Package: sjSDM (via r-universe)

November 22, 2026

Type Package

Title Scalable Joint Species Distribution Modeling

Version 1.1.0

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 can 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. (2022) <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, for grouping-factor random effects in the environmental formula, and for deep neural networks, which can replace the linear predictor or sit beside it as one block of a formula that also holds interpretable linear terms. 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 'torch', which can be run on CPUs or GPUs.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0)

Imports torch (>= 0.14.0), reformulas, stats, mvtnorm, utils, rstudioapi, abind, graphics, grDevices, Metrics, parallel, mgcv, cli, crayon, ggplot2, checkmate, mathjaxr, beeswarm, qgam, scales, viridis

Suggests testthat, knitr, rmarkdown, fields

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

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

VignetteBuilder knitr

RdMacros mathjaxr

Config/roxygen2/version 8.1.0

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 https://cran.r-universe.dev

Date/Publication 2026-11-22 17:10:02 UTC

RemoteUrl https://github.com/cran/sjSDM

RemoteRef HEAD

RemoteSha d45bb98d8d9fcdeabdaa4e2d95b9ed43386456db

Contents

AccSGD 4
AdaBound 4
Adagrad 5
Adam 6
Adamax 6
AdamW 7
anova.sjSDM 8
bioticStruct 11
butterflies 16
coef.sjSDM 17
DiffGrad 18
DNN 19
eucalypts 24
generateSpatialEV 25
getCor 25
getCov 26
getRE 27
getSe 28
getWeights 29
importance 29
install_diagnostic 30
install_sjSDM 31
installation_help 32

internalStructure . . . 34
is_torch_available . . . 35
linear . . . 36
logLik.sjSDM . . . 41
madgrad . . . 42
NN . . . 42
plot.sjSDM . . . 44
plot.sjSDM.DNN . . . 45
plot.sjSDM_cv . . . 46
plot.sjSDManova . . . 47
plot.sjSDMimportance . . . 48
plot.sjSDMinternalStructure . . . 49
plotAssemblyEffects . . . 50
plotsjSDMcoef . . . 52
predict.sjSDM . . . 53
print.bioticStruct . . . 56
print.DNN . . . 56
print.linear . . . 57
print.NN . . . 57
print.sjSDM . . . 58
print.sjSDM_cv . . . 58
print.sjSDManova . . . 59
print.sjSDMimportance . . . 60
print.sjSDMinternalStructure . . . 61
re . . . 61
residuals.sjSDM . . . 62
RMSprop . . . 63
Rsquared . . . 63
setWeights . . . 64
SGD . . . 65
simulate.sjSDM . . . 65
simulate_SDM . . . 66
sjSDM . . . 67
sjSDM_cv . . . 76
sjSDMControl . . . 78
summary.sjSDM . . . 79
summary.sjSDM_cv . . . 80
summary.sjSDManova . . . 80
update.sjSDM . . . 82

Index **84**

AccSGD *AccSGD*

Description

Deprecated. 'torch' has no AccSGD implementation, so this redirects to SGD.

Usage

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

Arguments

kappa	long step, ignored
xi	advantage parameter, ignored
small_const	small constant, ignored
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

Deprecated. 'torch' has no AdaBound implementation, so this redirects to Adam.

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, ignored
gamma	small_const, ignored
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.

Adagrad *Adagrad*

Description

Adaptive subgradient optimizer, see Duchi et al., 2011

Usage

```
Adagrad(
  lr_decay = 0,
  weight_decay = 0,
  initial_accumulator_value = 0,
  eps = 1e-10
)
```

Arguments

lr_decay	learning rate decay
weight_decay	l2 penalty on weights
initial_accumulator_value	initial value of the gradient accumulator
eps	fuzz factor

Value

Anonymous function that returns optimizer when called.

References

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(7).

`Adam` *Adam*

Description

Adam optimizer, see Kingma and Ba, 2014

Usage

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

Arguments

`betas`	exponential decay rates
`eps`	fuzz factor
`weight_decay`	l2 penalty on weights
`amsgrad`	use the AMSGrad variant

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.

`Adamax` *Adamax*

Description

Deprecated. 'torch' has no Adamax implementation, so this redirects to `Adam`.

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.

AdamW *AdamW*

Description

Adam with decoupled weight decay, see Loshchilov and Hutter, 2019

Usage

```
AdamW(betas = c(0.9, 0.999), eps = 1e-08, weight_decay = 0.01, amsgrad = FALSE)
```

Arguments

betas	exponential decay rates
eps	fuzz factor
weight_decay	decoupled l2 penalty on weights
amsgrad	use the AMSGrad variant

Value

Anonymous function that returns optimizer when called.

References

Loshchilov, I., & Hutter, F. (2019). Decoupled weight decay regularization. International Conference on Learning Representations.

