Package 'midr'

August 27, 2025

Type Package

Title Learning from Black-Box Models by Maximum Interpretation Decomposition

Version 0.5.1

Description The goal of 'midr' is to provide a model-agnostic method for interpreting and explaining black-box predictive models by creating a globally interpretable surrogate model. The package implements 'Maximum Interpretation Decomposition' (MID), a functional decomposition technique that finds an optimal additive approximation of the original model. This approximation is achieved by minimizing the squared error between the predictions of the black-box model and the surrogate model. The theoretical foundations of MID are described in Iwasawa & Matsumori (2025) [Forthcoming], and the package itself is detailed in Asashiba et al. (2025) <doi:10.48550/arXiv.2506.08338>.

License MIT + file LICENSE

Encoding UTF-8

Imports graphics, grDevices, Rcpp, RcppEigen, rlang, stats, utils

Suggests datasets, ggplot2, khroma, knitr, RColorBrewer, rmarkdown, scales, shapviz, testthat, viridisLite

Config/testthat/edition 3

RoxygenNote 7.3.2

URL https://github.com/ryo-asashi/midr, https://ryo-asashi.github.io/midr/

BugReports https://github.com/ryo-asashi/midr/issues

LinkingTo Rcpp, RcppEigen

NeedsCompilation yes

Author Ryoichi Asasihba [aut, cre], Hirokazu Iwasawa [aut], Reiji Kozuma [ctb]

Maintainer Ryoichi Asasihba <ryoichi.asashiba@gmail.com>

Repository CRAN

Date/Publication 2025-08-27 19:20:02 UTC

Contents

`color.theme` *Color Themes for Graphics*

Description

The `color.theme()` function is the main interface for working with "color.theme" objects. It acts as a dispatcher that, depending on the class of `object`, can retrieve a pre-defined theme by name (see the "Theme Name Syntax" section), create a new theme from a vector of colors or a color-generating function, and modify an existing "color.theme" object.

Usage

```
color.theme(
  object,
  kernel.args = list(),
  options = list(),
  name = NULL,
  source = NULL,
  type = NULL,
  reverse = FALSE,
  env = color.theme.env(),
  ...
)
```

Arguments

object	a character string to retrieve a pre-defined theme, a color kernel (i.e., a vector of colors or a color generating function) to create a new theme, or a "color.theme" object to be modified. See the "Details" section.
kernel.args	a list of arguments to be passed to the color kernel.
options	a list of option values to control the color theme's behavior.
name	a character string for the color theme name.
source	a character string for the source name of the color theme.
type	a character string specifying the type of the color theme. One of "sequential", "diverging", or "qualitative".
reverse	logical. If TRUE, the order of colors is reversed.
env	an environment where the color themes are registered.
...	optional named arguments used to modify the color theme. Any argument passed here will override the corresponding settings in kernel.args or options.
kernel	a color vector, a palette function, or a ramp function that serves as the basis for generating colors.

Details

The "color.theme" object is a special environment that provides two color-generating functions: ...$palette() and ...$ramp().

...$palette() takes an integer n and returns a vector of n discrete colors. It is primarily intended for qualitative themes, where distinct colors are used to represent categorical data.

...$ramp() takes a numeric vector x with values in the [0, 1] interval, and returns a vector of corresponding colors. It maps numeric values onto a continuous color gradient, making it suitable for sequential and diverging themes.

This function, color.theme(), is a versatile dispatcher that behaves differently depending on the class of the object argument. If object is a character string (e.g., "Viridis", "grDevices/RdBu_r@q?alpha=.5"), the string is parsed according to the theme name syntax, and the corresponding pre-defined theme is loaded (see the "Theme Name Syntax" section for details). If object is a color kernel (i.e., a character vector of colors, a palette function, or a ramp function), a new color theme is created from

the kernel. If `object` is a "color.theme" object, the function returns a modified version of the theme, applying any other arguments to update its settings.

