Package: divoRce (via r-universe)

October 18, 2026

Title Diagnosing Separation Phenomena in Latent Polyhedral Categorical Response Models

Version 1.0-2

Maintainer Thomas Rusch <thomas.rusch@wu.ac.at>

Description Utilities for checking and diagnosing separation phenomena (complete separation, quasi-complete separation, overlap) for latent polyhedral categorical response models including binary, nominal and ordinal data models with link construction as baseline-category, adjacent-category, cumulative, sequential (continuation-ratio), ordered stereotype and arbitrary link error distribution (including logit, probit, cauchit, cloglog, loglog, log amongst others) based on structure vectors as described in Sablica, Hornik, Rusch (2026) <doi:10.48550/arXiv.2610.08100> and Rusch, Sablica, Hornik (2026) <doi:10.48550/arXiv.2610.09951>. The package functionality includes a check for separation and a check for overlap with a linear program, a quick check for overlap, a sequential and a parallel check for overlap based on subsamples, detailed separation diagnostics such as identification of columns in the design matrix/structure vector matrix that lead to separation, identification of design matrix/structure vector matrix rows that lead to separation, the dimension of the recession cone and the structure vectors that are linearities. Calculations can be done with package 'rcdd' to employ arbitrary precision and various other linear program solvers can also be used via package 'ROI'. Implemented checks can be done before model fitting based on outcome and design matrix or structure vector matrix or via formula (pre-fit), or after model fitting for objects of class 'glm', 'multinom', 'polr', 'osm', 'clm', 'bracl', 'brmultinom'. Wrappers for other classes can be easily constructed by third parties via pre-fit functions.

Depends R (>= 3.5.0)

Imports rcdd

Suggests clustord, detectseparation, MASS, nnet, ordinal, ROI

License GPL-2 | GPL-3

LazyData true

Encoding UTF-8

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Thomas Rusch [aut, cre] (ORCID: <https://orcid.org/0000-0002-7773-2096>), Lukas Sablica [aut] (ORCID: <https://orcid.org/0000-0001-9166-4563>), Kurt Hornik [aut] (ORCID: <https://orcid.org/0000-0003-4198-9911>)

Repository https://cran.r-universe.dev

Date/Publication 2026-10-18 16:30:14 UTC

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

RemoteRef HEAD

RemoteSha 6423ad28f22ac1e3d045a5498cccd9f9fe3ab2b8

Contents

Alligators . . . 3
check_overlap . . . 3
check_separation . . . 10
csepdat1 . . . 16
csepdat2 . . . 17
csepdatm . . . 17
csepdato . . . 18
diagnose_separation . . . 18
HDSS . . . 24
iris2 . . . 25
linearities . . . 26
make_yx . . . 27
nsduh2019 . . . 28
ohiovoters . . . 29
ovldat1 . . . 30
ovldat2 . . . 30
ovldatm . . . 31
ovldato . . . 31
print.sepmod . . . 32
print.sepmod_sl . . . 32
qcsepdat1 . . . 33
qcsepdat2 . . . 33
qcsepdatm . . . 34
qcsepdato . . . 34
recession_cone . . . 35
separation_columns . . . 38
separation_rows . . . 44

Silvapulle 47
structure_vectors 48
titanic3 50

Index **52**

`Alligators` *Untabeled alligator food choice data*

Description

Untabeled alligator food choice data

Usage

```
data(Alligators)
```

Format

A data frame with 219 rows corresponding to captured alligator and 5 variables:

* ‘foodchoice‘: primary food type, in volume, found in an alligator’s stomach, with levels ‘fish‘, ‘invertebrate‘,‘reptile‘, ‘bird‘, ‘other‘ * ‘lake‘: lake of capture with levels ‘Hancock‘, ‘Oklawaha‘, ‘Trafford‘, ‘George‘. * ‘sex‘: sex of the alligator with levels ‘Male‘ and ‘Female‘ * ‘size‘: size of the alligator with levels ‘<=2.3‘ meters long and ‘>2.3‘ meters long * ‘freq‘: number of alligators for each foodchoice, lake, gender and size combination

Source

The alligators data set is analyzed in Agresti (2002, Subsection 7.1.2). These data are an untabeled version.

References

Agresti A (2002). *Categorical Data Analysis*. Wiley Series in Probability and Statistics. Wiley.

`check_overlap` *Check overlap*

Description

This function checks for overlap by calling the appropriate low-level functions. It can be run with the linear program (‘quick = FALSE‘, default), linear program 2 (‘quick = TRUE‘), on sequential subsets of data (‘sequential = TRUE‘) or via multicore parallelization (‘parallel = TRUE‘).

Usage

```
check_overlap(object, ...)

## Default S3 method:
check_overlap(object, ...)

## S3 method for class 'factor'
check_overlap(
  object,
  X,
  rational = FALSE,
  quick = FALSE,
  sequential = FALSE,
  parallel = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  nc = NULL,
  nss = NULL,
  ...
)

## S3 method for class 'logical'
check_overlap(
  object,
  X,
  rational = FALSE,
  quick = FALSE,
  sequential = FALSE,
  parallel = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  nc = NULL,
  nss = NULL,
  ...
)

## S3 method for class 'numeric'
check_overlap(
  object,
  X,
  rational = FALSE,
  quick = FALSE,
  sequential = FALSE,
  parallel = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  nc = NULL,
  nss = NULL,
```

```
  ...
)

## S3 method for class 'integer'
check_overlap(
  object,
  X,
  rational = FALSE,
  quick = FALSE,
  sequential = FALSE,
  parallel = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  nc = NULL,
  nss = NULL,
  ...
)

## S3 method for class 'character'
check_overlap(
  object,
  X,
  rational = FALSE,
  quick = FALSE,
  sequential = FALSE,
  parallel = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  nc = NULL,
  nss = NULL,
  ...
)

## S3 method for class 'matrix'
check_overlap(
  object,
  rational = FALSE,
  quick = FALSE,
  sequential = FALSE,
  parallel = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  nc = NULL,
  nss = NULL,
  ...
)

## S3 method for class 'formula'
```

```
check_overlap(
  object,
  data,
  model = NULL,
  rational = FALSE,
  contrasts = NULL,
  quick = FALSE,
  sequential = FALSE,
  parallel = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'osm'
check_overlap(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  quick = FALSE,
  parallel = FALSE,
  sequential = FALSE,
  nss = NULL,
  nc = NULL,
  ...
)

## S3 method for class 'clm'
check_overlap(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  quick = FALSE,
  parallel = FALSE,
  sequential = FALSE,
  nss = NULL,
  nc = NULL,
  ...
)

## S3 method for class 'polr'
check_overlap(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
```

```
  quick = FALSE,
  parallel = FALSE,
  sequential = FALSE,
  nss = NULL,
  nc = NULL,
  ...
)

## S3 method for class 'multinom'
check_overlap(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  quick = FALSE,
  parallel = FALSE,
  sequential = FALSE,
  nss = NULL,
  nc = NULL,
  ...
)

## S3 method for class 'glm'
check_overlap(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  quick = FALSE,
  parallel = FALSE,
  sequential = FALSE,
  nss = NULL,
  nc = NULL,
  ...
)

## S3 method for class 'bracl'
check_overlap(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  quick = FALSE,
  parallel = FALSE,
  sequential = FALSE,
  nss = NULL,
  nc = NULL,
  ...
```

```
)

## S3 method for class 'brmultinom'
check_overlap(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  quick = FALSE,
  parallel = parallel,
  sequential = sequential,
  nss = nss,
  nc = nc,
  ...
)
```

Arguments

`object` an R object. Supported inputs:

- **Pre-fit (vector):** a vector of type `factor`, `character`, `logical`, `numeric` or `integer`. In this case one also needs to supply the argument `X` and, optionally but recommended, a `model`.
- **Pre-fit (matrix):** a matrix of structure vectors.
- **Pre-fit (formula):** an object of class `"formula"` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details' in `glm`. In this case one needs to supply `data`, with an optional `model` argument as well.
- **Post-fit:** a fitted model object of class `glm`, `polr`, `clm`, `osm`, `brmultinom`, `bracl`, `brglm` or `multinom`.