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,
  proposal = c("laplace", "prior"),
  ...
)
```

Arguments

object	model of object sjSDM
samples	Number of Monte Carlo samples
verbose	TRUE or FALSE, indicating whether progress should be printed or not
proposal	distribution the latent factors are drawn from. "laplace" (the default) draws from a Gaussian fitted to the per-site mode of the integrand and reweights the draws. "prior" draws from N(0, I), which is what every version before 1.1.0 did. The reweighting keeps the estimator of the likelihood unbiased and raises the effective sample size by orders of magnitude at many species. The log of that estimate, which is what the fractions are built from, keeps a bias of roughly (1/ess - 1)/(2 * samples) per site. See Details.
...	optional arguments which are passed to the calculation of the logLikelihood, including mc_batch, the number of sites whose Monte-Carlo draws are held in memory at once. It defaults to the model's step_size, lowered if that would exceed roughly 2.5 GB; lower it by hand on a device with less memory. For a model with random effects also re_draws (default 50), the number of draws of the group effects their marginal likelihood is averaged over; the cost of the anova grows linearly with it.

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'.

The sub-models without associations, and the null model, carry no latent term at all. Up to 1.0.7 they were fitted with an identity covariance, which for a probit link only rescales the coefficients but for the count, negative binomial and gaussian links adds a fixed unit latent variance; that misfit was then reported as associations.

For the count family this has a consequence worth knowing: the latent term of the full model also carries each species' own extra variance (the diagonal of the association matrix), i.e. log-normal overdispersion, and the sub-models without associations cannot. Overdispersion is therefore part of the Associations fraction. In simulations with 7 independent species and 180 sites, the Associations fraction (McFadden R-squared, mean over 5 datasets) of a Poisson model was 0.002 without overdispersion, 0.02 with a log-normal standard deviation of 0.5 and 0.15 with 1. For overdispersed counts use family = "nbinom", whose dispersion parameter is present in every sub-model; on the same data it was 0.002, 0.004 and 0.008.

proposal = "prior" draws the latent factors blind and weights them by the likelihood. At many species almost every draw lands where the observed community is implausible, so the effective sample size collapses – measured at 100 species it is 3 draws out of 1000 – and the negative log-likelihood -log(mean(exp(lp))) is biased upwards by an amount that grows with the variance of the per-draw log-probability. That variance differs between the sub-models anova compares, so it does not cancel in the differences the fractions are built from, and the sub-model that leans hardest on the latent factors is depressed most. proposal = "laplace" reduces this by sampling from a Gaussian centred on the per-site mode with the curvature of the integrand, and correcting each draw by log p(z) - log q(z). It is a proposal, not an approximation: a poor fit costs effective sample size, not correctness. The realised fraction is returned in the ess component, where 1 means every draw counted. Read it as a per-dimension efficiency raised to the power of the latent dimension df, not as a per-site quality score. The sub-models without associations have nothing to integrate and report 1.

For the gaussian family the McFadden R-squared is a ratio of log-densities, which change with the measurement scale of the response and turn positive for small residual variances, so it is not bounded by 1 and differs between units. The deviances and the Nagelkerke R-squared do not have this problem.

anova is not defined for a model fitted with bioticStruct(diag = TRUE), whose sub-models with associations would not be nested in it, and raises an error.

For a model with random-effect bars the anova is conditional on the random intercepts: the null model and every sub-model contain the random intercept of each grouping factor, so the fractions are the variation explained by environment, associations and space beyond the grouping. A random slope belongs to the abiotic component and is removed with it; for (temp | plot) the null holds (1 | plot) and only the sub-models with the abiotic component hold (temp | plot). The grouping component reports what the random intercepts alone explain against an intercept-only model. Every model is scored on its marginal likelihood, with the group effects integrated out by importance sampling from their fitted posterior (re_draws, see ...). With a random intercept on a site or plot identifier, covariates that are constant within sites compete with it for the variation between groups: in simulations such a covariate kept about half of its deviance, while covariates that vary within groups kept all of it. The estimated group variance does not reach zero, so a random intercept without a true group effect still takes a little from the full model (about 2.5 per cent of its R-squared in simulations), and a random slope without a true slope variance adds a little to the abiotic fraction.

With crossed grouping factors, e.g. `(1 | plot) + (1 | year)`, the group effects cannot be integrated separately and `re_draws` has to be raised substantially.

The `species` and `sites` components are not the printed table restricted to a species or a site, and they are read differently. The table composes the fractions on the log-likelihood scale and forms a single ratio at the end, so it is exact inclusion-exclusion on the joint likelihoods. The per-species and per-site values are allocations: each site's improvement over the null is apportioned across species in proportion to how much each species' conditional likelihood improved, and that share is computed separately for every sub-model. A per-species or per-site fraction is therefore a difference of two allocations computed under different shares, it is unbounded in both directions, and 1 is not a boundary for it. The shares are undefined when a sub-model is about as good as the null overall, so treat per-species and per-site values from such a model with care. The site totals are unaffected: the shares cancel when summed over species.

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.
`proposal`	the sampler the latent factors were drawn from.
`grouping`	for a model with random effects: log-likelihood of the intercept-only model, of the null with random intercepts, and the McFadden R-squared of the random intercepts alone.
`ess`	sites-by-sub-model matrix of effective sample sizes, as a fraction of `samples`. One column per fitted sub-model, so a low column shows which model's likelihood rests on few draws. `attr(, "newton_decrement")` carries the largest Newton decrement the Laplace mode-finder left behind, per sub-model.
`object`	the fitted model the anova was computed for

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	number of latent dimensions of the association matrix, i.e. its rank. If NULL, df is set to max(5, min(floor(J/2), floor(n/10))) for J species and n sites, see details
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)

Details

The association matrix is $\sigma\sigma^T + I$ with σ of size species x df. Up to 1.0.7 the default was max(5, floor(J/2)), which ignores how much information the data hold. Every latent dimension beyond what the sites can identify fits noise and inflates the associations: in simulations with 500 sites and a true rank of 5, the slope of estimated on true covariances was about 1 for df up to 100 (n/5), 1.2 at df = n/2, 1.9 at df = n and 36 at df = 5n (5000 species), and conditional predictions degraded with it. The default is therefore also capped at n/10. It changes nothing when J/2 <= n/10, i.e. for fewer species than a fifth of the sites. With many species and few sites, set df explicitly, or compare a few values by cross-validation; the cap is a guard against overfitting, not an estimate of the true rank.

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 assembly effects

plotAssemblyEffects(internal)

### see ?anova.sjSDM for more 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(result)
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)


# Grouping factors (random effects)
## the 100 sites lie in 10 plots; bars as in 'lme4' go into the environmental formula
com = simulate_SDM(env = 3L, species = 7L, sites = 100L)
X = data.frame(com$env_weights, plot = factor(rep(1:10, each = 10)))
model = sjSDM(Y = com$response,
              env = linear(X, ~ X1 + X2 + X3 + (1 | plot)),
              verbose = FALSE,
              iter = 50L) # increase iter for your own data
summary(model)
getRE(model)$plot$cov[, 1, 1] # group variance per species

## the variation partitioning is conditional on the plots, and reports what
## the plots alone explain
result = anova(model, verbose = FALSE)
summary(result)


# Faster training on a GPU with the traced likelihood (falls back to the CPU
# with a warning if no GPU is available; on the CPU it helps only for small communities)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              device = "gpu", control = sjSDMControl(jit = TRUE),
              verbose = FALSE,
              iter = 50L)

## 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)
```