Value

`color.theme()` returns a "color.theme" object, which is an environment with the special class attribute, containing the `...$palette()` and `...$ramp` functions, along with other metadata about the theme.

Theme Name Syntax

When retrieving a theme using a character string, you can use a special syntax to specify the source and apply modifications:

`"[(source)/](name)[_r][@(type)][?(query)]"`

- source: (optional) the source package or collection of the theme (e.g., "grDevices").
- name: the name of the theme (e.g., "RdBu").
- "_r": (optional) a suffix to reverse the color order.
- type: (optional) the desired theme type, which will be matched with "sequential", "diverging" or "qualitative" (i.e., "s", "d", and "q" are sufficient, but longer character strings such as "seq", "div", "qual" are also possible).
- query: (optional) a query string to overwrite the color theme's metadata including specific theme options or kernel arguments. Pairs are in `key=value` format and separated by `;` or `&` (e.g., "...?alpha=0.5;na.color='gray50'"). Possible keys include "name", "source", "type", "reverse" and any item of the theme's `options` and `kernel.args`.

See Also

`scale_color_theme`, `set.color.theme`, `color.theme.info`

Examples

```
# Retrieve a pre-defined theme
ct <- color.theme("Mako")
ct$palette(5L)
ct$ramp(seq.int(0, 1, length.out = 5))

# Use special syntax to get a reversed, qualitative theme with alpha value
ct <- color.theme("grDevices/Zissou 1_r@qual?alpha=0.75")
ct$palette(5L)
ct$ramp(seq.int(0, 1, length.out = 5))

# Create a new theme from a vector of colors
ct <- color.theme(c("#003f5c", "#7a5195", "#ef5675", "#ffa600"))
ct$palette(5L)
ct$ramp(seq.int(0, 1, length.out = 5))

# Create a new theme from a palette function
ct <- color.theme(grDevices::rainbow)
```

```
ct$palette(5L)
ct$ramp(seq.int(0, 1, length.out = 5))

# Modify an existing theme
ct <- color.theme(ct, type = "qualitative", kernel.args = list(v = 0.5))
ct$palette(5L)
ct$ramp(seq.int(0, 1, length.out = 5))
```

`color.theme.info` *Retrieve Color Theme Information*

Description

`color.theme.info()` returns a data frame listing all available color themes.

`color.theme.env()` provides direct access to the environment where the color themes are registered.

Usage

```
color.theme.info(env = color.theme.env())

color.theme.env()
```

Arguments

`env`	an environment where the color themes are registered.

Details

These functions provide tools for inspecting the color themes available in the current R session.

`color.theme.info()` is the primary user-facing function for discovering themes by name, source, and type.

`color.theme.env()` is an advanced function that returns the environment currently used as the theme registry. It first checks for a user-specified environment via `getOption("midr.color.theme.env")`. If this option is `NULL` (the default), the function returns the package's internal environment where the default themes are storesd.

Value

`color.theme.info()` returns a data frame with columns "name", "source", and "type".

`color.theme.env()` returns the environment currently used as the default theme registry.

See Also

`color.theme`, `set.color.theme`

Examples

```
# Get a data frame of all available themes
head(color.theme.info())

# Get the environment where color themes are stored
theme_env <- color.theme.env()
names(theme_env)[1:5]
```

`factor.encoder` *Encoder for Qualitative Variables*

Description

`factor.encoder()` creates an encoder function for a qualitative (factor or character) variable. This encoder converts the variable into a one-hot encoded (dummy) design matrix.

`factor.frame()` is a helper function to create a "factor.frame" object that defines the encoding scheme.