`...` additional arguments to be passed to other functions (e.g. to mclappy in parallel execution)

`X` a design matrix, e.g. generated via a call to `model.matrix`. `X` is expected to already contain the desired contrasts for factors (e.g., dummies) and any other expanded columns (e.g., for polynomials).

`rational` logical; should rational arithmetic be used?

`quick` logical; if `TRUE`, use the quick linear program variant, otherwise the full linear program.

`sequential` conduct a sequential check on data subsets (default 'FALSE').

`parallel` conduct a check in parallel on data subsets via multicores (default 'FALSE').

`backend` character string specifying the backend for the linear program. One of `"rcdd"` (default, and the only option when `rational = TRUE`) or `"ROI"`.

`solver` character string specifying the solver used by the backend. Defaults to `"DualSimplex"` for `backend = "rcdd"`, and to the first LP solver returned by `ROI_applicable_solvers` for `backend = "ROI"`.

`nc` number of cores to be used for parallel execution. Defaults to 1.

`nss` number of subsets for parallel or sequneital execution. Defaults to 'nc' for parallel and 10 for sequential. If nss is below 1 or above n-1, it uses 'nss = 1'.

`data` either a standard data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which the function is called.

Alternatively, `data` can be a data frame or matrix containing rational numbers as per the definition in **rcdd** (i.e. columns are characters, entries are either integer numbers or ratios of integer numbers, e.g. `"1"` or `"-234/19008"`). This is checked internally; see 'Details' for what happens when this structure is discovered.

`model` character string specifying the model class to be fitted. One of:

- `"b"` – binary
- `"bcl"` – baseline-category link
- `"cl"` – cumulative link
- `"acl"` – adjacent-category link
- `"sl"` – sequential link
- `"os"` – ordered stereotype model

If missing or `NULL`, defaults to `"cl"` for ordinal `y` and `"bcl"` otherwise.

`contrasts` an optional list. See the `contrasts.arg` of `model.matrix.default`. Only effective for standard data frames.

Details

`check_overlap` is an S3 generic function. For developers: If a method should be provided for the generic, it is best to have that method create a matrix of structure vectors `S` and use the low-level function `check_overlap_worker` with it.

The `formula` method is for standard data frames and formulas that work the same way as when used with `glm`. It does not support extended formulas, and may not work for functions that do formula processing differently.

For a data frame or matrix given as rational numbers in the **rcdd** definition, this is recognized but the formula does *not* get expanded and is taken literally. Consequently:

- variables in `formula` must match exactly with the column names in `data`;
- factors need to be converted to dummies beforehand (which would not be possible in the rational format in any other way anyway).

Value

A logical, 'TRUE' is overlap is detected.

Examples

```
 # pre fit

## Binary data
```

```
data(csepdat1)
outc<-csepdat1$y
desma<-cbind("(Intercept)"=1,csepdat1[,2:ncol(csepdat1)])
check_overlap(outc, X = desma, model = "b")

# Nominal data
# Baseline-category link
data(qcsepdatm)
check_overlap(y ~ x1 + x2, data = qcsepdatm, model = "bcl")

# Ordinal data
data(qcsepdato)
yo<-qcsepdato$y
Xo<-qcsepdato[,2:ncol(qcsepdato)]

# Sequential link
check_overlap(yo, Xo, model = "sl", rational = TRUE)

# Ordered stereotype
check_overlap(yo, Xo, model = "os", sequential = TRUE, nss =2)

# Adjacent-category link
check_overlap(yo, Xo, model= "acl", parallel = TRUE, nc = 1)

# Cumulative link
check_overlap(yo, Xo, model = "cl", quick = TRUE)


# post fit
m1 <- stats::glm(y~x1+x2,data=csepdat1,family=binomial())
check_overlap(m1)
```

`check_separation` *Check separation*

Description

General separation check function.

Usage

```
check_separation(object, ...)

## Default S3 method:
check_separation(object, ...)

## S3 method for class 'factor'
check_separation(
  object,
```

```
  X,
  model = NULL,
  rational = FALSE,
  quick = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'logical'
check_separation(
  object,
  X,
  model = NULL,
  rational = FALSE,
  quick = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'numeric'
check_separation(
  object,
  X,
  model = NULL,
  rational = FALSE,
  quick = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'integer'
check_separation(
  object,
  X,
  model = NULL,
  rational = FALSE,
  quick = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'character'
check_separation(
  object,
```

```
  X,
  model = NULL,
  rational = FALSE,
  quick = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'matrix'
check_separation(
  object,
  rational = FALSE,
  quick = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'formula'
check_separation(
  object,
  data,
  model = NULL,
  rational = FALSE,
  quick = FALSE,
  contrasts = NULL,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'osm'
check_separation(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  quick = FALSE,
  ...
)

## S3 method for class 'clm'
check_separation(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
```

```
  quick = FALSE,
  ...
)

## S3 method for class 'polr'
check_separation(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  quick = FALSE,
  ...
)

## S3 method for class 'multinom'
check_separation(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  quick = FALSE,
  ...
)

## S3 method for class 'glm'
check_separation(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  quick = FALSE,
  ...
)

## S3 method for class 'bracl'
check_separation(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  quick = FALSE,
  ...
)

## S3 method for class 'brmultinom'
check_separation(
  object,
  rational = FALSE,
```

```
  backend = c("rcdd", "ROI"),
  solver = NULL,
  quick = FALSE,
  ...
)
```

Arguments

`object` an R object. Supported inputs:

- **Pre-fit (vector):** a vector of type `factor`, `character`, `logical`, `numeric` or `integer`. In this case one also needs to supply the argument `X` and, optionally but recommended, a `model`.
- **Pre-fit (matrix):** a matrix of structure vectors.
- **Pre-fit (formula):** an object of class `"formula"` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details' in `glm`. In this case one needs to supply `data`, with an optional `model` argument as well.
- **Post-fit:** a fitted model object of class `glm`, `polr`, `clm`, `osm`, `brmultinom`, `bracl`, `brglm` or `multinom`.

`...` further arguments passed to the methods or lower-level functions.

`X` a design matrix, e.g. generated via a call to `model.matrix`. `X` is expected to already contain the desired contrasts for factors (e.g., dummies) and any other expanded columns (e.g., for polynomials).

`model` character string specifying the model class to be fitted. One of:

- `"b"` – binary
- `"bcl"` – baseline-category link
- `"cl"` – cumulative link
- `"acl"` – adjacent-category link
- `"sl"` – sequential link
- `"os"` – ordered stereotype model

If missing or `NULL`, defaults to `"cl"` for ordinal `y` and `"bcl"` otherwise.

`rational` logical; should rational arithmetic be used?

`quick` logical; if `TRUE`, use the quick linear program variant, otherwise the full linear program.

`backend` character string specifying the backend for the linear program. One of `"rcdd"` (default, and the only option when `rational = TRUE`) or `"ROI"`.

`solver` character string specifying the solver used by the backend. Defaults to `"DualSimplex"` for `backend = "rcdd"`, and to the first LP solver returned by `ROI_applicable_solvers` for `backend = "ROI"`.

`data` either a standard data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which the function is called.

