit"),
               verbose = FALSE,
               iter = 20) # increase iter for real analysis

# Calculate ANOVA for env, space, associations, for details see ?anova.sjSDM
an = anova(model, samples = 10, verbose = FALSE) # increase iter for real analysis

# Show anova fractions
plot(an)

# ANOVA tables with different way to handle fractions
summary(an)
summary(an, fractions = "discard")
summary(an, fractions = "proportional")
summary(an, fractions = "equal")

# Internal structure
int = internalStructure(an, fractions = "proportional")

print(int)

plot(int) # default is negative values will be set to 0
plot(int, negatives = "scale") # global rescaling of all values to range 0-1
plot(int, negatives = "raw") # negative values will be discarded

plotAssemblyEffects(int)
plotAssemblyEffects(int, negatives = "floor")
plotAssemblyEffects(int, response = "sites", pred = as.factor(c(rep(1, 50), rep(2, 50))))
plotAssemblyEffects(int, response = "species", pred = runif(10))
plotAssemblyEffects(int, response = "species", pred = as.factor(c(rep(1, 5), rep(2, 5))))

## End(Not run)
```

`print.sjSDMimportance` *Print importance*

Description

Print importance

Usage

```
## S3 method for class 'sjSDMimportance'
print(x, ...)
```

Arguments

x	an object of importance
...	optional arguments for compatibility with the generic function, no function implemented

Value

The matrix above is silently returned

---

print.sjSDMinternalStructure
Print internal structure object

Description

Print internal structure object

Usage

```
## S3 method for class 'sjSDMinternalStructure'
print(x, ...)
```

Arguments

x	object of class sjSDMinternalStructure
...	no function

print.sjSDM_cv *Print a fitted sjSDM_cv model*

Description

Print a fitted sjSDM_cv model

Usage

```
## S3 method for class 'sjSDM_cv'
print(x, ...)
```

Arguments

x	a model fitted by sjSDM_cv
...	optional arguments for compatibility with the generic function, no function implemented

Value

Above data frame is silently returned.

residuals.sjSDM *Residuals for a sjSDM model*

Description

Returns residuals for a fitted sjSDM model

Usage

```
## S3 method for class 'sjSDM'
residuals(object, type = "raw", ...)
```

Arguments

object	a model fitted by sjSDM
type	residual type. Currently only supports raw
...	further arguments, not supported yet.

Value

residuals in the format of the provided community matrix

RMSprop *RMSprop*

Description

RMSprop optimizer

Usage

```
RMSprop(
  alpha = 0.99,
  eps = 1e-08,
  weight_decay = 1e-04,
  momentum = 0.1,
  centered = FALSE
)
```

Arguments

alpha	decay factor
eps	fuzz factor
weight_decay	l2 penalty on weights
momentum	momentum
centered	centered or not

Value

Anonymous function that returns optimizer when called.

Rsquared *R-squared*

Description

calculate R-squared following McFadden or Nagelkerke

Usage

```
Rsquared(model, method = c("McFadden", "Nagelkerke"), verbose = TRUE)
```

Arguments

model	model
method	McFadden or Nagelkerke
verbose	TRUE or FALSE, indicating whether progress should be printed or not

Details

Calculate R-squared following Nagelkerke or McFadden:

- Nagelkerke: $R^2 = 1 - \exp(2/N \cdot (log\mathcal{L}_0 - log\mathcal{L}_1))$
- McFadden: $R^2 = 1 - log\mathcal{L}_1/log\mathcal{L}_0$

Value

R-squared as numeric value

Author(s)

Maximilian Pichler

`setWeights` *Set weights*

Description

set layer weights and sigma in `sjSDM` with `DNN` object

Usage

```
setWeights(object, weights)

## S3 method for class 'sjSDM'
setWeights(object, weights = NULL)
```

Arguments

`object`	object of class `sjSDM` with `DNN` object
`weights`	list of layer weights: `list(env=list(matrix(...)), spatial=list(matrix(...)), sigma=matrix(...))`, see `getWeights`

Value

No return value, weights are changed in place.

SGD *SGD*

Description

stochastic gradient descent optimizer

Usage