Usage

```
factor.encoder(
  x,
  k,
  use.catchall = TRUE,
  catchall = "(others)",
  tag = "x",
  frame = NULL,
  weights = NULL
)

factor.frame(levels, catchall = "(others)", tag = "x")
```

Arguments

`x`	a vector to be encoded as a qualitative variable.
`k`	an integer specifying the maximum number of distinct levels to retain (including the catch-all level). If not positive, all unique values of `x` are used.
`use.catchall`	logical. If `TRUE`, less frequent levels are grouped into the catch-all level.
`catchall`	a character string for the catch-all level.
`tag`	the name of the variable.
`frame`	a "factor.frame" object or a character vector that explicitly defines the levels of the variable.
`weights`	an optional numeric vector of sample weights for `x`.
`levels`	a vector to be used as the levels of the variable.

Details

This function is designed to handle qualitative data for use in the MID model's linear system formulation.

The primary mechanism is one-hot encoding. Each unique level of the input variable becomes a column in the output matrix. For a given observation, the column corresponding to its level is assigned a `1`, and all other columns are assigned `0`.

When a variable has many unique levels (high cardinality), you can use the `use.catchall = TRUE` and `k` arguments. This will group the `k - 1` most frequent levels into their own columns, while all other less frequent levels are consolidated into a single `catchall` level (e.g., "(others)" by default). This is crucial for preventing MID models from becoming overly complex.

Value

`factor.encoder()` returns an object of class "encoder". This is a list containing the following components:

`frame`	a "factor.frame" object containing the encoding information (levels).
`encode`	a function to convert a vector `x` into a one-hot encoded matrix.
`n`	the number of encoding levels (i.e., columns in the design matrix).
`type`	a character string describing the encoding type: "factor" or "null".

`factor.frame()` returns a "factor.frame" object containing the encoding information.

See Also

`numeric.encoder`

Examples

```
# Create an encoder for a qualitative variable
data(iris, package = "datasets")
enc <- factor.encoder(x = iris$Species, use.catchall = FALSE, tag = "Species")
enc

# Encode a vector with NA
enc$encode(x = c("setosa", "virginica", "ensata", NA, "versicolor"))

# Create an encoder with a pre-defined encoding frame
frm <- factor.frame(c("setosa", "virginica"), "other iris")
enc <- factor.encoder(x = iris$Species, frame = frm)
enc
enc$encode(c("setosa", "virginica", "ensata", NA, "versicolor"))

# Create an encoder with a character vector specifying the levels
enc <- factor.encoder(x = iris$Species, frame = c("setosa", "versicolor"))
enc$encode(c("setosa", "virginica", "ensata", NA, "versicolor"))
```

get.yhat *Wrapper Prediction Function*

Description

get.yhat() is a generic function that provides a unified interface for obtaining predictions from various fitted model objects.

Usage

```
get.yhat(object, newdata, ...)

## Default S3 method:
get.yhat(object, newdata, target = -1L, ...)

## S3 method for class 'mid'
get.yhat(object, newdata, ...)

## S3 method for class 'lm'
get.yhat(object, newdata, ...)

## S3 method for class 'glm'
get.yhat(object, newdata, ...)

## S3 method for class 'rpart'
get.yhat(object, newdata, target = -1L, ...)

## S3 method for class 'randomForest'
get.yhat(object, newdata, target = -1L, ...)

## S3 method for class 'ranger'
get.yhat(object, newdata, target = -1L, ...)

## S3 method for class 'svm'
get.yhat(object, newdata, target = -1L, ...)

## S3 method for class 'ksvm'
get.yhat(object, newdata, target = -1L, ...)

## S3 method for class 'AccurateGLM'
get.yhat(object, newdata, ...)

## S3 method for class 'glmnet'
get.yhat(object, newdata, ...)

## S3 method for class 'model_fit'
get.yhat(object, newdata, target = -1L, ...)
```

```
## S3 method for class 'rpf'
get.yhat(object, newdata, target = -1L, ...)
```

Arguments

object	a fitted model object.
newdata	a data.frame or matrix.
...	optional arguments passed on to the underlying predict() method for the object's class.
target	an integer or character vector specifying the target levels used for the classification models that return a matrix or data frame of class probabilities. The default, -1, represents the probability of not being the base level.