Alternatively, `data` can be a data frame or matrix containing rational numbers as per the definition in **rcdd** (i.e. columns are characters, entries are either integer numbers or ratios of integer numbers, e.g. `"1"` or `"-234/19008"`). This is checked internally; see 'Details' for what happens when this structure is discovered.

`contrasts` an optional list. See the `contrasts.arg` of `model.matrix.default`. Only effective for standard data frames.

Details

This function checks for (quasi-)complete separation by calling the appropriate low-level functions either with linear program 1 ('quick = FALSE'; default') or linear program 2 ('quick = TRUE').

`check_separation` is an S3 generic function.

For developers: If a method should be provided for the generic, it is best to have that method create a matrix of structure vectors `S` and use the generic with the matrix.

The `formula` method is for standard data frames and formulas that work the same way as when used with `glm`. It does not support extended formulas, and may not work for functions that do formula processing differently.

For a data frame or matrix given as rational numbers in the **rcdd** definition, this is recognized but the formula does *not* get expanded and is taken literally. Consequently:

- variables in `formula` must match exactly with the column names in `data`;
- factors need to be converted to dummies beforehand (which would not be possible in the rational format in any other way anyway).

Value

A logical, 'TRUE' is separation is detected.

Examples

```
 # pre fit

## Binary data
data(csepdat1)
outc<-csepdat1$y
desma<-cbind("(Intercept)"=1,csepdat1[,2:ncol(csepdat1)])
check_separation(outc, X = desma, model = "b")

# Nominal data
# Baseline-category link
data(qcsepdatm)
y<-qcsepdatm$y
X<-cbind(1,qcsepdatm[,2:ncol(qcsepdatm)])
check_separation(y, X = X, model = "bcl")

# Ordinal data
data(qcsepdato)
yo<-qcsepdato$y
```

```
Xo<-qcsepdato[,2:ncol(qcsepdato)]

# Sequential link
check_separation(yo, Xo, model = "sl")

# Ordered stereotype
check_separation(yo, Xo, model = "os")

# Adjacent-category link
check_separation(yo, Xo, model= "acl")

# Cumulative link
check_separation(yo, Xo, model = "cl")


# post fit
m1 <- stats::glm(y~x1+x2,data=csepdat1,family=binomial())
check_separation(m1)
```

`csepdat1` *Toy data set with complete separation*

Description

Toy data with binary reponse that exhibit complete separation.

Usage

```
data(csepdat1)
```

Format

A 10 x 3 data frame:

y Binary response (factor)

x1 Numeric predictor

x2 Numeric predictor

csepdat2 — *Toy data set with complete separation*

Description

Toy data with binary reponse that exhibit complete separation.

Usage

```
data(csepdat2)
```

Format

A 10 x 3 data frame:

y Binary response (factor)

x1 Numeric predictor

x2 Numeric predictor

csepdatm — *Toy data set with complete separation*

Description

Toy data with multinomial response that exhibit complete separation.

Usage

```
data(csepdatm)
```

Format

A 10 x 3 data frame:

y Categorical response (factor) with 3 levels

x1 Numeric predictor

x2 Numeric predictor

csepdato — *Toy data set with complete separation*

Description

Toy data with ordinal response that exhibit complete separation.

Usage

```
data(csepdato)
```

Format

A 10 x 3 data frame:

y Ordered response with 3 levels

x1 Numeric predictor

x2 Numeric predictor

diagnose_separation — *Detailed separation diagnostic for all categorical outcomes.*

Description

This function checks whether there is (quasi-) complete separation, which type if any, gives the dimension of the recession cone, lists the number of columns in the design matrix that give rise to the separation as well as the columns names and lists the rows in X/S for which we have separation.

Usage

```
diagnose_separation(object, ...)

## Default S3 method:
diagnose_separation(object, ...)

## S3 method for class 'factor'
diagnose_separation(
  object,
  X,
  model = NULL,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)
```

```
## S3 method for class 'character'
diagnose_separation(
  object,
  X,
  model = NULL,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'logical'
diagnose_separation(
  object,
  X,
  model = NULL,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'numeric'
diagnose_separation(
  object,
  X,
  model = NULL,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'integer'
diagnose_separation(
  object,
  X,
  model = NULL,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'matrix'
diagnose_separation(
  object,
```

```
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'formula'
diagnose_separation(
  object,
  data,
  model = NULL,
  rational = FALSE,
  contrasts = NULL,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'osm'
diagnose_separation(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'clm'
diagnose_separation(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'polr'
diagnose_separation(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'multinom'
diagnose_separation(
  object,
```

```
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'glm'
diagnose_separation(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'bracl'
diagnose_separation(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'brmultinom'
diagnose_separation(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)
```

Arguments

`object` an R object. Supported inputs:

- **Pre-fit (vector):** a vector of type `factor`, `character`, `logical`, `numeric` or `integer`. In this case one also needs to supply the argument `X` and, optionally but recommended, a `model`.
- **Pre-fit (matrix):** a matrix of structure vectors.
- **Pre-fit (formula):** an object of class `"formula"` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details' in `glm`. In this case one needs to supply `data`, with an optional `model` argument as well.
- **Post-fit:** a fitted model object of class `glm`, `polr`, `clm`, `osm`, `brmultinom`, `bracl`, `brglm` or `multinom`.

`...` further arguments passed to the methods or lower-level functions.

`X` a design matrix, e.g. generated via a call to `model.matrix`. `X` is expected to already contain the desired contrasts for factors (e.g., dummies) and any other expanded columns (e.g., for polynomials).

`model` character string specifying the model class to be fitted. One of:

- `"b"` – binary
- `"bcl"` – baseline-category link
- `"cl"` – cumulative link
- `"acl"` – adjacent-category link
- `"sl"` – sequential link
- `"os"` – ordered stereotype model

If missing or `NULL`, defaults to `"cl"` for ordinal `y` and `"bcl"` otherwise.

`rational` logical; should rational arithmetic be used?

`backend` character string specifying the backend for the linear program. One of `"rcdd"` (default, and the only option when `rational = TRUE`) or `"ROI"`.

`solver` character string specifying the solver used by the backend. Defaults to `"DualSimplex"` for `backend = "rcdd"`, and to the first LP solver returned by `ROI_applicable_solvers` for `backend = "ROI"`.

`data` either a standard data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which the function is called.

Alternatively, `data` can be a data frame or matrix containing rational numbers as per the definition in **rcdd** (i.e. columns are characters, entries are either integer numbers or ratios of integer numbers, e.g. `"1"` or `"-234/19008"`). This is checked internally; see 'Details' for what happens when this structure is discovered.

`contrasts` an optional list. See the `contrasts.arg` of `model.matrix.default`. Only effective for standard data frames.

Details

`diagnose_separation` is an S3 generic function.

For developers: If a method should be provided for the generic, it is best to have that method create a matrix of structure vectors `S` and use the generic with the matrix.

The `formula` method is for standard data frames and formulas that work the same way as when used with `glm`. It does not support extended formulas, and may not work for functions that do formula processing differently.

For a data frame or matrix given as rational numbers in the **rcdd** definition, this is recognized but the formula does *not* get expanded and is taken literally. Consequently:

- variables in `formula` must match exactly with the column names in `data`;
- factors need to be converted to dummies beforehand (which would not be possible in the rational format in any other way anyway).

Value

an object of class 'sepmod' that is a list with the components:

- separation boolean whether there is separation ('TRUE' means separation)
- septype which type of separation (or not). A string of either "Overlap", "Quasi-Complete Separation" or "Complete Separation".
- reccdim dimension of recession cone
- offrows rows associated with separation
- nr.offcols number of columns of the design matrix that have separation
- offcols columns associated with separation

For `model = "sl"` an object of class 'sepmod_sl'. It is a list or the above lists with the elements corresponding to each category.

