

Package ‘modeltools’

September 30, 2026

Title Tools and Classes for Statistical Models

Version 0.2-25

Description A collection of tools to deal with statistical models.

The functionality is experimental and the user interface is likely to change in the future. The documentation is rather terse, but packages ‘coin’ and ‘party’ have some working examples. However, if you find the implemented ideas interesting we would be very interested in a discussion of this proposal. Contributions are more than welcome!

Depends stats, stats4

Imports methods

LazyLoad yes

License GPL-2

NeedsCompilation no

URL <https://codeberg.org/thothorn/modeltools>

BugReports <https://codeberg.org/thothorn/modeltools/issues>

Author Torsten Hothorn [aut, cre] (ORCID:

<https://orcid.org/0000-0001-8301-0471>),

Friedrich Leisch [aut] (ORCID: <https://orcid.org/0000-0001-7278-1983>),

Achim Zeileis [aut] (ORCID: <https://orcid.org/0000-0003-0918-3766>)

Maintainer Torsten Hothorn <Torsten.Hothorn@R-project.org>

Repository CRAN

Date/Publication 2026-09-30 20:00:08 UTC

Contents

FormulaParts-class	2
Generics	2
info	3
MEapply	4
ModelEnv-class	5
ModelEnvFormula	6

ModelEnvFormula-class	8
ModelEnvMatrix	9
Predict	10
StatModel-class	11
StatModelCapabilities-class	12
Index	13

FormulaParts-class	<i>Class "FormulaParts"</i>
--------------------	-----------------------------

Description

A class describing the parts of a formula.

Objects from the Class

Objects can be created by calls of the form `new("FormulaParts", ...)`.

Slots

`formula`: Object of class "list".

Methods

No methods defined with class "FormulaParts" in the signature.

Generics	<i>Generic Utility Functions</i>
----------	----------------------------------

Description

A collection of standard generic functions for which other packages provide methods.

Usage

```
ICL(object, ...)
KLdiv(object, ...)
Lapply(object, FUN, ...)
clusters(object, newdata, ...)
getModel(object, ...)
parameters(object, ...)
posterior(object, newdata, ...)
prior(object, ...)
refit(object, newdata, ...)
relabel(object, by, ...)
ParseFormula(formula, data = list())
```

Arguments

<code>object</code>	S4 classed object.
<code>formula</code>	A model formula.
<code>data</code>	An optional data frame.
<code>FUN</code>	The function to be applied.
<code>newdata</code>	Optional new data.
<code>by</code>	Typically a character string specifying how to relabel the object.
<code>...</code>	Some methods for these generic function may take additional, optional arguments.

Details

ICL: Integrated Completed Likelihood criterion for model selection.

KLdiv: Kullback-Leibler divergence.

Lapply: S4 generic for `lapply`

clusters: Get cluster membership information from a model or compute it for new data.

getModel: Get single model from a collection of models.

parameters: Get parameters of a model (similar to but more general than `coefficients`).

posterior: Get posterior probabilities from a model or compute posteriors for new data.

prior: Get prior probabilities from a model.

refit: Refit a model (usually to obtain additional information that was not computed or stored during the initial fitting process).

relabel: Relabel a model (usually to obtain a new permutation of labels in mixture models or cluster objects).

Author(s)

Friedrich Leisch

 info

Get Information on Fitted Objects

Description

Returns descriptive information about fitted objects.

Usage

```
info(object, which, ...)
## S4 method for signature 'ANY,missing'
info(object, which, ...)
infoCheck(object, which, ...)
```

Arguments

object	fitted object.
which	which information to get. Use which="help" to list available information.
...	passed to methods.

Details

Function info can be used to access slots of fitted objects in a portable way.

Function infoCheck returns a logical value that is TRUE if the requested information can be computed from the object.

Author(s)

Friedrich Leisch

MEapply

Apply functions to Data in Object of Class "ModelEnv"

Description

Apply a single function or a collection of functions to the data objects stored in a model environment.

Usage

```
## S4 method for signature 'ModelEnv'
MEapply(object, FUN, clone = TRUE, ...)
```

Arguments

object	Object of class "ModelEnv".
FUN	Function or list of functions.
clone	If TRUE, return a clone of the original object, if FALSE, modify the object itself.
...	Passed on to FUN.

Examples

```
data("iris")
me <- ModelEnvFormula(Species~Petal.Width~-1, data=iris,
                      subset=sample(1:150, 10))

me1 <- MEapply(me, FUN=list(designMatrix=scale,
                           response=function(x) sapply(x, as.numeric)))

me@get("designMatrix")
me1@get("designMatrix")
```

ModelEnv-class

Class "ModelEnv"

Description

A class for model environments.

