Package ‘sjSDM’

June 22, 2022

Type Package
Title Scalable Joint Species Distribution Modeling
Version 1.0.2
Date 2022-06-22
Description A scalable method to estimate joint Species Distribution Models (jSDMs) for big community datasets based on a Monte Carlo approximation of the joint likelihood. The numerical approximation is based on ‘PyTorch’ and ‘reticulate’, and can be run on CPUs and GPUs alike. The method is described in Pichler & Hartig (2021) <doi:10.1111/2041-210X.13687>. The package contains various extensions, including support for different response families, ability to account for spatial autocorrelation, and deep neural networks instead of the linear predictor in jSDMs.

License GPL-3
Encoding UTF-8
LazyData true
Depends R (>= 3.0)
Imports reticulate,
            stats,
            mvtnorm,
            utils,
            rstudioapi,
            abind,
            graphics,
            grDevices,
            Metrics,
            parallel,
            mgcv,
            Ternary,
            cli,
            crayon,
            ggplot2,
            checkmate,
            mathjaxr,
            ggtern
Suggests testthat,
          knitr,
          rmarkdown
RoxygenNote 7.2.0
URL: https://theoreticalecology.github.io/s-jSDM/


Roxygen: list(old_usage = FALSE)

VignetteBuilder: knitr

RdMacros: mathjaxr

R topics documented:

AccSGD ............................................. 3
AdaBound ........................................ 4
Adamax ........................................... 4
anova.sjSDM .................................... 5
bioticStruct .................................... 6
checkModel ..................................... 9
check_module ................................ 9
goof.sjSDM ...................................... 10
DiffGrad .......................................... 10
DNN ............................................... 11
generateSpatialEV ............................. 14
goof ............................................... 15
getImportance .................................. 15
getSe ............................................ 16
getWeights ...................................... 16
importance ...................................... 17
installation_help .............................. 18
install_diagnostic ............................. 20
install_sjSDM .................................. 21
is_torch_available ............................ 21
linear ........................................... 22
logLik.sjSDM .................................... 25
madgrad ......................................... 25
new_image ...................................... 26
plot.sjSDM ...................................... 26
plot.sjSDM.DNN ................................ 27
plot.sjSDManova ................................ 28
plot.sjSDMimportance ......................... 29
plot.sjSDM_cv .................................. 30
plot.sjSDMcoef ................................ 31
predict.sjSDM .................................. 32
print.bioticStruct ............................. 32
print.DNN ....................................... 33
print.linear .................................... 33
print.sjSDM ..................................... 34
print.sjSDManova ............................... 34
print.sjSDMimportance ......................... 35
print.sjSDM_cv .................................. 35
RMSprop ......................................... 36
Rsquared ......................................... 36
setWeights ...................................... 37
SGD ................................................ 38


**AccSGD**

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

**Usage**

```r
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**

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

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


Description

Calculate type I anova in the following order:
Null, biotic, abiotic (environment), and spatial (if present).
Deviance for interactions (e.g. between space and environment) are also calculated and can be visualized via `plot.sjSDManova`.

Usage

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

Arguments

- object: model of object `sjSDM`
- ...: optional arguments for compatibility with the generic function, no function implemented

Details

Compute analysis of variance

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.

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

Description

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

Usage

```r
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: λ * (lasso + ridge)

alpha
  weighting between lasso and ridge: (1 − α) * |covariances| + α * ||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

plot.sjSDManova, print.sjSDManova

sjSDM
Examples

```r
## 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)
# increase iter for your own data

coeff(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)
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) # increase iter for your own data
summary(model)

## without intercept:
model = update(model, env_formula = \~0+X1:X2 + X3)
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) # increase iter for your own data
```
bioticStruct

summary(model)
predict(model, newdata = com$env_weights, SP = XY)
R2 = Rsquared(model)
predict(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) # increase iter for your own data
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)

## Visualize internal meta-community structure
an = anova(model)
plot(an, internal=TRUE)

## 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
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$response,
              env = linear(com$env_weights, lambda = 0.01, alpha = 0.5),
              spatial = linear(SPV, ~0+.),
              biotic = bioticStruct(lambda = 0.01, alpha = 0.5),
              iter = 2L) # increase iter for your own data
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+.),
              iter = 50L) # increase iter for your own data
result = anova(model)
print(result)
plot(result)

## visualize meta-community structure
plot(result, internal=TRUE)
# Deep neural network

## we can fit also a deep neural network instead of a linear model:

```r
deanet = sjSDM(Y = com$response,
               env = DNN(com$env_weights, hidden = c(10L, 10L, 10L)),
               iter = 2L) # increase iter for your own data
```

```r
summary(deanet)
getCov(deanet)
pred = predict(deanet, newdata = com$env_weights)
```