Examples

```
data(qcsepdatm)

## Binary data
data(csepdat1)
outc<-csepdat1$y
desma<-cbind("(Intercept)"=1,csepdat1[,2:ncol(csepdat1)])
c1<- diagnose_separation(outc, X = desma, model = "b")
print(c1)
print(c1, "full")

# Nominal data
# Baseline-category link
data(qcsepdatm)
y<-qcsepdatm$y
X<-cbind(1,qcsepdatm[,2:ncol(qcsepdatm)])
diagnose_separation(y, X = X, model = "bcl")

# Ordinal data
data(qcsepdato)
yo<-qcsepdato$y
Xo<-qcsepdato[,2:ncol(qcsepdato)]

# Sequential link
diagnose_separation(yo, Xo, model = "sl")

# Ordered stereotype
diagnose_separation(yo, Xo, model = "os")

# Adjacent-category link
diagnose_separation(yo, Xo, model= "acl")

# Cumulative link
diagnose_separation(yo, Xo, model = "cl")
```

```
#post fit
if (require('nnet')) {
m1 <- nnet::multinom(y ~ x1 + x2, data = qcsepdatm)
diagnose_separation(m1)
}
```

HDSS *Willingness to Share Health Data of Drexler (2026)*

Description

These data are a snapshot from a study on willingness to share health data in Austria. We look at the willingness to share data with a social health insurance provider for people with chronic conditions. The data is a random sample of 300 complete cases from the original 1260 observations and fully anonymized. We did the following data preparation steps over the original data: We reversed the polarity of the item variables by mapping the "worst" category to the lowest number and we also combined the two highest trust categories to trust category 4.

Usage

```
data(HDSS)
```

Format

A 300 x 5 data frame:

ID An ID (not the original one from the study).

WSTSSHI Item: Willingness to share personal health information with the social health insurance provider. An ordered factor with 1 indicating lowest willingness and 4 highest willingness.

knowledge Item: Knowledge about health data. An ordinal scale from 1 to 4 with 1 being the lowest knowledge and 4 highest knowledge.

chronic Number of chronic conditions (1 meaning "yes, one" 2 meaning "yes, many" and 3 meaning "none").

trustSHI Item: Trust in the social health insurance provider. An ordinal scale from 1 to 4 with 1 being the lowest trust and 4 highest trust.

gender A person's gender identification with levels "f" for female, "m" for male and "x" for neither.

Source

Drexler, S. (2026). Patient-reported and register-based indicators for system-wide primary care quality evaluation: An analysis of applicability, benefits, limitations, and mitigation strategies. Phd thesis, WU Vienna University of Economics and Business, Vienna, Austria.

`iris2` *Adapted Iris Dataset*

Description

This dataset is an adaptation of the famous Iris dataset. The original dataset contains measurements of iris flowers from three different species. In this adaptation, the dataset has been modified to create an ordinal response variable based on petal length, categorizing it into three groups: Low, Medium, and High, so that it shows separation. The dataset consists of five predictor variables and one ordinal response variable.

Usage

```
data(iris2)
```

Format

A data frame with 150 rows and 6 coumns/variables:

y An ordered factor representing the petal length category:

- Low: Petal length less than 2.5
- Medium: Petal length between 2.5 and 5.0
- High: Petal length greater than 5.0

Sepal.Length Numeric. The length of the sepal in centimeters.

Sepal.Width Numeric. The width of the sepal in centimeters.

Petal.Length Numeric. The length of the petal in centimeters.

Petal.Width Numeric. The width of the petal in centimeters.

Species Categorical. The species of Iris.

Source

The original Iris dataset is available as data(iris) in R

Examples

```
# View the structure of the dataset
str(iris2)
```

`linearities` *Linearities*

Description

This function calculates the linearities in the negative structure vector matrix X*, so the row vectors for which there is no separation. If this is an empty set or of length 0, then we have overlap.

Usage

```
linearities(
  y,
  X,
  S,
  rational = FALSE,
  model = c("b", "bcl", "acl", "os", "sl", "cl")
)
```

Arguments

`y`	the outcome variable. Can be factor, ordered, numeric, character or boolean. Works best if it is a factor or ordered factor. If it is not an (ordered) factor, we treat the outcome as nominal.
`X`	a design matrix, e.g. generated via a call to 'model.matrix'. This means we expect that X already contains the desired contrasts for factors (e.g., dummies) and any other expanded columns (e.g., for polynomials).
`S`	a matrix of structure vectors
`rational`	boolean flag whether rational arithmetic should be used. Default is FALSE.
`model`	what model class is intended to be fitted? Can be any of "b" for binary, "bcl" for baseline-category link, "cl" for cumulative link, "acl" for adjacent-category link. "sl" for sequential link, "os" for ordered stereotype model. If missing or NULL it defaults to cumulative link for ordinal y and baseline-category for everything else.

Value

a list with elements $lins which lists the row vectors that are linearities for any category and $index which gives the row index of the linearities. If there is complete separation they are both empty set. If their length/row dimension is number of categories*dim(X)[1], there is overlap. Anything in between is quasi-complete separation.

`make_yx` *This function is meant to set up a response variable and a design matrix from a formula-data combination for the pre-fit separation check functions.*

Description

This function is meant to set up a response variable and a design matrix from a formula-data combination for the pre-fit separation check functions.

Usage

```
make_yx(formula, data, contrasts = NULL)
```

Arguments

`formula`	An object of class '"formula"' (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details' in `glm`.
`data`	Either a standard data frame, list or environment (or object coercible by as.data.frame to a data frame) containing variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `make\_yx` is called. Alternatively, data can be a data frame or matrix containing rational numbers as per the definition in `rcdd` (i.e. columns are characters, the entries are either integer numbers or ratios of integer numbers, e.g. "1", or "-234/19008". This is checked internally; see the Details for what happens when this structure is discovered,
`contrasts`	contrasts: an optional list. See the `contrasts.arg` of `model.matrix.default`. Only effective for standard data frames.

Details

For standard data frames and formulas this function returns a list with the first element being the response variable (as specified by the left hand side of the formula) and the second element being a design matrix corresponding to the right hand side of the formula and the given optional contrasts. The design matrix is created in exactly the same way as in `glm`. For a data frame/matrix given as rational numbers in the rcdd definition, it returns a list with the first element being the response variable in rational format (as specified by the left hand side of the formula) and the second element being the character matrix in rational format corresponding to the right hand side of the formula. Note that in the latter the formula does not get expanded and is taken literally, so e.g. variables in formula must match exactly with the column names in data, or factors need to be converted to dummies before that (wouldn't be possible in the rational format in any other way anyway).

Value

A list with two elements, $y being the response variable (as specified by the left hand side of the formula) and the second element $X being a design matrix corresponding to the right hand side of the formula.

Examples

```
## standard data frame
data(nsduh2019)
frml <- her_lifetime ~ alc_agefirst + demog_age_cat6 + demog_sex
mk <- make_yx(frml,nsduh2019)
str(mk)

## rational structure
yr <- as.character(as.numeric(mk$y))
Xr <- apply(mk$X,2,as.character)
datr1 <- data.frame(yr,Xr) #data frame with rational data
frmlr1 <- yr~X.Intercept.+ alc_agefirst + demog_sexFemale
mkr1 <- make_yx(frmlr1,datr1)
str(mkr1)

datr2 <- cbind(yr,Xr) #character matrix with rational data
frmlr2 <- yr~(Intercept) + alc_agefirst #note the intercept column is labeled differently here
mkr2 <- make_yx(frmlr2,datr2)
str(mkr2)
```

`nsduh2019` *A snapshot of the NSDUH 2019 data*

Description

These data are from Nahhas (2024) to illustrate quasi-complete separation in his book Introduction to Regression Methods for Public Health Using R.