---

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

Details

DNN replaces the linear predictor by one network over all covariates, so the model has no interpretable coefficients left. Use linear with an NN block instead when a flexible term should sit beside interpretable linear terms.

A DNN formula is a plain model formula. NN blocks and random-effect bars are a linear formula's business and are rejected here.

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`, `NN`, `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 assembly effects

plotAssemblyEffects(internal)

### see ?anova.sjSDM for more 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(result)
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)


# Grouping factors (random effects)
## the 100 sites lie in 10 plots; bars as in 'lme4' go into the environmental formula
com = simulate_SDM(env = 3L, species = 7L, sites = 100L)
X = data.frame(com$env_weights, plot = factor(rep(1:10, each = 10)))
model = sjSDM(Y = com$response,
              env = linear(X, ~ X1 + X2 + X3 + (1 | plot)),
              verbose = FALSE,
              iter = 50L) # increase iter for your own data
summary(model)
getRE(model)$plot$cov[, 1, 1] # group variance per species

## the variation partitioning is conditional on the plots, and reports what
## the plots alone explain
result = anova(model, verbose = FALSE)
summary(result)


# Faster training on a GPU with the traced likelihood (falls back to the CPU
# with a warning if no GPU is available; on the CPU it helps only for small communities)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
```

```
                device = "gpu", control = sjSDMControl(jit = TRUE),
                verbose = FALSE,
                iter = 50L)

## 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`	pairs of sites closer than `threshold` get weight 1 instead of 1/distance, which caps the weight of very close sites; 0 (the default) leaves all weights at 1/distance

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, which = c("residual", "re"))

## S3 method for class 'sjSDM'
getCor(object, which = c("residual", "re"))
```

Arguments

object	a model fitted by sjSDM, or sjSDM with DNN object
which	"residual", the default, for the residual association matrix. "re" for the correlation a random-effect bar induces, one entry per bar; see getCov for the shape. Not defined for a bar with re(loading = "shared"), where it is 1 by construction.

Value

Matrix of dimensions species by species corresponding to the covariance (occurrence) matrix, or for which = "re" a named list with one correlation matrix per random-effect bar.

See Also

sjSDM,DNN,getRE

---

getCov *getCov*

Description

get species-species association (covariance) matrix

Usage

```
getCov(object, which = c("residual", "re"))

## S3 method for class 'sjSDM'
getCov(object, which = c("residual", "re"))
```

Arguments

object	a model fitted by sjSDM, or sjSDM with DNN object
which	"residual", the default, for the residual association matrix $\sigma\sigma' + I$. "re" for the covariance a random-effect bar induces, $\Lambda\Lambda'$, one entry per bar.

Details

which = "re" returns a list named by the grouping variables. A bar with q design columns gives the full $Jq \times Jq$ covariance of the group effect, flattened species-fast and named <species>_k<column>, so species j's own $q \times q$ block is getRE(object)[[b]]$cov[j, ,]; at $q = 1$ it is the plain $J \times J$ matrix named by species. A bar with re(loading = "shared") has no species dimension and returns the scalar λ^2.

Unlike the residual covariance, $\Lambda\Lambda'$ carries no $+I$, so its rank is at most d and it is singular whenever $d < Jq$. getCor warns there: the group effects of the columns are linearly dependent, and at $d = 1$ every correlation is exactly ± 1.

Value

Matrix of dimensions species by species corresponding to the covariance (occurrence) matrix, or for `which = "re"` a named list with one covariance matrix per random-effect bar.

See Also

sjSDM,DNN,getRE

---

getRE *Get the random effects*

Description

Group-level latent factors of a model fitted with random-effect bars in the environmental formula, one entry per bar.

Usage

```
getRE(object)

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

Arguments

`object`	a model fitted by `sjSDM`

Details

Λ_b is identified only up to an orthogonal rotation of R^{d_b}, exactly as `sigma` is, so the quantity to report is `cov`, the per-species covariance $G_j = \Lambda_j \Lambda_j'$ of the bar's design columns, and `mean`, the posterior mean group effect $m_g' \Lambda_{jk}$ on the linear predictor. `lambda` and `L` are the raw ingredients, for a user who needs them: `L` is the posterior Cholesky factor, the counterpart of `lme4`'s `condVar`, and $L_g L_g'$ is the posterior covariance of u_g that error bars on a group effect are built from.

A bar with `re(loading = "shared")` has no species dimension: `lambda` is a vector of length q and `cov` is $\lambda\lambda'$, a scalar for `(1 | g)`, rather than the `species` by q by q array.

Value

A list with one element per bar, named by its grouping variable, each holding `group` (the grouping variable), `terms` and `levels`, `q` (the number of design columns of the bar), `nlevels`, `df`, `loading`, `cov` (species by q by q), `mean` (levels by q by species), `lambda` (species by q by df, the loading Λ) and `L` (levels by df by df, the posterior Cholesky factor). `NULL` for a model without random effects.