## extract weights

```r
weights = getWeights(deanet)
```

## we can also assign weights:

```r
setWeights(deanet, weights)
```

## with regularization:

```r
deanet = sjSDM(Y = com$response,
               env = DNN(com$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),
               iter = 2L) # increase iter for your own data
```

```r
getCov(deanet)
getWeights(deanet)
```

## End(Not run)

---

### Description

**checkModel**

check model check model and rebuild if necessary

### Usage

```r
checkModel(object)
```

### Arguments

- **object**
  - of class sjSDM

---

### Description

check module check module

### Usage

```r
check_module()
```
coef.sjSDM  

Return coefficients from a fitted sjSDM model

Description

Return coefficients from a fitted sjSDM model

Usage

```r
## 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

```r
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,  # matrix of environmental predictors
  formula = NULL,  # formula object for predictors
  hidden = c(10L, 10L, 10L),  # hidden units in layers, length of hidden corresponds to number of layers
  activation = "relu",  # 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 = TRUE,  # 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 = 0,  # lambda penalty, strength of regularization: $\lambda \ast (\text{lasso} + \text{ridge})$
  alpha = 0.5,  # weighting between lasso and ridge: $(1 - \alpha) \ast |\text{weights}| + \alpha ||\text{weights}||^2$
  dropout = 0  # probability of dropout rate
)

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: $\lambda \ast (\text{lasso} + \text{ridge})$
- alpha: weighting between lasso and ridge: $(1 - \alpha) \ast |\text{weights}| + \alpha ||\text{weights}||^2$
- 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:

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

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

coef(model)
summary(model)

## plot results
species = c("sp1", "sp2", "sp3", "sp4", "sp5", "sp6", "sp7")
group = c("mammal", "bird", "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)

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

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

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) # 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) # increase iter for your own data
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)

## Visualize internal meta-community structure
an = anova(model)
plot(an, internal=TRUE)

## 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
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
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+.),
            iter = 50L) # increase iter for your own data
result = anova(model)
print(result)
generateSpatialEV

Generate spatial eigenvectors

description

function to generate spatial eigenvectors to account for spatial autocorrelation

Usage

generateSpatialEV(coords = NULL, threshold = 0)

Arguments

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

Value

Matrix of spatial eigenvectors.
**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**

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

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 \texttt{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 variance partitioning approach is based on Ovaskainen et al., 2017. For an example how to interpret the outputs, see Leibold et al., 2021. This function will be deprecated in the future. Please use \texttt{plot(anova(model), internal=TRUE)} (currently only supported for spatial models).

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 \texttt{print.sjSDMimportance} and \texttt{plot.sjSDMimportance}

Author(s)

Maximilian Pichler

References


See Also

\texttt{print.sjSDMimportance, plot.sjSDMimportance}
## Examples

```r
## 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)
imp = importance(model)
plot(imp)
## End(Not run)
```

---

## 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:

```bash
$ 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:

```bash
$ 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.

---

**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.
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).

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: $\lambda \ast (\text{lasso} + \text{ridge})$
alpha weighting between lasso and ridge: $(1-\alpha) \ast \|\text{coefficients}\| + \alpha \|\text{coefficients}\|^2$

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)
# increase iter for your own data

coeff(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)
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)  # increase iter for your own data
summary(model)

## without intercept:
model = update(model, env_formula = ~0+X1:X2 + X3)
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)  # 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)  # increase iter for your own data
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)

## Visualize internal meta-community structure
an = anova(model)
plot(an, internal=TRUE)
## non-linear(deep neural network) model

```r
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
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

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

# Anova

```r
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+.),
  iter = 50L) # increase iter for your own data
result = anova(model)
print(result)
plot(result)
```

## visualize meta-community structure

```r
plot(result, internal=TRUE)
```

# Deep neural network

## we can fit also a deep neural network instead of a linear model:

```r
model = sjSDM(Y = com$response,
  env = DNN(com$env_weights, hidden = c(10L, 10L, 10L)),
  iter = 2L) # increase iter for your own data
summary(model)
getCov(model)
pred = predict(model, newdata = com$env_weights)
```

## extract weights

```r
weights = getWeights(model)
```

## we can also assign weights:

```r
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),
               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**

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

**Arguments**

- **object**
  - a model fitted by `sjSDM`
- **individual**
  - returns internal ll structure, mostly for internal usage
- **...**
  - 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**

```r
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


---

`new_image` function

Description

`new_image` function

Usage

```r
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` function

Description

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

Usage

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

Arguments

- `x`: a model fitted by `sjSDM`
- `...`: Additional arguments to pass to `plotsjSDMcoef`
plot.sjSDM.DNN  