Usage

```
data(nsduh2019)
```

Format

A 843 x 4 data frame:

her_lifetime A binary factor of heroin consumption in a person's lifetime ('No' and 'Yes').

alc_agefirst Age at first use of alcohol.

demog_age_cat6 Current age category of the person.

demog_sex Sex of the person "Male" or ""Female".

Source

Nahhas R. (2024). Introduction to Regression Methods for Public Health Using R. Chapman & Hall.

`ohiovoters` *Voters in Ohio*

Description

This is a cleaned-up and anonymized random snapshot (N=6544) of the data set used in Rusch, Lee, Hornik, Jank and Zeileis (2013).

Usage

```
data(ohiovoters)
```

Format

A 6544 x 83 data frame including: * gender : Factor w/ 2 levels "F" for female,"M" for male * partyAffiliation : Factor w/ 3 levels "D" for Democrat,"R" for Republican,"U" for unaffiliated/independent * partyMix : Factor w/ 8 levels "unknown" for unkown,"allR" for all are Republican, "allD" all are Democr#' ats, "noneRorD" none are Republican or Democrat, "noneD" none are Democrat, "onlyRorD" only #' Republican or Democrat, "noneR" none are Republican, "allRorDorLegal" all are either Republi#' can or Democrats or any other legal value (e.g., other party affiliation; describes the hous#' ehold composition in terms of party affiliation * householdHead : Factor w/ 2 levels "H" for household head,"M" for household member * householdRank : Factor w/ 3 levels "3+" < "2" < "1" rank in the household * income : Factor w/ 12 levels Level of estimated household income from census or estimated using demog#' raphic and census data. "A" - Under 15,000; "B" - 15,000-24,999; "C" - 25,000-34,999; "D" #' - 35,000-49,999; "E"- 50,000-74,999; "F" - 75,000-99,999; "G" - 100,000-124,999; "H" - 125,0#' 00-149,999; "I" - 150,000-174,999; "J" - 175,000-199,999; "K" - 200,000-249,999; "L"- 250,00#' 0+ * homeOwner : Factor w/ 5 levels "1" renter,"A" renter,"B" probable renter,"C" probable homeowner,"D" home#' owner (note: it is not clear how 1 is different from A) * mailOrder : Factor w/ 2 levels "0" no ,"1" yes; indicates mail order buyer * compOwner : Factor w/ 2 levels "0"no ,"1" yes; owns a computer * educationLevel : Factor w/ 7 levels "A","B","C","D","E","F","G" after ISCED (http://www.unesco.org/education/#' information/nfsunesco/doc/isced_1997.htm; A=0 and G=6) * donateXXX : Factor w/ 2 levels "0" no,"1" yes, donated to cause XXX (arts, politics, children, animals, hea#' lth, poverty, environment, religion) * fec_XX_YY : Factor w/ 2 levels "0" for no,"1" for yes; federal contributor in years XX and YY * dontPhone : Factor w/ 3 levels "N" for no ,"U" for unknown,"Y" for yes; person is on the federal do not call list * ageD : age in days * cnt : number of attended elections * genXX : Factor w/ 2 levels "0" for no,"1" for yes; attended general elections in year XX * priXX : Factor w/ 2 levels "0" for no,"1" for yes; attended primary elections in year XX * othXX : Factor w/ 2 levels "0" for no,"1" for yes; attended other elections in year XX * pppXX : Factor w/ 2 levels "0" for no,"1" for yes; attended presidential primary elections in year XX (is the priXX variable for years with presidential elections) * cntEligible : Number of election the person was eligible to vote * percentAttended : cnt/cntEligible * ageY : age in years (ageD/365.25) * pProb : predicted probabilities from the y~sle model * pNodes : predicted node number * ageC : Factor w/ 4 levels; age category * incC : Factor w/ 3 levels "< 35k" less than 35,000 USD a year, "35k-75k" 35,001 to 75,000 USD per y#' ear ">75k more than 75,000 USD per year"; income categories * eduC : Factor w/ 3 levels "primary" primary education,"secondary" secondary education,"postsec" pos#' tsecondary eucation; education categories (collapsed educationLevel) * pSegs : Factor w/ 7 levels; predicted segment number

Source

Rusch, Lee, Hornik, Jank, Zeileis (2013). Influencing elections with statistics: Targeting voters with logistic regression trees. The Annals of Applied Statistics, 7(3), pp. 1612-1639. doi: 10.1214/13-AOAS648

---

ovldat1 *Toy data set with overlap.*

Description

Toy data with binary response that exhibit overlap and thus existence of MLE.

Usage

```
data(ovldat1)
```

Format

A 10 x 3 data frame:

y Binary response (factor)

x1 Numeric predictor

x2 Numeric predictor

---

ovldat2 *Toy data set with overlap.*

Description

Toy data with binary response that exhibit overlap and existence of MLE.

Usage

```
data(ovldat2)
```

Format

A 10 x 3 data frame:

y Binary response (factor)

x1 Numeric predictor

x2 Numeric predictor

ovldatm — *Toy data set with overlap.*

Description

Toy data with categorical response that exhibit overlap and thus existence of MLE.

Usage

```
data(ovldatm)
```

Format

A 10 x 3 data frame:

y Categorical response (factor) with 3 levels

x1 Numeric predictor

x2 Numeric predictor

ovldato — *Toy data set with overlap.*

Description

Toy data with ordered response that exhibit overlap.

Usage

```
data(ovldato)
```

Format

A 10 x 3 data frame:

y Ordered response with 3 levels

x1 Numeric predictor

x2 Numeric predictor

print.sepmod — *Print method for sepmod classes*

Description

Print method for sepmod classes

Usage

```
## S3 method for class 'sepmod'
print(x, info = c("minimal", "full"), ...)
```

Arguments

x	object of class sepmod
info	how much info is printed. Possibilities are 'minimal' (default) and 'full'.
...	additional arguments

Value

No return value, called for side effects. Returns the input object 'x' invisibly.

print.sepmod_sl — *Print method for sepmod_sl classes*

Description

Print method for sepmod_sl classes

Usage

```
## S3 method for class 'sepmod_sl'
print(x, info = c("minimal", "full"), ...)
```

Arguments

x	object of class sepmod_sl
info	how much info is printed. Possibilities are 'minimal' (default) and 'full'.
...	additional arguments

Value

No return value, called for side effects. Returns the input object 'x' invisibly.

qcsepdat1 — *Toy data set with quasi-complete separation*

Description

Toy data with binary reponse that exhibit quasi-complete separation.

Usage

```
data(qcsepdat1)
```

Format

A 10 x 3 data frame:

y Binary response (factor)

x1 Numeric predictor

x2 Numeric predictor

qcsepdat2 — *Toy data set with quasi-complete separation*

Description

Toy data with binary reponse that exhibit quasi-complete separation.

Usage

```
data(qcsepdat2)
```

Format

A 10 x 3 data frame:

y Binary response (factor)

x1 Numeric predictor

x2 Numeric predictor

qcsepdatm — *Toy data set with quasi-complete separation*

Description

Toy data with categorical response that exhibit quasi-complete separation.

Usage

```
data(qcsepdatm)
```

Format

A 10 x 3 data frame:

y Categorical response (factor) with 3 levels

x1 Numeric predictor

x2 Numeric predictor

qcsepdato — *Toy data set with quasi-complete separation*

Description

Toy data with ordered response that exhibit quasi-complete separation.