Details

Objects of class ModelEnv basically consist of an [environment](#) for data storage as well as `get` and `set` methods.

`na.fail` returns FALSE when at least one missing value occurs in `object@env`. `na.pass` returns object unchanged and `na.omit` returns a copy of object with all missing values removed.

Objects from the Class

Objects can be created by calls of the form `new("ModelEnv", ...)`.

Slots

`env`: Object of class "environment".

`get`: Object of class "function" for extracting objects from environment `env`.

`set`: Object of class "function" for setting object in environment `env`.

`hooks`: A list of hook collections.

Methods

clone signature(object = "ModelEnv"): copy an object.

dimension signature(object = "ModelEnv", which = "character"): get the dimension of an object.

empty signature(object = "ModelEnv"): Return TRUE, if the model environment contains no data.

has signature(object = "ModelEnv", which = "character"): check if an object which is available in `env`.

initialize signature(.Object = "ModelEnv"): setup new objects.

show signature(object = "ModelEnv"): show object.

subset signature(x = "ModelEnv"): extract subsets from an object.

na.pass [na.action](#) method for ModelEnv objects.

na.fail [na.action](#) method for ModelEnv objects.

na.omit [na.action](#) method for ModelEnv objects.

Examples

```
### a new object
me <- new("ModelEnv")

## the new model environment is empty
empty(me)

### define a bivariate response variable
me@set("response", data.frame(y = rnorm(10), x = runif(10)))
me

## now it is no longer empty
empty(me)

### check if a response is available
has(me, "response")

### the dimensions
dimension(me, "response")

### extract the data
me@get("response")

df <- data.frame(x = rnorm(10), y = rnorm(10))

## hook for set method:
mf <- ModelEnvFormula(y ~ x-1, data = df, setHook=list(designMatrix=scale))
mf@get("designMatrix")
mf@set(data=df[1:5,])
mf@get("designMatrix")

### NA handling
df$x[1] <- NA
mf <- ModelEnvFormula(y ~ x, data = df, na.action = na.pass)
mf
na.omit(mf)
```

ModelEnvFormula	<i>Generate a model environment from a classical formula based interface.</i>
-----------------	---

Description

A flexible implementation of the classical formula based interface.

Usage

```
ModelEnvFormula(formula, data = list(), subset = NULL,
                 na.action = NULL, frame = NULL,
```

```
enclos = sys.frame(sys.nframe()), other = list(),
designMatrix = TRUE, responseMatrix = TRUE,
setHook = NULL, ...)
```

Arguments

<code>formula</code>	a symbolic description of the model to be fit.
<code>data</code>	an optional data frame containing the variables in the model. If not found in data, the variables are taken from <code>frame</code> , by default the environment from which <code>ModelEnvFormula</code> is called.
<code>subset</code>	an optional vector specifying a subset of observations to be used in the fitting process.
<code>na.action</code>	a function which indicates what should happen when the data contain NA's.
<code>frame</code>	an optional environment formula is evaluated in.
<code>enclos</code>	specifies the enclosure passed to <code>eval</code> for evaluating the model frame. The model frame is evaluated in <code>envir = frame</code> with <code>enclos = enclos</code> , see <code>eval</code> .
<code>other</code>	an optional named list of additional formulae.
<code>designMatrix</code>	a logical indicating whether the design matrix defined by the right hand side of formula should be computed.
<code>responseMatrix</code>	a logical indicating whether the design matrix defined by the left hand side of formula should be computed.
<code>setHook</code>	a list of functions to <code>MEapply</code> every time <code>set</code> is called on the object.
<code>...</code>	additional arguments for be passed to function, for example <code>contrast.arg</code> to <code>model.matrix</code> .

Details

This function is an attempt to provide a flexible infrastructure for the implementation of classical formula based interfaces. The arguments `formula`, `data`, `subset` and `na.action` are well known and are defined in the same way as in `lm`, for example.

`ModelEnvFormula` returns an object of class `ModelEnvFormula-class` - a high level object for storing data improving upon the capabilities of `data.frames`.

Value

An object of class `ModelEnvFormula-class`.