```
SGD(momentum = 0.5, dampening = 0, weight_decay = 0, nesterov = TRUE)
```

Arguments

momentum	strength of momentum
dampening	decay
weight_decay	l2 penalty on weights
nesterov	Nesterov momentum or not

Value

Anonymous function that returns optimizer when called.

simulate.sjSDM *Generates simulations from sjSDM model*

Description

Simulate nsim responses from the fitted model following a multivariate probit model. So currently only supported for `family = stats::binomial("probit")`

Usage

```
## S3 method for class 'sjSDM'
simulate(object, nsim = 1, seed = NULL, ...)
```

Arguments

object	a model fitted by sjSDM
nsim	number of simulations
seed	seed for random number generator
...	optional arguments for compatibility with the generic function, no functionality implemented

Value

Array of simulated species occurrences of dimension order (nsim, sites, species)

simulate_SDM *Simulate joint Species Distribution Models*

Description

Simulate species distributions

Usage

```
simulate_SDM(
  env = 5L,
  sites = 100L,
  species = 5L,
  correlation = TRUE,
  weight_range = c(-1, 1),
  link = "probit",
  response = "pa",
  sparse = NULL,
  tolerance = 0.05,
  iter = 20L,
  seed = NULL
)
```

Arguments

env	number of environment variables
sites	number of sites
species	number of species
correlation	correlated species TRUE or FALSE, can be also a function or a matrix
weight_range	sample true weights from uniform range, default -1,1
link	probit, logit or identical
response	pa (presence-absence) or count
sparse	sparse rate
tolerance	tolerance for sparsity check
iter	tries until sparse rate is achieved
seed	random seed. Default = 42

Details

Probit is not possible for abundance response (response = 'count')

Value

List of simulation results:

`env`	Number of environmental covariates
`species`	Number of species
`sites`	Number of sites
`link`	Which link
`response_type`	Which response type
`response`	Species occurrence matrix
`correlation`	Species covariance matrix
`species_weights`	Species-environment coefficients
`env_weights`	Environmental covariates
`corr_acc`	Method to calculate sign accurracy

Author(s)

Maximilian Pichler

`sjSDM` *Fitting scalable joint Species Distribution Models (sjSDM)*

Description

`sjSDM` is used to fit joint Species Distribution models (jSDMs) using the central processing unit (CPU) or the graphical processing unit (GPU). The default is a multivariate probit model based on a Monte-Carlo approximation of the joint likelihood. `sjSDM` can be used to fit linear but also deep neural networks and supports the well known formula syntax.

Usage

```
sjSDM(
  Y = NULL,
  env = NULL,
  biotic = bioticStruct(),
  spatial = NULL,
  family = stats::binomial("probit"),
  iter = 100L,
  step_size = NULL,
  learning_rate = 0.01,
  se = FALSE,
  sampling = 100L,
  parallel = 0L,
  control = sjSDMControl(),
```

```
  device = "cpu",
  dtype = "float32",
  seed = 758341678,
  verbose = TRUE
)