See Also

sjSDM, re, getCov

Examples

```
## Not run:
com = simulate_SDM(env = 3L, species = 5L, sites = 60L)
X = data.frame(com$env_weights)
X$plot = factor(rep(1:6, each = 10))

model = sjSDM(com$response, env = linear(X, ~ X1 + re(1 | plot, df = 2)),
              iter = 10L, verbose = FALSE)

rr = getRE(model)
dim(rr$plot$cov)   # species by q by q
dim(rr$plot$mean)  # groups by q by species

getCov(model, which = "re")$plot

## End(Not run)
```

`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

- env - list of the environmental layer weights
- spatial - list of the spatial layer weights (NULL without a spatial model)
- sigma - weight to construct covariance matrix
- state - every parameter and buffer of the model by name, including NN() blocks, random-effect blocks and the dispersion; this is what setWeights restores

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`

`install_diagnostic` *install diagnostic*

Description

Print information about the 'torch' installation and the compute devices.

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's dependencies*

Description

Install sjSDM's dependencies

Usage

```
install_sjSDM(...)
```

Arguments

`...`	ignored, kept for backward compatibility (e.g. `version = "gpu"`)

Details

Since version 1.1.0 sjSDM runs on 'torch' (libtorch) and no longer needs 'python', 'conda' or 'PyTorch'. The only remaining dependency is the libtorch binary, which is downloaded by `install_torch`. This function is a thin wrapper around it so that existing scripts keep working. Whether the GPU is used is decided by the torch installation, not by an argument here.

Value

No return value, called for side effects (installation of the 'torch' binaries).

See Also

`installation_help`, `install_diagnostic`

`installation_help` *Installation help*

Description

Trouble shooting guide for the installation of the sjSDM package

Since version 1.1.0 sjSDM runs on the 'torch' package (libtorch) and no longer needs 'python', 'conda', 'reticulate' or 'PyTorch'. Installing sjSDM from CRAN pulls in 'torch', and 'torch' then downloads the libtorch and liblantern binaries on first use. Almost every installation problem is a problem with that download.

Before you start

Check what is missing:

```
sjSDM::install_diagnostic()
```

If it reports `libtorch installed: FALSE`, the binaries are missing. Get them with

```
torch::install_torch()
# or, equivalently
sjSDM::install_sjSDM()
```

and restart R afterwards.

Behind a proxy or without an internet connection

'torch' can be installed from local files. Download the libtorch and liblantern archives matching your platform on a machine that has access, then point 'torch' at them before loading it:

```
Sys.setenv(TORCH_URL = "/path/to/libtorch.zip")
Sys.setenv(LANTERN_URL = "/path/to/liblantern.zip")
torch::install_torch()
```

See `?torch::install_torch` for the current variable names and the download URLs.

GPU support

There is no separate 'gpu' version of sjSDM any more, and `install_sjSDM()` has no `version` argument. Whether the GPU can be used is decided entirely by the 'torch' installation:

- CUDA: install a CUDA enabled 'torch' build, then check `torch::cuda_is_available()`. Pass `device = "gpu"` or `device = 0L` (the CUDA device index) to `sjSDM`.
- Apple silicon: `torch::backends_mps_is_available()`, then `device = "mps"`.
- otherwise everything runs on the CPU, which is the default.

Migrating from sjSDM 1.0.x

The 'r-sjsdm' conda environment is no longer used and can be deleted. Model code runs unchanged, but results differ from 1.0.x:

- the default optimizer is `Adam()`; `sjSDMControl(optimizer = RMSprop())` restores the old one.
- `predict()` returns marginal predictions; `predict(marginal = FALSE)` gives the 1.0.x behaviour.
- `anova()` and `Rsquared()` results change, because several defects were fixed (see `news(package = "sjSDM")`).
- `model$history` is a data.frame, the losses are in `model$history$train_l`.
- the backends use different random number streams, so no result is bit-identical.
- `sjSDMControl(mixed = TRUE)` (half precision) is accepted but ignored.

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:

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

`internalStructure` *Plot internal metacommunity structure*

Description

Plot internal metacommunity structure

Usage

```
internalStructure(
  object,
  Rsquared = c("McFadden", "Nagelkerke"),
  fractions = c("mvp_proportional", "mvp", "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, one of "mvp_proportional" (the default), "mvp", "discard", "proportional" or "equal". See `summary.sjSDManova` for what each does.
`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: $(1-\alpha)*\|coefficients\|+\alpha\|\|coefficients\|\|^2$

Details

The formula is the usual model formula with two additions, and a formula that uses neither is parsed exactly as before.

`NN(soil + ndvi, hidden = c(50, 50))` marks a group of covariates as a non-linear block. The block is a network from its own covariates to the species and its output is added to the linear predictor, so the parametric terms beside it keep their interpretation. Several blocks are allowed.

A bar adds a grouping-factor random effect with 'lme4' semantics: `(1 | plot)`, `(temp | plot)`, `(0 + temp | plot)`, `(temp || plot)`, `(1 | plot/subplot)`. `re(1 | plot, df = 2)` is the explicit form where a bar needs arguments. The group effect is a design-level latent factor with its own species covariance, see `sjSDM`.

Rejected while the formula is parsed, each with an error naming the offending term:

- a covariate in both the parametric part and an NN block, or in two blocks. A linear output layer represents a linear term exactly, so the likelihood is flat along the split and only the ratio of the two penalties decides it. `~ temp + NN(.)` is an error, not a shorthand.
- NN inside an interaction. Put both covariates inside the block instead.
- `df` below the number of columns the bar produces, which would force the intercept-slope correlation to ± 1, and `df` together with `loading = "shared"`, which has no latent rank to choose.
- re wrapping a `||` or a nested `g/h` bar. The formula parser then either drops its arguments or leaves the bar unexpanded, so the bars have to be written out: `re(1 | g, df = 2) + re(0 + x | g, df = 2)`.

Neither blocks nor bars are available in the spatial formula or in DNN, and sjSDM_cv does not support them.

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`
`nn`	List of the NN blocks the formula asks for, empty for a purely parametric formula. `X` stays the parametric design.
`re`	List of the random-effect blocks the formula's bars ask for, empty when it has none. Each holds the bar's design `X`, the group index, the levels, `df` and `loading`.

Implemented S3 methods include `print.linear`

See Also

DNN, NN, 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 assembly effects

plotAssemblyEffects(internal)

### see ?anova.sjSDM for more 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(result)
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)


# Grouping factors (random effects)
## the 100 sites lie in 10 plots; bars as in 'lme4' go into the environmental formula
com = simulate_SDM(env = 3L, species = 7L, sites = 100L)
X = data.frame(com$env_weights, plot = factor(rep(1:10, each = 10)))
```

```
model = sjSDM(Y = com$response,
              env = linear(X, ~ X1 + X2 + X3 + (1 | plot)),
              verbose = FALSE,
              iter = 50L) # increase iter for your own data
summary(model)
getRE(model)$plot$cov[, 1, 1] # group variance per species

## the variation partitioning is conditional on the plots, and reports what
## the plots alone explain
result = anova(model, verbose = FALSE)
summary(result)


# Faster training on a GPU with the traced likelihood (falls back to the CPU
# with a warning if no GPU is available; on the CPU it helps only for small communities)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              device = "gpu", control = sjSDMControl(jit = TRUE),
              verbose = FALSE,
              iter = 50L)

## 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`)

Details

For a model with random-effect bars this is the conditional log-likelihood at the posterior means of the group effects, not the evidence lower bound and not a marginal likelihood. The Kullback-Leibler term enters the training loss only and is visible through `object$history$train_l`. Values are therefore not comparable with those of a model fitted without a bar, and neither are information criteria built from them.

Value

Numeric value or numeric matrix if individual is true.

`madgrad` *madgrad*

Description

Deprecated. 'torch' has no MADGRAD implementation, so this redirects to `Adam`.

Usage

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

Arguments

`momentum`	strength of momentum, ignored
`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.

`NN` *Neural network block inside an environmental formula*

Description

Marks a group of covariates as a non-linear block of the environmental predictor. The block is a feed-forward network from its own covariates to the species, and its output is added to the linear predictor, so the parametric terms beside it keep their interpretation.

Usage

```
NN(
  terms,
  hidden = c(10L, 10L, 10L),
  activation = "selu",
  bias = TRUE,
  lambda = 0,
  alpha = 0.5,
  dropout = 0
)
```

Arguments

terms	the covariates the block is a function of, written as formula terms (a + b + c, or . for every column of the data). Captured unevaluated.
hidden	hidden units per layer, length of hidden corresponds to number of layers
activation	activation functions, of length one or one per hidden layer. Currently supported: tanh, relu, leakyrelu, selu, or sigmoid
bias	whether the hidden layers carry biases. The output layer never does, it would alias with the parametric intercept.
lambda	lambda penalty on this block's weights, strength of regularization: $\lambda * (lasso + ridge)$
alpha	weighting between lasso and ridge: $(1 - \alpha) * \|weights\| + \alpha \|\|weights\|\|^2$
dropout	probability of dropout rate

Details

A covariate may not appear both in the parametric part of the formula and in an NN() block: a linear output layer can represent a linear term exactly, so the likelihood is flat along the split and only the ratio of the two penalties decides it. ~ temp + NN(.) is therefore an error, not a shorthand.

The block has no interpretable coefficients, so coef() and summary() report the parametric part only, and standard errors become conditional on the fitted block.

Value

An S3 class of type 'NN', a specification object holding the unevaluated terms and the architecture. It is evaluated by linear when the formula is parsed.

See Also

linear, DNN, sjSDM

Examples

```
## Not run:
com = simulate_SDM(env = 3L, species = 5L, sites = 100L)
X = data.frame(com$env_weights)
```

```
model = sjSDM(com$response,
              env = linear(X, ~ X1 + NN(X2 + X3, hidden = c(10L, 10L))),
              iter = 20L, verbose = FALSE)

coef(model)  # the parametric part only

## switch the block off and refit
model_without = update(model, env_blocks = 1L, verbose = FALSE)

## End(Not run)
```

`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.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.

---

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
...	no function, for compatibility with the generic

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`	currently unused, kept for compatibility
`env_deviance`	environmental deviance/gradient (points will be colored)
`negatives`	how to handle negative R squareds
`...`	no function

Value

An empty list, invisibly; the function is called for the plot.

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)
```