Details

While many predictive models have a stats::predict() method, the structure and type of their outputs are not uniform. For example, some return a numeric vector, others a matrix of class probabilities, and some a list. This function, get.yhat(), abstracts away this complexity.

For regression models, it returns the numeric prediction in the original scale of the response variable. For classification models, it returns the sum of class probabilities for the classes specified by the target argument.

Furthermore, get.yhat() provides more consistent handling of missing values. While some stats::predict() methods may return a shorter vector by omitting NAs, get.yhat() is designed to return a vector of the same length as newdata, preserving NAs in their original positions.

The design of get.yhat() is strongly influenced by DALEX::yhat().

Value

get.yhat() returns a numeric vector of model predictions for newdata.

See Also

predict.mid

Examples

```
data(trees, package = "datasets")
model <- glm(Volume ~ ., trees, family = Gamma(log))

# The output of stats::predict() might not be in the scale of the response variable
predict(model, trees[1:5, ])

# get.yhat() returns a numeric vector in the original scale of the response variable
get.yhat(model, trees[1:5, ])
predict(model, trees[1:5, ], type = "response")
```

ggmid *Plot MID Component Functions with ggplot2*

Description

ggmid() is an S3 generic function for creating various visualizations from MID-related objects using **ggplot2**. For "mid" objects (i.e., fitted MID models), it visualizes a single component function specified by the term argument.

Usage

```
ggmid(object, ...)

## S3 method for class 'mid'
ggmid(
  object,
  term,
  type = c("effect", "data", "compound"),
  theme = NULL,
  intercept = FALSE,
  main.effects = FALSE,
  data = NULL,
  limits = c(NA, NA),
  jitter = 0.3,
  resolution = c(100L, 100L),
  ...
)

## S3 method for class 'mid'
autoplot(object, ...)
```

Arguments

object	a "mid" object to be visualized.
...	optional parameters passed to the main plotting layer.
term	a character string specifying the component function to be plotted.
type	the plotting style. One of "effect", "data" or "compound".
theme	a character string or object defining the color theme. See color.theme for details.
intercept	logical. If TRUE, the intercept is added to the MID values.
main.effects	logical. If TRUE, main effects are included in the interaction plot.
data	a data frame to be plotted with the corresponding MID values. If not provided, data is automatically extracted based on the function call.
limits	a numeric vector of length two specifying the limits of the plotting scale. NA values are replaced by the minimum and/or maximum MID values.

jitter	a numeric value specifying the amount of jitter for the data points.
resolution	an integer or vector of two integers specifying the resolution of the raster plot for interactions.

Details

For "mid" objects, ggmid() creates a "ggplot" object that visualizes a component function of the fitted MID model.

The type argument controls the visualization style. The default, type = "effect", plots the component function itself. In this style, the plotting method is automatically selected based on the effect's type: a line plot for quantitative main effects; a bar plot for qualitative main effects; and a raster plot for interactions. The type = "data" option creates a scatter plot of data, colored by the values of the component function. The type = "compound" option combines both approaches, plotting the component function alongside the data points.

Value

ggmid.mid() returns a "ggplot" object.

See Also

interpret, ggmid.mid.importance, ggmid.mid.conditional, ggmid.mid.breakdown, plot.mid

Examples

```
data(diamonds, package = "ggplot2")
set.seed(42)
idx <- sample(nrow(diamonds), 1e4)
mid <- interpret(price ~ (carat + cut + color + clarity)^2, diamonds[idx, ])

# Plot a quantitative main effect
ggmid(mid, "carat")

# Plot a qualitative main effect
ggmid(mid, "clarity")

# Plot an interaction effect with data points and a raster layer
ggmid(mid, "carat:clarity", type = "compound", data = diamonds[idx, ])

# Use a different color theme
ggmid(mid, "clarity:color", theme = "RdBu")
```

---

ggmid.mid.breakdown *Plot MID Breakdowns with ggplot2*

Description

For "mid.breakdown" objects, ggmid() visualizes the breakdown of a prediction by component functions.