sjSDM.tune(object)
```

Arguments

Y	matrix of species occurrences/responses in range
env	matrix of environmental predictors, object of type linear or DNN
biotic	defines biotic (species-species associations) structure, object of type bioticStruct
spatial	defines spatial structure, object of type linear or DNN
family	error distribution with link function, see details for supported distributions
iter	number of fitting iterations
step_size	batch size for stochastic gradient descent, if NULL then step_size is set to: step_size = 0.1*nrow(X)
learning_rate	learning rate for Adamax optimizer
se	calculate standard errors for environmental coefficients
sampling	number of sampling steps for Monte Carlo integration
parallel	number of cpu cores for the data loader, only necessary for large datasets
control	control parameters for optimizer, see sjSDMControl
device	which device to be used, "cpu" or "gpu"
dtype	which data type, most GPUs support only 32 bit floats.
seed	seed for random operations
verbose	TRUE or FALSE, indicating whether progress should be printed or not
object	object of type sjSDM_cv

Details

The function fits per default a multivariate probit model via Monte-Carlo integration (see Chen et al., 2018) of the joint likelihood for all species.

Model description:

The most common jSDM structure describes the site ($i = 1, ..., I$) by species ($j = 1, ..., J$) matrix Y_{ij} as a function of environmental covariates $X_{in} (n = 1, ..., N$ covariates), and the species-species covariance matrix Σ accounts for correlations in e_{ij}:

$$g(Z_{ij}) = \beta_{j0} + \Sigma_{n=1}^{N} X_{in} \beta_{nj} + e_{ij}$$

with $g(.)$ as link function. For the multivariate probit model, the link function is:

$$Y_{ij} = 1(Z_{ij} > 0)$$

The probability to observe the occurrence vector $\mathbf{Y_i}$ is:

$$Pr(\mathbf{Y_i}|\mathbf{X_i}\beta, \mathbf{\Sigma}) = \int_{\mathbf{A_{iJ}}} ... \int_{\mathbf{A_{i1}}} \phi_{\mathbf{J}}(\mathbf{Y_i^*}; \mathbf{X_i}\beta, \mathbf{\Sigma}) \mathbf{dY_{i1}^*} ... \mathbf{dY_{iJ}^*}$$

in the interval A_{ij} with $(-\inf, 0]$ if $Y_{ij} = 0$ and $[0, +\inf)$ if $Y_{ij} = 1$.

and ϕ being the density function of the multivariate normal distribution.

The probability of $\mathbf{Y_i}$ requires to integrate over $\mathbf{Y_i^*}$ which has no closed analytical expression for more than two species which makes the evaluation of the likelihood computationally costly and needs a numerical approximation. The previous equation can be expressed more generally as:

$$\mathcal{L}(\beta, \Sigma; \mathbf{Y_i}, \mathbf{X_i}) = \int_{\mathbf{\Omega}} \prod_{\mathbf{j=1}}^{\mathbf{J}} \mathbf{Pr}(\mathbf{Y_{ij}}|\mathbf{X_i}\beta + \zeta) \mathbf{Pr}(\zeta|\mathbf{\Sigma}) \mathbf{d}\zeta$$

`sjSDM` approximates this integral by M Monte-Carlo samples from the multivariate normal species-species covariance. After integrating out the covariance term, the remaining part of the likelihood can be calculated as in an univariate case and the average of the M samples are used to get an approximation of the integral:

$$\mathcal{L}(\beta, \Sigma; \mathbf{Y_i}, \mathbf{X_i}) \approx \frac{\mathbf{1}}{\mathbf{M}} \mathbf{\Sigma_{m=1}^{M}} \prod_{\mathbf{j=1}}^{\mathbf{J}} \mathbf{Pr}(\mathbf{Y_{ij}}|\mathbf{X_i}\beta + \zeta_{\mathbf{m}})$$

with $\zeta_m \sim MVN(0, \Sigma)$.

`sjSDM` uses 'PyTorch' to run optionally the model on the graphical processing unit (GPU). Python dependencies needs to be installed before being able to use the `sjSDM` function. We provide a function which installs automatically python and the python dependencies. See `install_sjSDM`, `vignette("Dependencies", package = "sjSDM")`

See Pichler and Hartig, 2020 for benchmark results.

Supported distributions:

Currently supported distributions and link functions, which are :

- `binomial`: `"probit"` or `"logit"`
- `poisson`: `"log"`
- `"nbinom"`: `"log"`
- `gaussian`: `"identity"`

Space:

We can extend the model to account for spatial auto-correlation between the sites by:

$$g(Z_{ij}) = \beta_{j0} + \Sigma_{n=1}^{N} X_{in}\beta_{nj} + \Sigma_{m=1}^{M} S_{im}\alpha_{mj} + e_{ij}$$

There are two ways to generate spatial predictors S:

- trend surface model - using spatial coordinates in a polynomial: `linear(data=Coords, ~0+poly(X, Y, degree = 2))`

- eigenvector spatial filtering - using spatial eigenvectors. Spatial eigenvectors can be generated by the generateSpatialEV function:
  `SPV = generateSpatialEV(Coords)`
  Then we use, for example, the first 20 spatial eigenvectors:
  `linear(data=SPV[ ,1:20], ~0+.)`

It is important to set the intercept to 0 in the spatial term (e.g. via `~0+.`) because the intercept is already set in the environmental object.

Installation:

`install_sjSDM` should be theoretically able to install conda and 'PyTorch' automatically. If `sjSDM` still does not work after reloading RStudio, you can try to solve this on your following our trouble shooting guide `installation_help`. If the problem remains, please create an issue on issue tracker with a copy of the `install_diagnostic` output as a quote.

Value

An S3 class of type 'sjSDM' including the following components:

`cl`	Model call
`formula`	Formula object for environmental covariates.
`names`	Names of environmental covariates.
`species`	Names of species (can be `NULL` if columns of Y are not named).
`get_model`	Method which builds and returns the underlying 'python' model.
`logLik`	negative log-Likelihood of the model and the regularization loss.
`model`	The actual model.
`settings`	List of model settings, see arguments of `sjSDM`.
`family`	Response family.
`time`	Runtime.
`data`	List of Y, X (and spatial) model matrices.
`sessionInfo`	Output of `sessionInfo`.
`weights`	List of model coefficients (environmental (and spatial)).
`sigma`	Lower triangular weight matrix for the covariance matrix.
`history`	History of iteration losses.
`se`	Matrix of standard errors, if `se = FALSE` the field 'se' is `NULL`.

Implemented S3 methods include `summary.sjSDM`, `plot.sjSDM`, `print.sjSDM`, `predict.sjSDM`, and `coef.sjSDM`. For other methods, see section 'See Also'.

`sjSDM.tune` returns an S3 object of class 'sjSDM', see above for information about values.

Author(s)

Maximilian Pichler

References

Chen, D., Xue, Y., & Gomes, C. P. (2018). End-to-end learning for the deep multivariate probit model. arXiv preprint arXiv:1803.08591.

Pichler, M., & Hartig, F. (2021). A new joint species distribution model for faster and more accurate inference of species associations from big community data. Methods in Ecology and Evolution, 12(11), 2159-2173.

See Also

getCor, getCov, update.sjSDM, sjSDM_cv, DNN, plot.sjSDM, print.sjSDM, predict.sjSDM, coef.sjSDM, summary.sjSDM, simulate.sjSDM, getSe, anova.sjSDM, importance

Examples