Value

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

Author(s)

CAI Wang

See Also

plotsjSDMcoef

Examples

```r
## 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)

# create a group dataframe for plot
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)
```

### Description

Plot training loss history

#### Usage

```r
## 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.
plot.sjSDManova

Examples

```r
## 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)

# create a group dataframe for plot
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

```r
## S3 method for class 'sjSDManova'
plot(
x,
y,
type = c("Deviance", "Nagelkerke", "McFadden"),
internal = FALSE,
cols = c("#7FC97F", "#BEAED4", 
alpha = 0.15,
env_deviance = NULL,
...
)
```

Arguments

- **x**: anova object from `anova.sjSDM`
- **y**: unused argument
- **type**: deviance, Nagelkerke or McFadden R-squared
- **internal**: logical, plot internal or total structure
- **cols**: colors for the groups
- **alpha**: alpha for colors
- **env_deviance**: environmental deviance
- **...**: Additional arguments to pass to `plot()`

The `internal = TRUE` plot was heavily inspired by Leibold et al., 2022
Value

List with the following components:

If `internal=TRUE`:
- **plots**: ggplot objects for sites and species.
- **data**: List of data.frames with the shown results.

else:
- **VENN**: Matrix of shown results.

References


Description

Plot importance

Usage

```r
## S3 method for class 'sjSDMimportance'
plot(
x, 
y, 
contour = FALSE,
col.points = "#24526e",
cex.points = 1.2,
pch = 19,
col.contour = "#ffbf02",
...
)
```

Arguments

- **x**: a model fitted by `importance`
- **y**: unused argument
- **contour**: plot contour or not
- **col.points**: point color
- **cex.points**: point size
- **pch**: point symbol
- **col.contour**: contour color
- **...**: Additional arguments to pass to `plot()`

Value

The visualized matrix is silently returned.
Description

Plot elastic net tuning

Usage

```r
## 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.
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)

# create a group dataframe for plot
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**
```r
## S3 method for class 'sjSDM'
predict(
  object,
  newdata = NULL,
  SP = 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)
- `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)

---

**print.bioticStruct**  
*Print a bioticStruct object*

**Description**  
Print a bioticStruct object

**Usage**
```r
## 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

Description

Print sjSDM anova

Usage

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

Arguments

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

Value

The above matrix is silently returned
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.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.
### Description
RMSprop optimizer

#### Usage
```
RMSprop(
    alpha = 0.99,
    eps = 1e-08,
    weight_decay = 0.01,
    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.

### Description
R-squared

#### Usage
```
Rsquared(model, method = c("Nagelkerke", "McFadden"))
```

#### Arguments
- **model**: model
- **method**: Nagelkerke or McFadden
setWeights

Details

Calculate R-squared following Nagelkerke or McFadden:

• Nagelkerke: \( R^2 = 1 - \exp\left(\frac{2}{N} \cdot (\log L_0 - \log L_1)\right) \)
• McFadden: \( R^2 = 1 - \frac{\log L_1}{\log L_0} \)

Value

R-squared as numeric value

Author(s)

Maximilian Pichler

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**

**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**

```r
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 accuracy

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**

```r
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"
)

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 family functions
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.
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 $\Sigma$ accounts for correlations in $e_{ij}$:

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

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 $Y_i$ is:

$$P_r(Y_i|X_i, \beta, \Sigma) = \int_{A_{i1}}^{\text{inf}} \cdots \int_{A_{i1}}^{\text{inf}} \phi_X(Y_{i1}, X_{i1}, \beta, \Sigma) dY_{i1} \ldots dY_{iJ}$$

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

and $\phi$ being the density function of the multivariate normal distribution.

The probability of $Y_i$ requires to integrate over $Y_{i1}^*$ 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:

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

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:
\[
\mathcal{L}(\beta, \Sigma; Y_i, X_i) \approx \frac{1}{M} \sum_{m=1}^{M} \prod_{j=1}^{J} \Pr(Y_{ij}|X_i \beta + \zeta_m)
\]
with \(\zeta_m \sim MVN(0, \Sigma)\).

sjSDM uses 'PyTorch' to run optionally the model on the graphical processing unit (GPU). Python dependencies need 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:
- **binomial**: "probit" or "logit"
- **poisson**: "log"
- **gaussian**: "identity"

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

\[
g(Z_{ij}) = \beta_0 + \sum_{n=1}^{N} X_n \beta_{nj} + \sum_{m=1}^{M} S_m \alpha_{mj} + \epsilon_{ij}
\]