Usage

```
data(qcsepdato)
```

Format

A 10 x 3 data frame:

y Ordered response with 3 levels

x1 Numeric predictor

x2 Numeric predictor

recession_cone *Recession cone calculation*

Description

This function calculates the dimension of the recession cone and returns the recession cone.

Usage

```
recession_cone(object, ...)

## Default S3 method:
recession_cone(object, ...)

## S3 method for class 'factor'
recession_cone(object, X, model = NULL, rational = FALSE, ...)

## S3 method for class 'character'
recession_cone(object, X, model = NULL, rational = FALSE, ...)

## S3 method for class 'logical'
recession_cone(object, X, model = NULL, rational = FALSE, ...)

## S3 method for class 'numeric'
recession_cone(object, X, model = NULL, rational = FALSE, ...)

## S3 method for class 'integer'
recession_cone(object, X, model = NULL, rational = FALSE, ...)

## S3 method for class 'matrix'
recession_cone(object, rational = FALSE, ...)

## S3 method for class 'formula'
recession_cone(
  object,
  data,
  model = NULL,
  rational = FALSE,
  contrasts = NULL,
  ...
)

## S3 method for class 'osm'
recession_cone(object, rational = FALSE, ...)

## S3 method for class 'clm'
recession_cone(object, rational = FALSE, ...)
```

```
## S3 method for class 'polr'
recession_cone(object, rational = FALSE, ...)

## S3 method for class 'multinom'
recession_cone(object, rational = FALSE, ...)

## S3 method for class 'glm'
recession_cone(object, rational = FALSE, ...)

## S3 method for class 'bracl'
recession_cone(object, rational = FALSE, ...)

## S3 method for class 'brmultinom'
recession_cone(object, rational = FALSE, ...)
```

Arguments

`object` an R object. Supported inputs:

- **Pre-fit (vector):** a vector of type `factor`, `character`, `logical`, `numeric` or `integer`. In this case one also needs to supply the argument `X` and, optionally but recommended, a `model`.
- **Pre-fit (matrix):** a matrix of structure vectors.
- **Pre-fit (formula):** an object of class `"formula"` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details' in `glm`. In this case one needs to supply `data`, with an optional `model` argument as well.
- **Post-fit:** a fitted model object of class `glm`, `polr`, `clm`, `osm`, `brmultinom`, `bracl`, `brglm` or `multinom`.

`...` further arguments passed to the methods or lower-level functions.

`X` a design matrix, e.g. generated via a call to `model.matrix`. `X` is expected to already contain the desired contrasts for factors (e.g., dummies) and any other expanded columns (e.g., for polynomials).

`model` character string specifying the model class to be fitted. One of:

- `"b"` – binary
- `"bcl"` – baseline-category link
- `"cl"` – cumulative link
- `"acl"` – adjacent-category link
- `"sl"` – sequential link
- `"os"` – ordered stereotype model

If missing or `NULL`, defaults to `"cl"` for ordinal `y` and `"bcl"` otherwise.

`rational` logical; should rational arithmetic be used.

`data` either a standard data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`,

the variables are taken from `environment(formula)`, typically the environment from which the function is called.

Alternatively, `data` can be a data frame or matrix containing rational numbers as per the definition in **rcdd** (i.e. columns are characters, entries are either integer numbers or ratios of integer numbers, e.g. `"1"` or `"-234/19008"`). This is checked internally; see 'Details' for what happens when this structure is discovered.

`contrasts` an optional list. See the `contrasts.arg` of `model.matrix.default`. Only effective for standard data frames.

Details

Note that in case of non full column rank, the 'reccdim' value is the dimension of the recession cone due to separation plus the number of columns that are linear dependent.

The `formula` method is for standard data frames and formulas that work the same way as when used with `glm`. It does not support extended formulas, and may not work for functions that do formula processing differently.

For a data frame or matrix given as rational numbers in the **rcdd** definition, this is recognized but the formula does *not* get expanded and is taken literally. Consequently:

- variables in `formula` must match exactly with the column names in `data`;
- factors need to be converted to dummies beforehand (which would not be possible in the rational format in any other way anyway).

Value

A list with the components:

- cone being the recession cone,
- reccdim being the dimensions of the recession cone
- index the row index of the structure vectors that are not linearities.

For `model = "sl"` 'cone' is the recession cone over all categories, 'reccdim' the dimension of the largest recession cone of any category and 'index' the row index of the structure vectors that are not linearities over all categories.

Examples

```
## Binary data
data(csepdat1)
outc<-csepdat1$y
desma<-cbind("(Intercept)"=1,csepdat1[,2:ncol(csepdat1)])
recession_cone(outc, X = desma, model = "b")

# Nominal data
# Baseline-category link
data(qcsepdatm)
y<-qcsepdatm$y
X<-cbind(1,qcsepdatm[,2:ncol(qcsepdatm)])
```

```
recession_cone(y, X = X, model = "bcl")

# Ordinal data
data(qcsepdato)
yo<-qcsepdato$y
Xo<-qcsepdato[,2:ncol(qcsepdato)]

# Sequential link
recession_cone(yo, Xo, model = "sl")

# Ordered stereotype
recession_cone(yo, Xo, model = "os")

# Adjacent-category link
recession_cone(yo, Xo, model= "acl")

# Cumulative link
recession_cone(yo, Xo, model = "cl")

# post fit
if (require('ordinal')) {
m1 <- ordinal::clm(y~x1+x2, data = qcsepdato)
recession_cone(m1)
}
```

`separation_columns` *Identify separation columns*

Description

This function identifies the columns in a design matrix/structure vector matrix that are responsible for separation. It calls lower level functions if given an argument or chooses based on the response type.

Usage

```
separation_columns(object, ...)

## Default S3 method:
separation_columns(object, ...)

## S3 method for class 'factor'
separation_columns(
  object,
  X,
  model = NULL,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
```

```
  ...
)

## S3 method for class 'character'
separation_columns(
  object,
  X,
  model = NULL,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'logical'
separation_columns(
  object,
  X,
  model = NULL,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'numeric'
separation_columns(
  object,
  X,
  model = NULL,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'integer'
separation_columns(
  object,
  X,
  model = NULL,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'matrix'
```

```
separation_columns(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'formula'
separation_columns(
  object,
  data,
  model = NULL,
  rational = FALSE,
  contrasts = NULL,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'osm'
separation_columns(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'clm'
separation_columns(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'polr'
separation_columns(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'multinom'
```

```
separation_columns(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'glm'
separation_columns(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'bracl'
separation_columns(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)

## S3 method for class 'brmultinom'
separation_columns(
  object,
  rational = FALSE,
  backend = c("rcdd", "ROI"),
  solver = NULL,
  ...
)
```

Arguments

`object` an R object. Supported inputs:

- **Pre-fit (vector):** a vector of type `factor`, `character`, `logical`, `numeric` or `integer`. In this case one also needs to supply the argument `X` and, optionally but recommended, a `model`.
- **Pre-fit (matrix):** a matrix of structure vectors.
- **Pre-fit (formula):** an object of class `"formula"` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details' in `glm`. In this case one needs to supply `data`, with an optional `model` argument as well.
- **Post-fit:** a fitted model object of class `glm`, `polr`, `clm`, `osm`, `brmultinom`, `bracl`, `brglm` or `multinom`.

`...` further arguments passed to the methods or lower-level functions.

`X` a design matrix, e.g. generated via a call to `model.matrix`. `X` is expected to already contain the desired contrasts for factors (e.g., dummies) and any other expanded columns (e.g., for polynomials).