```
## Not run:

# Basic workflow:
## simulate community:
com = simulate_SDM(env = 3L, species = 7L, sites = 100L)

## fit model:
model = sjSDM(Y = com$response,env = com$env_weights, iter = 50L,
              verbose = FALSE)
# increase iter for your own data

# Default distribution is binomial("probit"). Alternatively, you can use
# binomial(logit), poisson("log"), "nbinom" (with log, still somewhat
# experimental) and gaussian("identity")

coef(model)
summary(model)
getCov(model)

## plot results
species=c("sp1","sp2","sp3","sp4","sp5","sp6","sp7")
group=c("mammal","bird","fish","fish","mammal","amphibian","amphibian")
group = data.frame(species=species,group=group)
plot(model,group=group)

## calculate post-hoc p-values:
p = getSe(model)
summary(p)

## or turn on the option in the sjSDM function:
model = sjSDM(Y = com$response, env = com$env_weights, se = TRUE,
              family = binomial("probit"),
              iter = 2L,
              verbose = FALSE)
summary(model)

## fit model with interactions:
```

```
model = sjSDM(Y = com$response,
              env = linear(data = com$env_weights, formula = ~X1:X2 + X3),
              se = TRUE,
              iter = 2L,
              verbose = FALSE) # increase iter for your own data
summary(model)

## without intercept:
model = update(model, env_formula = ~0+X1:X2 + X3,
               verbose = FALSE)

summary(model)