Examples

```
### the `usual' interface
data(iris)
mf <- ModelEnvFormula(Species ~ ., data = iris)
mf

### extract data from the ModelEnv object
summary(mf@get("response"))
summary(mf@get("input"))
```

```

dim(mf@get("designMatrix"))

### contrasts
mf <- ModelEnvFormula(Petal.Width ~ Species, data = iris,
                      contrasts.arg = list(Species = contr.treatment))
attr(mf@get("designMatrix"), "contrasts")
mf <- ModelEnvFormula(Petal.Width ~ Species, data = iris,
                      contrasts.arg = list(Species = contr.sum))
attr(mf@get("designMatrix"), "contrasts")

### additional formulae
mf <- ModelEnvFormula(Petal.Width ~ Species, data = iris,
                      other = list(pl = ~ Petal.Length))
ls(mf@env)
identical(mf@get("pl")[[1]], iris[["Petal.Length"]])

```

ModelEnvFormula-class *Class "ModelEnvFormula"*

Description

A class for formula-based model environments.

Objects from the Class

Objects can be created by calls of the form `new("ModelEnvFormula", ...)`.

Slots

env: Object of class "environment".
get: Object of class "function" for extracting objects from environment env.
set: Object of class "function" for setting object in environment env.
formula: Object of class "list".
hooks: A list of hook collections.

Extends

Class "ModelEnv", directly. Class "FormulaParts", directly.

Methods

No methods defined with class "ModelEnvFormula" in the signature.

ModelEnvMatrix	<i>Generate a model environment from design and response matrix</i>
----------------	---

Description

A simple model environment creator function working off matrices for input and response. This is much simpler and more limited than formula-based environments, but faster and easier to use, if only matrices are allowed as input.

Usage

```
ModelEnvMatrix(designMatrix=NULL, responseMatrix=NULL,
               subset = NULL, na.action = NULL, other=list(), ...)
```

Arguments

designMatrix	design matrix of input
responseMatrix	matrix of responses
subset	an optional vector specifying a subset of observations to be used in the fitting process.
na.action	a function which indicates what should happen when the data contain NA's.
other	an optional named list of additional formulae.
...	currently not used

Details

ModelEnvMatrix returns an object of class [ModelEnv-class](#) - a high level object for storing data improving upon the capabilities of simple data matrices.

Funny things may happen if the input and response matrices do not have distinct column names and the data new data are supplied via the get and set slots.

Value

An object of class [ModelEnv-class](#).

Examples

```
### use Sepal measurements as input and Petal as response
data(iris)
me <- ModelEnvMatrix(iris[,1:2], iris[,3:4])
me

### extract data from the ModelEnv object
dim(me@get("designMatrix"))
summary(me@get("responseMatrix"))
```

```

### subsets and missing values
iris[1,1] <- NA
me <- ModelEnvMatrix(iris[,1:2], iris[,3:4], subset=1:5, na.action=na.omit)

## First case is not complete, so me contains only cases 2:5
me
me@get("designMatrix")
me@get("responseMatrix")

## use different cases
me@set(data=iris[10:20,])
me@get("designMatrix")

## these two should be the same
stopifnot(all.equal(me@get("responseMatrix"), as.matrix(iris[10:20,3:4])))

```

Predict

Model Predictions

Description

A function for predictions from the results of various model fitting functions.

Usage

```
Predict(object, ...)
```

Arguments

object	a model object for which prediction is desired.
...	additional arguments affecting the predictions produced.

Details

A somewhat improved version of [predict](#) for models fitted with objects of class [StatModel-class](#).

Value

Should return a vector of the same type as the response variable specified for fitting object.

Examples

```

df <- data.frame(x = runif(10), y = rnorm(10))
mf <- dpp(linearModel, y ~ x, data = df)
Predict(fit(linearModel, mf))

```

StatModel-class	Class "StatModel"
-----------------	-------------------

Description

A class for unfitted statistical models.

Objects from the Class

Objects can be created by calls of the form `new("StatModel", ...)`.

Slots

name: Object of class "character", the name of the model.
dpp: Object of class "function", a function for data preprocessing (usually formula-based).
fit: Object of class "function", a function for fitting the model to data.
predict: Object of class "function", a function for computing predictions.
capabilities: Object of class "StatModelCapabilities".

Methods

fit `signature(model = "StatModel", data = "ModelEnv")`: fit model to data.

Details

This is an attempt to provide unified infra-structure for unfitted statistical models. Basically, an unfitted model provides a function for data pre-processing (`dpp`, think of generating design matrices), a function for fitting the specified model to data (`fit`), and a function for computing predictions (`predict`).