`model` character string specifying the model class to be fitted. One of:

- `"b"` – binary
- `"bcl"` – baseline-category link
- `"cl"` – cumulative link
- `"acl"` – adjacent-category link
- `"sl"` – sequential link
- `"os"` – ordered stereotype model

If missing or `NULL`, defaults to `"cl"` for ordinal `y` and `"bcl"` otherwise.

`rational` logical; should rational arithmetic be used.

`backend` character string specifying the backend for the linear program. One of `"rcdd"` (default, and the only option when `rational = TRUE`) or `"ROI"`.

`solver` character string specifying the solver used by the backend. Defaults to `"DualSimplex"` for `backend = "rcdd"`, and to the first LP solver returned by `ROI_applicable_solvers` for `backend = "ROI"`.

`data` either a standard data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which the function is called.

Alternatively, `data` can be a data frame or matrix containing rational numbers as per the definition in **rcdd** (i.e. columns are characters, entries are either integer numbers or ratios of integer numbers, e.g. `"1"` or `"-234/19008"`). This is checked internally; see 'Details' for what happens when this structure is discovered.

`contrasts` an optional list. See the `contrasts.arg` of `model.matrix.default`. Only effective for standard data frames.

Details

`separation_columns` is an S3 generic function.

For developers: If a method should be provided for the generic, it is best to have that method create a matrix of structure vectors `S` and use the generic with the matrix.

The `formula` method is for standard data frames and formulas that work the same way as when used with `glm`. It does not support extended formulas, and may not work for functions that do formula processing differently.

For a data frame or matrix given as rational numbers in the **rcdd** definition, this is recognized but the formula does *not* get expanded and is taken literally. Consequently:

- variables in `formula` must match exactly with the column names in `data`;
- factors need to be converted to dummies beforehand (which would not be possible in the rational format in any other way anyway).

Value

A list with the components:

- ls the solution vector of the linear program
- offcols the names of the columns that show separation
- colnrs the index number of the columns that show separation
- separated a logical vector of whetehr the colum shows separation

For `model = "sl"` it is a list of these lists for each category.

Examples

```
#' ## Binary data
data(csepdat1)
outc<-csepdat1$y
desma<-cbind("(Intercept)"=1,csepdat1[,2:ncol(csepdat1)])
separation_columns(outc, X = desma, model = "b")

# Nominal data
# Baseline-category link
data(qcsepdatm)
y<-qcsepdatm$y
X<-cbind(1,qcsepdatm[,2:ncol(qcsepdatm)])
separation_columns(y, X = X, model = "bcl")

# Ordinal data
data(qcsepdato)
yo<-qcsepdato$y
Xo<-qcsepdato[,2:ncol(qcsepdato)]

# Sequential link
separation_columns(yo, Xo, model = "sl")

# Ordered stereotype
separation_columns(yo, Xo, model = "os")

# Adjacent-category link
separation_columns(yo, Xo, model= "acl")

# Cumulative link
separation_columns(yo, Xo, model = "cl")


# post fit
if (require('clustord')) {
m1 <- clustord::osm(y~x1+x2, data = qcsepdato)
separation_columns(m1)
}
```

separation_rows *Identify separation rows*

Description

This function identifies the rows in a design matrix/structure vector matrix that are associated for separation. It calls lower level functions if given an argument or chooses based on the response type.

Usage

```
separation_rows(object, ...)

## Default S3 method:
separation_rows(object, ...)

## S3 method for class 'factor'
separation_rows(object, X, model = NULL, rational = FALSE, ...)

## S3 method for class 'character'
separation_rows(object, X, model = NULL, rational = FALSE, ...)

## S3 method for class 'logical'
separation_rows(object, X, model = NULL, rational = FALSE, ...)

## S3 method for class 'numeric'
separation_rows(object, X, model = NULL, rational = FALSE, ...)

## S3 method for class 'integer'
separation_rows(object, X, model = NULL, rational = FALSE, ...)

## S3 method for class 'matrix'
separation_rows(object, rational = FALSE, ...)

## S3 method for class 'formula'
separation_rows(
  object,
  data,
  model = NULL,
  rational = FALSE,
  contrasts = NULL,
  ...
)

## S3 method for class 'osm'
separation_rows(object, rational = FALSE, ...)
```

```
## S3 method for class 'clm'
separation_rows(object, rational = FALSE, ...)

## S3 method for class 'polr'
separation_rows(object, rational = FALSE, ...)

## S3 method for class 'multinom'
separation_rows(object, rational = FALSE, ...)

## S3 method for class 'glm'
separation_rows(object, rational = FALSE, ...)

## S3 method for class 'bracl'
separation_rows(object, rational = FALSE, ...)

## S3 method for class 'brmultinom'
separation_rows(object, rational = FALSE, ...)
```

Arguments

`object` an R object. Supported inputs:

- **Pre-fit (vector):** a vector of type `factor`, `character`, `logical`, `numeric` or `integer`. In this case one also needs to supply the argument `X` and, optionally but recommended, a `model`.
- **Pre-fit (matrix):** a matrix of structure vectors.
- **Pre-fit (formula):** an object of class `"formula"` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details' in `glm`. In this case one needs to supply `data`, with an optional `model` argument as well.
- **Post-fit:** a fitted model object of class `glm`, `polr`, `clm`, `osm`, `brmultinom`, `bracl`, `brglm` or `multinom`.

`...` further arguments passed to the methods or lower-level functions.

`X` a design matrix, e.g. generated via a call to `model.matrix`. `X` is expected to already contain the desired contrasts for factors (e.g., dummies) and any other expanded columns (e.g., for polynomials).

`model` character string specifying the model class to be fitted. One of:

- `"b"` – binary
- `"bcl"` – baseline-category link
- `"cl"` – cumulative link
- `"acl"` – adjacent-category link
- `"sl"` – sequential link
- `"os"` – ordered stereotype model

If missing or `NULL`, defaults to `"cl"` for ordinal `y` and `"bcl"` otherwise.

`rational` logical; should rational arithmetic be used.

`data` either a standard data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which the function is called.

Alternatively, `data` can be a data frame or matrix containing rational numbers as per the definition in **rcdd** (i.e. columns are characters, entries are either integer numbers or ratios of integer numbers, e.g. `"1"` or `"-234/19008"`). This is checked internally; see 'Details' for what happens when this structure is discovered.

`contrasts` an optional list. See the `contrasts.arg` of `model.matrix.default`. Only effective for standard data frames.

Details

`separation_rows` is S3 generic. For developers: If a method should be provided for the generic, it is best to have that method create a matrix of structure vectors `S` and use the low-level function `seprows_worker` with it.

The `formula` method is for standard data frames and formulas that work the same way as when used with `glm`. It does not support extended formulas, and may not work for functions that do formula processing differently.

For a data frame or matrix given as rational numbers in the **rcdd** definition, this is recognized but the formula does *not* get expanded and is taken literally. Consequently:

- variables in `formula` must match exactly with the column names in `data`;
- factors need to be converted to dummies beforehand (which would not be possible in the rational format in any other way anyway).

Value

A list with the components:

- offrows the submatrix of the matrix (X,y) with the rows responsible for separation
- index the index of the rows responsible for separation

For `model = "sl"` it is a list of these lists for each category.