Usage

```
## S3 method for class 'mid.breakdown'
ggmid(
  object,
  type = c("waterfall", "barplot", "dotchart"),
  theme = NULL,
  terms = NULL,
  max.terms = 15L,
  width = NULL,
  vline = TRUE,
  catchall = "others",
  format = c("%t=%v", "%t"),
  ...
)

## S3 method for class 'mid.breakdown'
autoplot(object, ...)
```

Arguments

`object`	a "mid.breakdown" object to be visualized.
`type`	the plotting style. One of "waterfall", "barplot" or "dotchart".
`theme`	a character string or object defining the color theme. See `color.theme` for details.
`terms`	an optional character vector specifying which terms to display.
`max.terms`	the maximum number of terms to display in the plot. Less important terms will be grouped into a "catchall" category.
`width`	a numeric value specifying the width of the bars.
`vline`	logical. If `TRUE`, a vertical line is drawn at the zero or intercept line.
`catchall`	a character string for the catchall label.
`format`	a character string or character vector of length two to be used as the format of the axis labels. Use "%t" for the term name (e.g., "Wind") and "%v" for the values (e.g., "30").
`...`	optional parameters passed on to the main layer.

Details

This is an S3 method for the `ggmid()` generic that creates a breakdown plot from a "mid.breakdown" object, visualizing the contribution of each component function to a single prediction.

The `type` argument controls the visualization style. The default, `type = "waterfall"` (default), creates a waterfall plot that shows how the prediction builds from the intercept, with each term's contribution sequentially added or subtracted. The `type = "barplot"` option creates a standard bar plot where the length of each bar represents the magnitude of the term's contribution. The `type = "dotchart"` option creates a dot plot showing the contribution of each term as a point connected to a zero baseline.

Value

ggmid.mid.breakdown() returns a "ggplot" object.

See Also

mid.breakdown, ggmid, plot.mid.breakdown

Examples

```
data(diamonds, package = "ggplot2")
set.seed(42)
idx <- sample(nrow(diamonds), 1e4)
mid <- interpret(price ~ (carat + cut + color + clarity)^2, diamonds[idx, ])
mbd <- mid.breakdown(mid, diamonds[1L, ])

# Create a waterfall plot
ggmid(mbd, type = "waterfall")

# Create a bar plot with a different theme
ggmid(mbd, type = "barplot", theme = "highlight")

# Create a dot chart
ggmid(mbd, type = "dotchart", size = 3)
```

---

ggmid.mid.conditional *Plot MID Conditional Expectations with ggplot2*

Description

For "mid.conditional" objects, ggmid() visualizes Individual Conditional Expectation (ICE) curves derived from a fitted MID model.

Usage

```
## S3 method for class 'mid.conditional'
ggmid(
  object,
  type = c("iceplot", "centered"),
  theme = NULL,
  term = NULL,
  var.alpha = NULL,
  var.color = NULL,
  var.linetype = NULL,
  var.linewidth = NULL,
  reference = 1L,
  dots = TRUE,
  sample = NULL,
  ...
```

```
)

## S3 method for class 'mid.conditional'
autoplot(object, ...)
```

Arguments

object	a "mid.conditional" object to be visualized.
type	the plotting style. One of "iceplot" or "centered".
theme	a character string or object defining the color theme. See color.theme for details.
term	an optional character string specifying an interaction term. If passed, the ICE curve for the specified term is plotted.
var.alpha	a variable name or expression to map to the alpha aesthetic.
var.color	a variable name or expression to map to the color aesthetic.
var.linetype	a variable name or expression to map to the linetype aesthetic.
var.linewidth	a variable name or expression to map to the linewidth aesthetic.
reference	an integer specifying the index of the sample points to use as the reference for centering the c-ICE plot.
dots	logical. If TRUE, points representing the actual predictions for each observation are plotted.
sample	an optional vector specifying the names of observations to be plotted.
...	optional parameters passed on to the main layer.