Examples for such unfitted models are provided by `linearModel` and `glinearModel` which provide interfaces in the "StatModel" framework to `lm.fit` and `glm.fit`, respectively. The functions return objects of S3 class "linearModel" (inheriting from "lm") and "glinearModel" (inheriting from "glm"), respectively. Some methods for S3 generics such as `predict`, `fitted`, `print` and `model.matrix` are provided to make use of the "StatModel" structure. (Similarly, `survReg` provides an experimental interface to `survreg`.)

Examples

```
### linear model example
df <- data.frame(x = runif(10), y = rnorm(10))
mf <- dpp(linearModel, y ~ x, data = df)
mylm <- fit(linearModel, mf)

### equivalent
print(mylm)
lm(y ~ x, data = df)
```

```
### predictions  
Predict(mylm, newdata = data.frame(x = runif(10)))
```

```
StatModelCapabilities-class  
Class "StatModelCapabilities"
```

Description

A class describing capabilities of a statistical model.

Objects from the Class

Objects can be created by calls of the form `new("StatModelCapabilities", ...)`.

Slots

`weights`: Object of class "logical"
`subset`: Object of class "logical"

Methods

No methods defined with class "StatModelCapabilities" in the signature.

Index

- * **classes**
 - FormulaParts-class, 2
 - ModelEnv-class, 5
 - ModelEnvFormula-class, 8
 - StatModel-class, 11
 - StatModelCapabilities-class, 12
- * **methods**
 - Generics, 2
 - info, 3
 - MEapply, 4
- * **misc**
 - ModelEnvFormula, 6
 - ModelEnvMatrix, 9
 - Predict, 10
- clone (ModelEnv-class), 5
- clone, ModelEnv-method (ModelEnv-class), 5
- clusters (Generics), 2
- coefficients, 3
- dimension (ModelEnv-class), 5
- dimension, ModelEnv, character-method (ModelEnv-class), 5
- dpp (StatModel-class), 11
- dpp, StatModel-method (StatModel-class), 11
- empty (ModelEnv-class), 5
- empty, ModelEnv-method (ModelEnv-class), 5
- environment, 5
- eval, 7
- fit (StatModel-class), 11
- fit, StatModel, ModelEnv-method (StatModel-class), 11
- fitted.glinearmodel (StatModel-class), 11
- fitted.linearModel (StatModel-class), 11
- fitted.survReg (StatModel-class), 11
- FormulaParts-class, 2
- Generics, 2
- getModel (Generics), 2
- glinearmodel (StatModel-class), 11
- glm.fit, 11
- has (ModelEnv-class), 5
- has, ModelEnv, character-method (ModelEnv-class), 5
- ICL (Generics), 2
- info, 3
- info, ANY, missing-method (info), 3
- infoCheck (info), 3
- initialize, ModelEnv-method (ModelEnv-class), 5
- KLdiv (Generics), 2
- Lapply (Generics), 2
- linearModel (StatModel-class), 11
- lm, 7
- lm.fit, 11
- logLik.survReg (StatModel-class), 11
- MEapply, 4, 7
- MEapply, ModelEnv-method (MEapply), 4
- model.matrix, 7
- model.matrix.glinearmodel (StatModel-class), 11
- model.matrix.linearModel (StatModel-class), 11
- ModelEnv-class, 5
- ModelEnvFormula, 6
- ModelEnvFormula-class, 8
- ModelEnvMatrix, 9
- na.action, 5
- na.fail (ModelEnv-class), 5

na.fail, ModelEnv-method
 (ModelEnv-class), 5
na.omit (ModelEnv-class), 5
na.omit, ModelEnv-method
 (ModelEnv-class), 5
na.pass (ModelEnv-class), 5
na.pass, ModelEnv-method
 (ModelEnv-class), 5

parameters (Generics), 2
ParseFormula (Generics), 2
posterior (Generics), 2
Predict, 10
predict, 10
predict.glinearModel (StatModel-class),
 11
predict.linearModel (StatModel-class),
 11
print.glinearModel (StatModel-class), 11
print.linearModel (StatModel-class), 11
print.survReg (StatModel-class), 11
prior (Generics), 2

refit (Generics), 2
relabel (Generics), 2

show, ModelEnv-method (ModelEnv-class), 5
StatModel-class, 11
StatModelCapabilities-class, 12
subset (ModelEnv-class), 5
subset, ModelEnv-method
 (ModelEnv-class), 5
survReg (StatModel-class), 11
survreg, 11

weights.linearModel (StatModel-class),
 11
weights.survReg (StatModel-class), 11