`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,
  marginal = TRUE,
  loo = FALSE,
  ...
)
```

Arguments

object	a model fitted by sjSDM
newdata	newdata for predictions
SP	spatial predictors (e.g. X and Y coordinates)
Y	Observed responses (occurrences, abundances or values) of the sites in newdata, a matrix with one column per species. NA cells are predicted conditional on the observed cells of the same site, through the latent factors that carry the species associations: the conditional probability of occurrence for binomial models, the conditional expected abundance for poisson and negative binomial models and the conditional mean for gaussian models. All NA cells of a site are predicted in one pass, whatever their pattern. The result has one column per species with at least one NA cell; observed cells are NA in it. attr(, "ess") holds the effective sample size of the importance weights per site, as a fraction of sampling (default 1000 for conditional predictions); values near 0 mean the prediction rests on few draws, which happens with many observed species and a large latent rank df. Every site needs at least one observed cell; predict a site without observations unconditionally.
type	raw or link
dropout	use dropout for predictions or not, only supported for DNNs and NN() blocks
marginal	integrate the latent factor out of the prediction. The species associations are generated by an unobserved latent factor z, so a marginal prediction is $E_z[\mathrm{link}(\mu + z\Sigma')]$ and not $\mathrm{link}(\mu)$, which is the prediction at $z = 0$. The two differ whenever the associations are non-zero: for the probit link the marginal is $\Phi(\mu/\sqrt{\mathrm{diag}(\Sigma\Sigma' + I)})$. The probit, poisson, negative binomial and gaussian families use exact closed forms; the logit and linear links have none, so their marginal is approximated by Monte Carlo and is stochastic. marginal = FALSE restores the behaviour of sjSDM <= 1.0.7 and is only there for reproducing older results. The default is already the marginal, so code that rescaled predictions by hand (e.g. by sqrt(diag(getCov(model)))) to obtain it must drop that step.
loo	with a complete Y: predict every species conditional on all other species of the same site (leave one species out), all in one pass. attr(, "ess") is then a sites

by species matrix. The Laplace proposal is built from the complete row, the target species included, and only the importance weights remove it, so at a low effective sample size the result is optimistic: in simulations with 1000 species (`df = 500`) by 0.012 AUC (binomial) and 0.08 Spearman correlation (Poisson) against setting the species to `NA`. With 5000 species, `df = 100` and an effective sample size of 0.9 the two agreed (AUC 0.856 and 0.857). For an evaluation, check `attr(, "ess")`, or set the held-out species to `NA` in `Y` instead.

`...` `sampling`, the number of Monte-Carlo draws for the logit and linear marginals and for conditional predictions (default 1000)

Details

For a model with random-effect bars the prediction for a group the fit saw is taken at the posterior mean of that group. A group it did not see - a level of the grouping factor that was not in the training data, or `NA` - contributes no mean, and its group effect is integrated over the prior instead. That widens the prediction rather than shifting it, and for the `logit` and `linear` links the prior variance is added by sampling, so those predictions carry one more source of Monte-Carlo noise.

Value

Matrix of predictions, sites by species. With `Y`: sites by the species that have an `NA` cell, with the per-site effective sample size in `attr(, "ess")`; with `loo = TRUE`: sites by species, and `attr(, "ess")` a matrix of the same shape.

Examples

```
## Not run:

## Conditional predictions based on observed species
com = simulate_SDM(sites = 200L, species = 10L)
## first 100 observations are the training data
model = sjSDM(com$response[1:100, ], com$env_weights[1:100,], iter = 50L, verbose = FALSE)
## Assume that for the other 100 observations the first three species are missing
## and we want to use the other species to improve their predictions:
Y_focal = com$response[101:200, ]
Y_focal[, 1:3] = NA # set to NA because occurrences are unknown

## all NA cells of a site are predicted together, conditional on its observed species
pred_conditional = predict(model, newdata = com$env_weights[101:200,], Y = Y_focal)
pred_unconditional = predict(model, newdata = com$env_weights[101:200,])[, 1:3]

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

## effective sample size of the importance sampler per site, 1 means every draw counted
summary(attr(pred_conditional, "ess"))

## Leave one species out: every species conditional on all others, in one pass
pred_loo = predict(model, newdata = com$env_weights[101:200,],
                   Y = com$response[101:200, ], loo = TRUE)
```

```
## 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

Value

`x`, invisibly.

`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

Value

x, invisibly.

`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.NN` *Print an NN object*

Description

Print an NN object

Usage

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

Arguments

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

Value

Invisible NN 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.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.

`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`

Value

The table printed by `summary.sjSDManova`, invisibly.

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

Value

x, invisibly.

re *Random-effect bar with an explicit rank*

Description

Wraps a random-effect bar so that the rank of its group-level latent factor can be set. `re(x | g, df = 4)` is `(x | g)` with `df` latent dimensions instead of the default.

Usage

```
re(bar, df = NULL, loading = "species")
```

Arguments

bar	a random-effect bar, `lhs | group`, with `lme4` semantics.
df	number of latent dimensions d of the group effect. Must be at least the number of design columns q the bar produces, otherwise the per-species $q \times q$ covariance $\Lambda_j \Lambda_j'$ is rank deficient and the intercept-slope correlation is forced to ± 1. Defaults to `max(q, 5)`.
loading	`"species"`, the default, gives every species its own loading Λ and so its own species covariance for the group effect. `"shared"` gives the bar one loading vector for all species, i.e. the classical mixed-model row effect in which a group shifts every species by the same amount.

Details

`df` caps the rank of the *cross-species* covariance of the group effect: at `df = 1` every species' group effect is a multiple of one common group score. It does not restrict the per-species covariance, which is unconstrained once `df >= q`.

`loading = "shared"` has one scalar per group rather than a latent vector, so there is no rank to choose and `df` alongside it is an error. The implied species covariance of the group effect is $\lambda^2 \mathbf{1}\mathbf{1}'$, rank one by construction, and with $q > 1$ the bar's columns are likewise perfectly correlated: one group score loaded onto all of them. `getRE` then reports `cov` without the species dimension and `getCor(which = "re")` is not defined for the bar.

The call is read out of the formula by the parser and is never evaluated, so `re()` is syntax rather than a function to call.

Value

A list holding the bar, `df` and `loading`.

See Also

`linear`, `getRE`, `sjSDM`

`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: the observed response minus the fitted marginal expectation on the response scale
`...`	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: $R^2 = 1 - log\mathcal{L}_1/log\mathcal{L}_0$

For a model with random-effect bars the likelihood is conditional on the posterior group means, so the R-squared is a conditional one: it rises with a bar the data do not need and is not comparable with the R-squared of a model without it. For the gaussian family McFadden's R-squared depends on the measurement scale of the response and is not bounded by 1; use Nagelkerke's.

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`. If the list holds a `state` element, as returned by `getWeights()`, the complete state is restored from it and the other elements are ignored.