## predict with model:
preds = predict(model, newdata = com$env_weights)

## calculate R-squared:
R2 = Rsquared(model)
print(R2)

# With spatial terms:
## linear spatial model
XY = matrix(rnorm(200), 100, 2)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(XY, ~0+X1:X2),
              iter = 50L,
              verbose = FALSE) # increase iter for your own data
summary(model)
predict(model, newdata = com$env_weights, SP = XY)
R2 = Rsquared(model)
print(R2)

## Using spatial eigenvectors as predictors to account
## for spatial autocorrelation is a common approach:
SPV = generateSpatialEV(XY)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(SPV, ~0+., lambda = 0.1),
              iter = 50L,
              verbose = FALSE) # increase iter for your own data
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)

## Visualize internal meta-community structure
an = anova(model,
           verbose = FALSE)

internal = internalStructure(an)
plot(internal)

## Visualize community assemlby effects

plotAssemblyEffects(internal)
```

```
### see ?anova.sjSDM for mroe details


## non-linear(deep neural network) model
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = DNN(SPV,hidden = c(5L, 5L), ~0+.),
              iter = 2L,# increase iter for your own data
              verbose = FALSE)
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)


# Regularization
## lambda is the regularization strength
## alpha weights the lasso or ridge penalty:
## - alpha = 0 --> pure lasso
## - alpha = 1.0 --> pure ridge
model = sjSDM(Y = com$response,
              # mix of lasso and ridge
              env = linear(com$env_weights, lambda = 0.01, alpha = 0.5),
              # we can do the same for the species-species associations
              biotic = bioticStruct(lambda = 0.01, alpha = 0.5),
              iter = 2L,# increase iter for your own data
              verbose = FALSE)
summary(model)
coef(model)
getCov(model)



# Anova
com = simulate_SDM(env = 3L, species = 15L, sites = 200L, correlation = TRUE)

XY = matrix(rnorm(400), 200, 2)
SPV = generateSpatialEV(XY)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              spatial = linear(SPV, ~0+.),
              verbose = FALSE,
              iter = 50L) # increase iter for your own data
result = anova(model, verbose = FALSE)
print(result)
plot(result)

## visualize internal meta-community structure
internal = internalStructure(an)
plot(internal)



# Deep neural networks
## we can fit also a deep neural network instead of a linear model:
model = sjSDM(Y = com$response,
              env = DNN(com$env_weights, hidden = c(10L, 10L, 10L)),
```

```
              verbose = FALSE,
              iter = 2L) # increase iter for your own data
summary(model)
getCov(model)
pred = predict(model, newdata = com$env_weights)

## extract weights
weights = getWeights(model)

## we can also assign weights:
setWeights(model, weights)

## with regularization:
model = sjSDM(Y = com$response,
              # mix of lasso and ridge
              env = DNN(com$env_weights, lambda = 0.01, alpha = 0.5),
              # we can do the same for the species-species associations
              biotic = bioticStruct(lambda = 0.01, alpha = 0.5),
              verbose = FALSE,
              iter = 2L) # increase iter for your own data
getCov(model)
getWeights(model)

## End(Not run)
```

`sjSDMControl` *sjSDM control object*

Description

sjSDM control object

Usage

```
sjSDMControl(
  optimizer = RMSprop(),
  scheduler = 0,
  lr_reduce_factor = 0.99,
  early_stopping_training = 0,
  mixed = FALSE
)
```

Arguments

`optimizer`	object of type RMSprop, Adamax, SGD, AccSGD, madgrad, or AdaBound
`scheduler`	reduce lr on plateau scheduler or not (0 means no scheduler, > 0 number of epochs before reducing learning rate)
`lr_reduce_factor`	factor to reduce learning rate in scheduler

`early_stopping_training`
: number of epochs without decrease in training loss before invoking early stopping (0 means no early stopping).

`mixed`
: mixed (half-precision) training or not. Only recommended for GPUs > 2000 series

Value

List with the following fields:

`optimizer`
: Function which returns an optimizer.

`scheduler_boolean`
: Logical, use scheduler or not.

`scheduler_patience`
: Integer, number of epochs to wait before applying plateau scheduler.

`lr_reduce_factor`
: Numerical, learning rate reduce factor.

`mixed`
: Logical, use mixed training or not.

`early_stopping_training`
: Numerical, early stopping after n epochs.

`sjSDM_cv` *Cross validation of elastic net tuning*

Description

Cross validation of elastic net tuning

Usage