There are two ways to generate spatial predictors \(S\):
- **trend surface model** - using spatial coordinates in a polynomial:
  \[
  \text{linear(data=Coords, ~0+poly(X, Y, degree = 2))}
  \]
- **eigenvector spatial filtering** - using spatial eigenvectors. Spatial eigenvectors can be generated by the `generateSpatialEV` function:
  \[
  \text{SPV = generateSpatialEV(Coords)}
  \]
  Then we use, for example, the first 20 spatial eigenvectors:
  \[
  \text{linear(data=SPV[ ,1:20], ~0+.)}
  \]

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

See Also
update.sjSDM, sjSDM_cv, DNN, plot.sjSDM, print.sjSDM, predict.sjSDM, coef.sjSDM, summary.sjSDM, getCov, 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)
# increase iter for your own data

coeff(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)
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) # increase iter for your own data
summary(model)

## without intercept:
model = update(model, env_formula = ~0+X1:X2 + X3)
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) # 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) # increase iter for your own data
summary(model)
predict(model, newdata = com$env_weights, SP = SPV)

## Visualize internal meta-community structure
an = anova(model)
plot(an, internal=TRUE)
## non-linear (deep neural network) model

```r
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
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

```r
model = sjSDM(Y = com$response,
               env = linear(com$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
summary(model)
coef(model)
getCov(model)
```

### Anova

```r
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+.),
               iter = 50L) # increase iter for your own data
result = anova(model)
print(result)
plot(result)
```

### Deep neural network

- We can fit also a deep neural network instead of a linear model:

```r
model = sjSDM(Y = com$response,
               env = DNN(com$env_weights, hidden = c(10L, 10L, 10L)),
               iter = 2L) # increase iter for your own data
summary(model)
getCov(model)
pred = predict(model, newdata = com$env_weights)
```

### Extract weights

```r
weights = getWeights(model)
```

### Assign weights

```r
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),
    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**

```r
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
- **tune_steps**: number of tuning steps
- **alpha_cov**: weighting of l1 and l2 on covariances: \((1 - \alpha) \cdot |cov| + \alpha ||cov||^2\)
alpha_coef  weighting of l1 and l2 on coefficients: \((1 - \alpha) \times |\text{coef}| + \alpha ||\text{coef}||^2\)

alpha_spatial  weighting of l1 and l2 on spatial coefficients: \((1 - \alpha) \times |\text{coef_s_p}| + \alpha ||\text{coef_s_p}||^2\)

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

```r
## 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:
# 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 parameter:
# 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

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

### Description

Return summary of a fitted sjSDM_cv model

### Usage

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

**Update and re-fit a model call**

**Description**

Update and re-fit a model call

**Usage**

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

**Arguments**

- **object**: a model fitted by `sjSDM_cv`
- **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

AccSGD, 3, 46
AdaBound, 4, 46
Adamax, 4, 46
anova.sjSDM, 5, 28, 34, 43
binomial, 42
bioticStruct, 6, 30, 32, 40, 47
check_module, 9
checkModel, 9
coeff.sjSDM, 10, 43
DiffGrad, 10
DNN, 11, 15, 16, 22, 27, 30, 33, 37, 40, 41, 43, 47, 48
gaussian, 42
generateSpatialEV, 14, 42
getCov, 15, 43
getImportance, 15
getSe, 16, 43
getWeights, 16, 37
importance, 17, 29, 35, 43
install_diagnostic, 20, 20, 42
install_sjSDM, 18, 21, 21, 42
installation_help, 18, 20, 21, 42
is_torch_available, 21
linear, 12, 22, 30, 33, 40, 41, 47
logLik.sjSDM, 25
madgrad, 25, 46
new_image, 26
plot.sjSDM, 26, 43
plot.sjSDM.DNN, 27
plot.sjSDM_cv, 30, 48
plot.sjSDManova, 5, 6, 28
plot.sjSDManova, 17, 29
plotsjSDMcoef, 26, 27, 31
poisson, 42
predict.sjSDM, 32, 43
print.bioticStruct, 6, 32
print.DNN, 11, 33
print.linear, 22, 33
print.sjSDM, 34, 43
print.sjSDM_cv, 35, 48
print.sjSDManova, 5, 6, 34
print.sjSDMimportance, 17, 35
RMSprop, 36, 46
Rsquared, 36
sessionInfo, 43
setWeights, 37
SGD, 38, 46
simulate.sjSDM, 38, 43
simulate_sjSDM, 39
sjSDM, 5, 6, 10, 12, 15–17, 22, 25–27, 31, 32, 34, 37, 38, 40, 42, 43, 48–50
sjSDM.tune, 43, 48
sjSDM_cv, 30, 35, 41, 43, 47, 50
sjSDMcontrol, 41, 46
summary.sjSDM, 43, 49
summary.sjSDM_cv, 48, 49
update.sjSDM, 43, 50