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. Each simulation draws the latent factors first, so the simulated species carry the fitted associations, and then the response from the family: binomial, poisson, negative binomial or gaussian. Random-effect blocks enter at their predicted group effects.

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; the caller's random number stream is restored afterwards
...	optional arguments for compatibility with the generic function, no functionality implemented

Value

Array of simulated responses 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), count, or identical (the latent continuous response)
`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. The environmental formula carries structure: it may hold `NN` blocks and 'lme4' random-effect bars beside the parametric terms, see the details.

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. A linear formula may contain NN blocks and random-effect bars, see details
biotic	defines biotic (species-species associations) structure, object of type bioticStruct. Its rank df defaults to max(5, min(floor(J/2), floor(n/10))) for J species and n sites; with many species and few sites a larger rank overestimates the associations, see 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 the optimizer, see sjSDMControl
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", "gpu", "mps", or the integer index of a CUDA device. A GPU request falls back to the CPU with a warning when no CUDA device is present
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 'torch' (libtorch) and can optionally run the model on the graphical processing unit (GPU), see `installation_help` for how to enable that.

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.

Structure in the environmental formula:

A `linear` formula is not restricted to parametric terms. It may in addition hold `NN` blocks and random-effect bars:

`linear(X, ~ temp + NN(soil + ndvi, hidden = c(50, 50)) + (temp | plot))`

Every block is summed on the scale of the linear predictor,

$$g(Z_{ij}) = \mathbf{X_i}\beta_{\mathbf{j}} + \mathbf{f}(\mathbf{X_i^{NN}})_{\mathbf{j}} + \mathbf{\Sigma_b}\mathbf{\Sigma_k}\mathbf{Z_{ik}^b}\mathbf{u_{g(i)}^b}\mathbf{\Lambda_{jk}^b} + \mathbf{e_{ij}}$$

so the parametric part keeps its meaning: `coef.sjSDM`, `summary.sjSDM` and the standard errors describe β and nothing else.

`NN(soil + ndvi)` is a feed-forward network from its own covariates to the species. It sits beside the interpretable terms rather than replacing them, which is the difference to `DNN`. The block itself has no interpretable coefficients, and the standard errors are conditional on the fitted block: they ignore its uncertainty, and a warning says so. `update.sjSDM` with `env_blocks` refits with a subset of the blocks, which is how a block is switched off.

A bar is written in 'lme4' syntax and carries 'lme4' semantics, through the 'reformulas' package: `(1 | plot)`, `(temp | plot)` for a correlated intercept and slope, `(0 + temp | plot)`, `(temp || plot)` and `(1 | plot/subplot)`. `re` is the explicit form where a bar needs arguments, `re(1 | plot, df = 2)`.

A bar is a design-level latent factor, not a scalar shift. One group effect $\mathbf{u_g} \sim \mathbf{N}(\mathbf{0}, \mathbf{I_d})$ is drawn per group and loaded onto the species by Λ, so the group effect has its own species-species covariance $\Lambda\Lambda'$ of rank `df` and species may respond to a plot with different magnitude and sign. `df` defaults to `max(q, 5)`, with q the number of columns of the bar, and must not be smaller than q: below that the intercept-slope correlation is forced to ± 1. `re(loading = "shared")` gives the classical mixed-model case instead, one shift per group identical for every species.

The bars are integrated out by a stochastic variational approximation. Each group gets a Gaussian posterior, drawn with the reparameterisation trick, and the Kullback-Leibler term to the $N(0, \mathbf{I})$ prior supplies the shrinkage; the Monte-Carlo likelihood itself is unchanged. `getRE` reports the group effects, `getCov` and `getCor` with `which = "re"` the covariance the bar induces.

Limitations of blocks and bars:

- anova.sjSDM refuses on a model with a bar. The random block is re-estimated in every refit, so it absorbs the fraction that was removed instead of cancelling from the difference, and the per-species rescaling is recomputed with it. Use getRE for the group variances.
- logLik.sjSDM for a model with a bar is the conditional value at the posterior means, neither the evidence lower bound nor a marginal likelihood. It is not comparable with the log-likelihood of a model fitted without the bar, and neither are information criteria built from it.
- sjSDM_cv supports neither blocks nor bars. It raises an error rather than dropping them.
- Neither is available in the spatial formula.
- The reported group variance grows with df, because the Kullback-Leibler term constrains the posterior and not the loadings. Report the rank alongside the variance, and prefer a small df.

Installation:

sjSDM needs the 'torch' binaries, which install_sjSDM (a wrapper around torch::install_torch) downloads. If sjSDM still does not work after restarting R, follow the 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 'torch' 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	data.frame with the training loss (train_l) per epoch that ran.
theta	Dispersion parameters of the negative binomial and gaussian families.
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, getRE, update.sjSDM, sjSDM_cv, DNN, NN, re, 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 assembly effects

plotAssemblyEffects(internal)

### see ?anova.sjSDM for more 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(result)
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)


# Grouping factors (random effects)
## the 100 sites lie in 10 plots; bars as in 'lme4' go into the environmental formula
com = simulate_SDM(env = 3L, species = 7L, sites = 100L)
X = data.frame(com$env_weights, plot = factor(rep(1:10, each = 10)))
model = sjSDM(Y = com$response,
              env = linear(X, ~ X1 + X2 + X3 + (1 | plot)),
              verbose = FALSE,
              iter = 50L) # increase iter for your own data
summary(model)
getRE(model)$plot$cov[, 1, 1] # group variance per species

## the variation partitioning is conditional on the plots, and reports what
## the plots alone explain
result = anova(model, verbose = FALSE)
summary(result)


# Faster training on a GPU with the traced likelihood (falls back to the CPU
# with a warning if no GPU is available; on the CPU it helps only for small communities)
model = sjSDM(Y = com$response, env = linear(com$env_weights),
              device = "gpu", control = sjSDMControl(jit = TRUE),
              verbose = FALSE,
              iter = 50L)