Details

This is an S3 method for the ggmid() generic that produces ICE curves from a "mid.conditional" object. ICE plots are a model-agnostic tool for visualizing how a model's prediction for a single observation changes as one feature varies. This function plots one line for each observation in the data.

The type argument controls the visualization style: The default, type = "iceplot", plots the row ICE curves. The type = "centered" option creates the centered ICE (c-ICE) plot, where each curve is shifted so start at zero, which makes it easier to compare the slopes of the curves.

The var.color, var.alpha, etc., arguments allow you to map aesthetics to other variables in your data using (possibly) unquoted expressions.

Value

ggmid.mid.conditional() returns a "ggplot" object.

See Also

mid.conditional, ggmid, plot.mid.conditional

Examples

```
data(airquality, package = "datasets")
library(midr)
mid <- interpret(Ozone ~ .^2, airquality, lambda = 0.1)
ice <- mid.conditional(mid, "Temp", data = airquality)

# Create an ICE plot, coloring lines by 'Wind'
ggmid(ice, var.color = "Wind")

# Create a centered ICE plot, mapping color and linetype to other variables
ggmid(ice, type = "centered", theme = "Purple-Yellow",
      var.color = factor(Month), var.linetype = Wind > 10)
```

---

ggmid.mid.importance *Plot MID Importance with ggplot2*

Description

For "mid.importance" objects, `ggmid()` visualizes the importance of component functions of the fitted MID model.

Usage

```
## S3 method for class 'mid.importance'
ggmid(
  object,
  type = c("barplot", "dotchart", "heatmap", "boxplot"),
  theme = NULL,
  max.terms = 30L,
  ...
)

## S3 method for class 'mid.importance'
autoplot(object, ...)
```

Arguments

`object`	a "mid.importance" object to be visualized.
`type`	the plotting style. One of "barplot", "dotchart", "heatmap", or "boxplot".
`theme`	a character string or object defining the color theme. See `color.theme` for details.
`max.terms`	the maximum number of terms to display in the bar, dot and box plots.
`...`	optional parameters passed on to the main layer.

Details

This is an S3 method for the ggmid() generic that creates an importance plot from a "mid.importance" object, visualizing the average contribution of component functions to the fitted MID model.

The type argument controls the visualization style. The default, type = "barplot", creates a standard bar plot where the length of each bar represents the overall importance of the term. The type = "dotchart" option creates a dot plot, offering a clean alternative to the bar plot for visualizing term importance. The type = "heatmap" option creates a matrix-shaped heat map where the color of each cell represents the importance of the interaction between a pair of variables, or the main effect on the diagonal. The type = "boxplot" option creates a box plot where each box shows the distribution of a term's contributions across all observations, providing insight into the variability of each term's effect.

Value

ggmid.mid.importance() returns a "ggplot" object.

See Also

mid.importance, ggmid, plot.mid.importance

Examples

```
data(diamonds, package = "ggplot2")
set.seed(42)
idx <- sample(nrow(diamonds), 1e4)
mid <- interpret(price ~ (carat + cut + color + clarity)^2, diamonds[idx, ])
imp <- mid.importance(mid)

# Create a bar plot (default)
ggmid(imp)

# Create a dot chart
ggmid(imp, type = "dotchart", theme = "Okabe-Ito", size = 3)

# Create a heatmap
ggmid(imp, type = "heatmap")

# Create a boxplot to see the distribution of effects
ggmid(imp, type = "boxplot")
```

---

interpret *Fit MID Models*

Description

interpret() is used to fit a Maximum Interpretation Decomposition (MID) model. MID models are additive, highly interpretable models composed of functions, each with up to two variables.