```
sjSDM_cv(
  Y,
  env = NULL,
  biotic = bioticStruct(),
  spatial = NULL,
  tune = c("random", "grid"),
  CV = 5L,
  tune_steps = 20L,
  alpha_cov = seq(0, 1, 0.1),
  alpha_coef = seq(0, 1, 0.1),
  alpha_spatial = seq(0, 1, 0.1),
  lambda_cov = 2^seq(-10, -1, length.out = 20),
  lambda_coef = 2^seq(-10, -0.5, length.out = 20),
  lambda_spatial = 2^seq(-10, -0.5, length.out = 20),
  device = "cpu",
  n_cores = NULL,
```

```
  n_gpu = NULL,
  sampling = 5000L,
  blocks = 1L,
  ...
)
```

Arguments

Y	species occurrence matrix
env	matrix of environmental predictors or object of type linear, or DNN
biotic	defines biotic (species-species associations) structure, object of type bioticStruct. Alpha and lambda have no influence
spatial	defines spatial structure, object of type linear, or DNN
tune	tuning strategy, random or grid search
CV	n-fold cross validation or list of test indices
tune_steps	number of tuning steps
alpha_cov	weighting of l1 and l2 on covariances: $(1-\alpha) * \|cov\| + \alpha \|\|cov\|\|^2$
alpha_coef	weighting of l1 and l2 on coefficients: $(1-\alpha) * \|coef\| + \alpha \|\|coef\|\|^2$
alpha_spatial	weighting of l1 and l2 on spatial coefficients: $(1-\alpha) * \|coef_sp\| + \alpha \|\|coef_sp\|\|^2$
lambda_cov	overall regularization strength on covariances
lambda_coef	overall regularization strength on coefficients
lambda_spatial	overall regularization strength on spatial coefficients
device	device, default cpu
n_cores	number of cores for parallelization
n_gpu	number of GPUs
sampling	number of sampling steps for Monte Carlo integration
blocks	blocks of parallel tuning steps
...	arguments passed to sjSDM, see sjSDM

Value

An S3 class of type 'sjSDM_cv' including the following components:

tune_results	Data frame with tuning results.
short_summary	Data frame with averaged tuning results.
summary	Data frame with summarized averaged results.
settings	List of tuning settings, see the arguments in DNN.
data	List of Y, env (and spatial) objects.
config	List of sjSDM settings, see arguments of sjSDM.
spatial	Logical, spatial model or not.

Implemented S3 methods include sjSDM.tune, plot.sjSDM_cv, print.sjSDM_cv, and summary.sjSDM_cv

See Also

plot.sjSDM_cv, print.sjSDM_cv, summary.sjSDM_cv, sjSDM.tune

Examples

```
## Not run:
# simulate sparse community:
com = simulate_SDM(env = 5L, species = 25L, sites = 50L, sparse = 0.5)

# tune regularization:
tune_results = sjSDM_cv(Y = com$response,
                        env = com$env_weights,
                        tune = "random", # random steps in tune-paramter space
                        CV = 2L, # 3-fold cross validation
                        tune_steps = 2L,
                        alpha_cov = seq(0, 1, 0.1),
                        alpha_coef = seq(0, 1, 0.1),
                        lambda_cov = seq(0, 0.1, 0.001),
                        lambda_coef = seq(0, 0.1, 0.001),
                        n_cores = 2L,
                        sampling = 100L,
                        # small models can be also run in parallel on the GPU
                        iter = 2L # we can pass arguments to sjSDM via...
                        )

# print overall results:
tune_results

# summary (mean values over CV for each tuning step)
summary(tune_results)

# visualize tuning and best points:
# best = plot(tune_results, perf = "logLik")

# fit model with best regularization paramter:
model = sjSDM.tune(tune_results)

summary(model)