Examples

```
## Binary data
data(csepdat1)
outc<-csepdat1$y
desma<-cbind("(Intercept)"=1,csepdat1[,2:ncol(csepdat1)])
separation_rows(outc, X = desma, model = "b")

# Nominal data
# Baseline-category link
data(qcsepdatm)
y<-qcsepdatm$y
X<-cbind(1,qcsepdatm[,2:ncol(qcsepdatm)])
```

```
separation_rows(y, X = X, model = "bcl")

# Ordinal data
data(qcsepdato)
yo<-qcsepdato$y
Xo<-qcsepdato[,2:ncol(qcsepdato)]

# Sequential link
separation_rows(yo, Xo, model = "sl")

# Ordered stereotype
separation_rows(yo, Xo, model = "os")

# Adjacent-category link
separation_rows(yo, Xo, model= "acl")

# Cumulative link
separation_rows(yo, Xo, model = "cl")

# post fit
if (require('MASS')) {
m1 <- MASS::polr(y~x1+x2, data = qcsepdato)
separation_rows(m1)
}
```

Silvapulle *Psychiatric Cases Classification based on GHQ of Silvapulle (1981)*

Description

These data are from a psychiatric study of for 120 patients on the relation between the value of the score on the 12-item General Health Questionnaire (GHQ) and classification as a psychiatric case ("yes" or "no"). The patients attended as general practitioner's surgery and were administered the GHQ, resulting in a theoretical score between 0 and 12 (there were no cases or non-cases with GHQ scores of 11 or 12). Subsequently the patinets were given a full psychiatric examination by a psychiatrist who did not know the patient's GHQ score. The patient was classified by the psychiatrist as either a clinical case requiring psychiatric treatment (case="yes"), or as non-clinial (case="no"). The biological sex of the patients was also recorded.

Usage

```
data(Silvapulle)
```

Format

A 120 x 3 data frame:

sex A factor with levels 'male' and 'female'.

ghq The GHQ score on a scale from 0 to 12 (numeric).

case Diagnosis as a clinical case ('yes') or not ('no').

Source

Silvapulle, M. J. (1981), "On the existence of maximum likelihood estimators for the binomial response model", J. Roy. Statist. Soc. B., 43, 310-313.

`structure_vectors` *Structure Vector S3 Generic*

Description

Generic function to compute structure vectors from different types of inputs.

Method for `structure_vectors` when the input is a formula and data frame.

Usage

```
structure_vectors(x, ...)

## Default S3 method:
structure_vectors(
  x,
  X,
  model = c("b", "bcl", "acl", "os", "sl", "cl"),
  label = TRUE,
  rational = FALSE,
  ...
)

## S3 method for class 'formula'
structure_vectors(
  x,
  data,
  contrasts = NULL,
  model = NULL,
  label = TRUE,
  rational = FALSE,
  ...
)
```

Arguments

`x` input object. Supported inputs:

- **Pre-fit (default):** a vector of type `factor`, `character`, `logical`, `numeric` or `integer`. In this case one also needs to supply the argument `X` and, optionally but recommended, a `model`.
- **Pre-fit (formula):** an object of class `"formula"` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details' in `glm`. In this case one needs to supply `data`, with an optional `model` argument as well.

`...`	Additional arguments.
`X`	a design matrix, e.g. generated via a call to `model.matrix`. `X` is expected to already contain the desired contrasts for factors (e.g., dummies) and any other expanded columns (e.g., for polynomials).
`model`	character string specifying the model class to be fitted. One of: • `"b"` – binary • `"bcl"` – baseline-category link • `"cl"` – cumulative link • `"acl"` – adjacent-category link • `"sl"` – sequential link • `"os"` – ordered stereotype model If missing or `NULL`, defaults to `"cl"` for ordinal `y` and `"bcl"` otherwise.
`label`	Should the columns and rows be labeled?
`rational`	Should the matrix be returned in rational form (as per rcdd definition)?
`data`	either a standard data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which the function is called. Alternatively, `data` can be a data frame or matrix containing rational numbers as per the definition in **rcdd** (i.e. columns are characters, entries are either integer numbers or ratios of integer numbers, e.g. `"1"` or `"-234/19008"`). This is checked internally; see 'Details' for what happens when this structure is discovered.
`contrasts`	an optional list. See the `contrasts.arg` of `model.matrix.default`. Only effective for standard data frames.

Details

`structure_vectors` is an S3 generic function.

The `formula` method is for standard data frames and formulas that work the same way as when used with `glm`. It does not support extended formulas, and may not work for functions that do formula processing differently.

For a data frame or matrix given as rational numbers in the **rcdd** definition, this is recognized but the formula does *not* get expanded and is taken literally. Consequently:

- variables in `formula` must match exactly with the column names in `data`;
- factors need to be converted to dummies beforehand (which would not be possible in the rational format in any other way anyway).

Value

A structure vector matrix with or without labeled rows and columns. For `model = "sl"`, a list of structure vector matrices, where each list element corresponds sequentially to the categories of y, starting with the lowest and ending with the (K-1)-th category. At each category k, we consider all observations with category k or higher for the structure vector matrix.

Examples

```
data(qcsepdato)
yo<-qcsepdato$y
Xo<-qcsepdato[,2:ncol(qcsepdato)]

# Sequential link (default method and no labelling)
structure_vectors(yo, Xo, model = "sl", label = FALSE)

# Ordered stereotype (formula method)
structure_vectors(y ~ x1 + x2, data = qcsepdato, model = "os")
```

---

titanic3 *All people of age 3 in the Kaggle Titanic data set*

Description

This data set contains the subset of people in the Titanic data sets that were aged 3. It shows various separation types depending on the model fitted.

Usage

```
data(titanic3)
```

Format

A 6 x 12 data frame:

passengerId ID of the passenger

Survived Survived? (1=yes, 0=no)

Pclass Passenger Class (1=upper, 2=middle, 3=lower).

Name Name.

Sex Sex of the person.

Age Age of the person. All are 3 yrs.

SibSp Number of siblings or spouses on board the ship.

Parch Number of parents or children on board the ship.

Ticket Ticket number.

Fare Passenger fare.

Cabin Cabin number.

Embarked Port of embarkation (C=Cherbourg, Q=Queenstown, S=Southampton).

Source

Will Cukierski. Titanic - Machine Learning from Disaster. https://kaggle.com/competitions/titanic, 2012. Kaggle.

Examples

```
with(titanic3,ftable(Survived~Sex+Pclass))
```

Index

∗ **datasets**
- Alligators, 3
- csepdat1, 16
- csepdat2, 17
- csepdatm, 17
- csepdato, 18
- HDSS, 24
- iris2, 25
- nsduh2019, 28
- ohiovoters, 29
- ovldat1, 30
- ovldat2, 30
- ovldatm, 31
- ovldato, 31
- qcsepdat1, 33
- qcsepdat2, 33
- qcsepdatm, 34
- qcsepdato, 34
- Silvapulle, 47
- titanic3, 50

Alligators, 3
as.data.frame, *9*, *14*, *22*, *36*, *42*, *46*, *49*

check_overlap, 3
check_separation, 10
csepdat1, 16
csepdat2, 17
csepdatm, 17
csepdato, 18

diagnose_separation, 18

glm, *8, 9*, *14, 15*, *21, 22*, *27*, *36, 37*, *41, 42*, *45, 46*, *48, 49*

HDSS, 24

iris2, 25

linearities, 26

make_yx, 27
model.matrix, *8*, *14*, *22*, *36*, *42*, *45*, *49*
model.matrix.default, *9*, *15*, *22*, *37*, *42*, *46*, *49*

nsduh2019, 28

ohiovoters, 29
ovldat1, 30
ovldat2, 30
ovldatm, 31
ovldato, 31

print.sepmod, 32
print.sepmod_sl, 32

qcsepdat1, 33
qcsepdat2, 33
qcsepdatm, 34
qcsepdato, 34

recession_cone, 35
ROI_applicable_solvers, *8*, *14*, *22*, *42*

separation_columns, 38
separation_rows, 44
Silvapulle, 47
structure_vectors, 48

titanic3, 50