Usage

```
interpret(object, ...)

## Default S3 method:
interpret(
  object,
  x,
  y = NULL,
  weights = NULL,
  pred.fun = get.yhat,
  link = NULL,
  k = c(NA, NA),
  type = c(1L, 1L),
  frames = list(),
  interactions = FALSE,
  terms = NULL,
  singular.ok = FALSE,
  mode = 1L,
  method = NULL,
  lambda = 0,
  kappa = 1e+06,
  na.action = getOption("na.action"),
  verbosity = 1L,
  encoding.digits = 3L,
  use.catchall = FALSE,
  catchall = "(others)",
  max.ncol = 10000L,
  nil = 1e-07,
  tol = 1e-07,
  pred.args = list(),
  ...
)

## S3 method for class 'formula'
interpret(
  formula,
  data = NULL,
  model = NULL,
  pred.fun = get.yhat,
  weights = NULL,
  subset = NULL,
  na.action = getOption("na.action"),
  verbosity = 1L,
  mode = 1L,
  drop.unused.levels = FALSE,
  pred.args = list(),
  ...
)
```

Arguments

object	a fitted model object to be interpreted.
...	optional arguments. For interpret.formula(), arguments to be passed on to interpret.default(). For interpret.default(), ... can include convenient aliases (e.g., "ok" for singular.ok, "ie" for interaction) as well as several advanced fitting options (see the "Advanced Fitting Options" section for details).
x	a matrix or data.frame of predictor variables to be used in the fitting process. The response variable should not be included.
y	an optional numeric vector of the model predictions or the response variable.
weights	a numeric vector of sample weights for each observation in x.
pred.fun	a function to obtain predictions from a fitted model, where the first argument is for the fitted model and the second argument is for new data. The default is get.yhat().
link	a character string specifying the link function: one of "logit", "probit", "cauchit", "cloglog", "identity", "log", "sqrt", "1/mu^2", "inverse", "translogit", "transprobit", "identity-logistic" and "identity-gaussian", or an object containing two functions linkfun() and linkinv(). See help(make.link).
k	an integer or a vector of two integers specifying the maximum number of sample points for main effects (k[1]) and interactions (k[2]). If a single integer is provided, it is used for main effects while the value for interactions is automatically determined. Any NA value will also trigger this automatic determination. With non-positive values, all unique data points are used as sample points.
type	an integer or integer-valued vector of length two. The type of encoding. The effects of quantitative variables are modeled as piecewise linear functions if type is 1, and as step functions if type is 0. If a vector is passed, type[1L] is used for main effects and type[2L] is used for interactions.
frames	a named list of encoding frames ("numeric.frame" or "factor.frame" objects). The encoding frames are used to encode the variable of the corresponding name. If the name begins with "|" or ":", the encoding frame is used only for main effects or interactions, respectively.
interactions	logical. If TRUE and if terms and formula are not supplied, all interactions for each pair of variables are modeled and calculated.
terms	a character vector of term labels specifying the set of component functions to be modeled. If not passed, terms includes all main effects, and all interactions if interaction is TRUE.
singular.ok	logical. If FALSE, a singular fit is an error.
mode	an integer specifying the method of calculation. If mode is 1, the centralization constraints are treated as penalties for the least squares problem. If mode is 2, the constraints are used to reduce the number of free parameters.
method	an integer specifying the method to be used to solve the least squares problem. A non-negative value will be passed to RcppEigen::fastLmPure(). If negative, stats::lm.fit() is used.

`lambda`	the penalty factor for pseudo smoothing. The default is `0`.
`kappa`	the penalty factor for centering constraints. Used only when `mode` is `1`. The default is `1e+6`.
`na.action`	a function or character string specifying the method of `NA` handling. The default is "na.omit".
`verbosity`	the level of verbosity. `0`: fatal, `1`: warning (default), `2`: info or `3`: debug.
`encoding.digits`	an integer. The rounding digits for encoding numeric variables. Used only when `type` is `1`.
`use.catchall`	logical. If `TRUE`, less frequent levels of qualitative variables are dropped and replaced by the catchall level.
`catchall`	a character string specifying the catchall level.
`max.ncol`	integer. The maximum number of columns of the design matrix.
`nil`	a threshold for the intercept and coefficients to be treated as zero. The default is `1e-7`.
`tol`	a tolerance for the singular value decomposition. The default is `1e-7`.
`pred.args`	optional parameters other than the fitted model and new data to be passed to `pred.fun()`.
`formula`	a symbolic description of the MID model to be fit.
`data`	a data.frame, list or environment containing the variables in `formula`. If not found in data, the variables are taken from `environment(formula)`.
`model`	a fitted model object to be interpreted.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`drop.unused.levels`	logical. If `TRUE`, unused levels of factors will be dropped.