## End(Not run)
```

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

Details

Each fold refits on the already expanded design matrix, which the NN blocks and the random-effect bars of an environmental formula do not survive. A model with either raises an error on the resulting row-count mismatch rather than silently dropping them, so cross-validation of such a model is not available.

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)
```

`sjSDMControl` *sjSDM control object*

Description

sjSDM control object

Usage

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

Arguments

`optimizer`	object of type `Adam` (the default), `AdamW`, `Adagrad`, `RMSprop`, or `SGD`. `Adamax`, `AccSGD`, `madgrad` and `AdaBound` have no 'torch' counterpart, redirect to one of the above with a message, and will be removed in 1.2.0

`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. Accepted and ignored, 'torch' has no equivalent

`jit` trace the likelihood with `jit_trace` on the first batch and train with the traced graph. Same model, fewer R-level dispatches per step; the last incomplete batch is dropped in either case, so every traced batch has the same shape

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.

`jit` Logical, train with the traced likelihood or not.

`early_stopping_training` Numerical, early stopping after n epochs.

`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.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.

`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("mvp_proportional", "mvp", "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 six 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. "mvp_proportional" (the default) splits Abiotic and Spatial within the model without associations, distributes their shared part proportionally, and gives the remainder of the full model to the associations. Without a spatial component it is the same as "proportional"
2. "mvp" is the same split without distributing the shared Abiotic and Spatial part. Without a spatial component it is the same as "proportional"
3. "all" returns the shared fractions explicitly
4. "discard" discards the fractions, as typically in a type II Anova
5. "proportional" distributes shared fractions proportional to the unique fractions, weighted by their absolute size, so a negative unique fraction still attracts shared variance
6. "equal" distributes 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)
```

`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,
  env_blocks = NULL,
  seed = NULL,
  ...
)
```

Arguments

object	of class 'sjSDM'
env_formula	new environmental formula
spatial_formula	new spatial formula
biotic	new biotic config
env_blocks	indices of the environmental blocks to keep in the predictor, 1 being the parametric block and the NN blocks following it in formula order. Every other block is removed from the sum and contributes exactly zero. Random-effect bars are not selected here, they are kept in every refit.
seed	seed of the refit; NULL (the default) reuses the seed of object
...	additional arguments

Value

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

Index

∗ **datasets**
butterflies, 16
eucalypts, 24

AccSGD, 4, *78*
AdaBound, 4, *78*
Adagrad, 5, *78*
Adam, *4*, 6, *6*, *42*, *78*
Adamax, 6, *78*
AdamW, 7, *78*
anova.sjSDM, 8, *30*, *34*, *48*, *51*, *59*, *71, 72*, *80, 81*

binomial, *69*
bioticStruct, 11, *47*, *56*, *68*, *76*
butterflies, 16

coef.sjSDM, 17, *70–72*

DiffGrad, 18
DNN, 19, *26, 27*, *29*, *37*, *43*, *45*, *47*, *56*, *64*, *68*, *70*, *72*, *76, 77*

eucalypts, 24

gaussian, *69*
generateSpatialEV, 25, *70*
getCor, 25, *26*, *62*, *70*, *72*
getCov, 26, *26, 27*, *70*, *72*
getRE, *26*, 27, *27*, *62*, *70–72*
getSe, 28, *72*
getWeights, 29, *64*

importance, 29, *48*, *60*, *72*
install_diagnostic, 30, *31*, *33*, *71*
install_sjSDM, 31, *31*, *71*
install_torch, *31*
installation_help, *31*, 32, *69*, *71*
internalStructure, 34, *49*
is_torch_available, 35

jit_trace, *79*

linear, *19, 20*, 36, *43*, *47*, *57*, *62*, *68*, *70*, *76*
logLik.sjSDM, 41, *71*

madgrad, 42, *78*

NN, *19, 20*, *36, 37*, 42, *57*, *67, 68*, *70*, *72*, *77*, *83*

plot.sjSDM, 44, *71, 72*
plot.sjSDM.DNN, 45
plot.sjSDM_cv, 46, *77*
plot.sjSDManova, *9, 10*, 47
plot.sjSDMimportance, *30*, 48
plot.sjSDMinternalStructure, *10*, *34*, 49, *51*
plotAssemblyEffects, 50
plotsjSDMcoef, *44*, 52
poisson, *69*
predict.sjSDM, 53, *71, 72*
print.bioticStruct, *12*, 56
print.DNN, *20*, 56
print.linear, *37*, 57
print.NN, 57
print.sjSDM, 58, *71, 72*
print.sjSDM_cv, 58, *77*
print.sjSDManova, *10*, 59
print.sjSDMimportance, *30*, 60
print.sjSDMinternalStructure, *34*, 61

re, *26, 27*, *36, 37*, 61, *70*, *72*
residuals.sjSDM, 62
RMSprop, 63, *78*
Rsquared, 63

sessionInfo, *71*
setWeights, *29*, 64
SGD, *4*, 65, *78*
simulate.sjSDM, 65, *72*
simulate_SDM, 66

sjSDM, *8*, *13*, *18*, *20*, *26–29*, *32*, *36, 37*, *41*, *43–45*, *53, 54*, *58*, *62*, *64, 65*, 67, *71*, *77*, *79*, *83*
sjSDM-package *(*installation_help*)*, 32
sjSDM.tune, *71*, *77*
sjSDM_cv, *37*, *46*, *58*, *68*, *71, 72*, 76, *80*
sjSDMControl, *68*, 78
summary.sjSDM, *70–72*, 79
summary.sjSDM_cv, *77*, 80
summary.sjSDManova, *10*, *34*, *59*, 80

update.sjSDM, *70*, *72*, 82