## End(Not run)
```

---

summary.sjSDM *Return summary of a fitted sjSDM model*

Description

Return summary of a fitted sjSDM model

Usage

```
## S3 method for class 'sjSDM'
summary(object, ...)
```

Arguments

object	a model fitted by sjSDM
...	optional arguments for compatibility with the generic function, no functionality implemented

Value

The above matrix is silently returned.

---

summary.sjSDManova *Summary table of sjSDM anova*

Description

The function prints and returns invisible a summary table of an sjSDM ANOVA, created by anova.sjSDM

Usage

```
## S3 method for class 'sjSDManova'
summary(
  object,
  method = c("ANOVA"),
  fractions = c("all", "discard", "proportional", "equal"),
  ...
)
```

Arguments

object	an object of anova.sjSDM
method	method used to calculate the ANOVA
fractions	how to handle the shared fractions. See details
...	optional arguments for compatibility with the generic function, no function implemented

Details

The function returns a ANOVA table with Deviance as well as the pseudo-R2 metrics of Nagelkerke and McFadden

There are four options to handle shared ANOVA fractions, which is variance that can be explained, typically as a result of collinearity, by several of the fractions:

1. "all" returns the shared fractions explicitly
2. "discard" discards the fractions, as typically in a type II Anova
3. "proportional" distributes shared fractions proportional to the unique fractions
4. "equal" distributions shared fractions equally to the unique fractions

Value

The matrix that is printed out is silently returned

Examples

```
## Not run:
library(sjSDM)
# simulate community:
community = simulate_SDM(env = 3L, species = 10L, sites = 100L)

Occ <- community$response
Env <- community$env_weights
SP <- data.frame(matrix(rnorm(200, 0, 0.3), 100, 2)) # spatial coordinates


# fit model:
model <- sjSDM(Y = Occ,
               env = linear(data = Env, formula = ~X1+X2+X3),
               spatial = linear(data = SP, formula = ~0+X1*X2),
               family=binomial("probit"),
               verbose = FALSE,
               iter = 20) # increase iter for real analysis

# Calculate ANOVA for env, space, associations, for details see ?anova.sjSDM
an = anova(model, samples = 10, verbose = FALSE) # increase iter for real analysis

# Show anova fractions
plot(an)

# ANOVA tables with different way to handle fractions
summary(an)
summary(an, fractions = "discard")
summary(an, fractions = "proportional")
summary(an, fractions = "equal")

# Internal structure
int = internalStructure(an, fractions = "proportional")

print(int)

plot(int) # default is negative values will be set to 0
plot(int, negatives = "scale") # global rescaling of all values to range 0-1
plot(int, negatives = "raw") # negative values will be discarded

plotAssemblyEffects(int)
```

```
plotAssemblyEffects(int, negatives = "floor")
plotAssemblyEffects(int, response = "sites", pred = as.factor(c(rep(1, 50), rep(2, 50))))
plotAssemblyEffects(int, response = "species", pred = runif(10))
plotAssemblyEffects(int, response = "species", pred = as.factor(c(rep(1, 5), rep(2, 5))))

## End(Not run)
```

`summary.sjSDM_cv` *Return summary of a fitted sjSDM_cv model*

Description

Return summary of a fitted sjSDM_cv model

Usage

```
## S3 method for class 'sjSDM_cv'
summary(object, ...)
```

Arguments

`object`	a model fitted by `sjSDM_cv`
`...`	optional arguments for compatibility with the generic function, no functionality implemented

Value

Above data frame is silently returned.

`update.sjSDM` *Update and re-fit a model call*

Description

Update and re-fit a model call

Usage

```
## S3 method for class 'sjSDM'
update(object, env_formula = NULL, spatial_formula = NULL, biotic = NULL, ...)
```

Arguments

`object`	of class 'sjSDM'
`env_formula`	new environmental formula
`spatial_formula`	new spatial formula
`biotic`	new biotic config
`...`	additional arguments

Value

An S3 class of type 'sjSDM'. See `sjSDM` for more information.

Index