Details

The MID model approximates a target model's prediction function $f(\mathbf{x})$, or values of the response variable $\mathbf{y}$. This model, denoted as $\mathcal{F}(\mathbf{x})$, has the following structure:

$$\mathcal{F}(\mathbf{x}) = f_\phi + \sum_j f_j(x_j) + \sum_{j<k} f_{jk}(x_j, x_k)$$

where f_ϕ is the intercept, $f_j(x_j)$ is the main effect of feature j, and $f_{jk}(x_j, x_k)$ is the second-order interaction effect between features j and k.

To ensure that the decomposed components are unique, they are fitted under the *centering constraints*: each main effect's average is constrained to be zero, and each interaction effect's conditional averages are also constrained to be zero. The model is fitted by minimizing the squared error between the target, $f(\mathbf{x})$ or $\mathbf{y}$, and the surrogate $\mathcal{F}(\mathbf{x})$, which is typically evaluated on a representative dataset.

Value

`interpret()` returns an object of class "mid". This is a list with the following components:

`weights`	a numeric vector of the sample weights.
`call`	the matched call.
`terms`	the term labels.
`link`	a "link-glm" or "link-midr" object containing the link function.
`intercept`	the intercept.
`encoders`	a list of variable encoders.
`main.effects`	a list of data frames representing the main effects.
`interacions`	a list of data frames representing the interactions.
`ratio`	the ratio of the sum of squared error between the target model predictions and the fitted MID values, to the sum of squared deviations of the target model predictions.
`fitted.matrix`	a matrix showing the breakdown of the predictions into the effects of the component functions.
`linear.predictors`	a numeric vector of the linear predictors.
`fitted.values`	a numeric vector of the fitted values.
`residuals`	a numeric vector of the working residuals.
`na.action`	information about the special handlings of `NA`s.

Advanced Fitting Options

The `...` argument can be used to pass several advanced fitting options:

fit.intercept logical. If `TRUE`, the intercept term is fitted as part of the least squares problem. If `FALSE` (default), it is calculated as the weighted mean of the response.

interpolate.beta a character string specifying the method for interpolating unestimable coefficients (betas) that arise from sparse deta regions. Can be "iterative" for an iterative smoothing process, "direct" for solving a linear system, or "none" to disable interpolation.

maxit an integer specifying the maximum number of iterations for the "iterative" interpolation method.

weighted.norm logical. If `TRUE`, the columns of the design matrix are normalized by the square root of their weighted sum. This is required to ensure the minimum-norm least squares solution obtained by appropriate methods (i.e., `4` or `5`) of `fastLmPure()` is the minimum-norm solution in a *weighted* sense.

weighted.endoding logical. If `TRUE`, sample weights are used during the encoding process (e.g., for calculating quantiles to determine knots).

References

Asashiba R, Kozuma R, Iwasawa H (2025). "midr: Learning from Black-Box Models by Maximum Interpretation Decomposition." 2506.08338, https://arxiv.org/abs/2506.08338.

See Also

print.mid, summary.mid, predict.mid, plot.mid, ggmid, mid.plots, mid.effect, mid.terms, mid.importance, mid.conditional, mid.breakdown

Examples

```
# Fit a MID model as a surrogate for another model
data(cars, package = "datasets")
model <- lm(dist ~ I(speed^2) + speed, cars)
mid <- interpret(dist ~ speed, cars, model)
plot(mid, "speed", intercept = TRUE)
points(cars)

# Fit a MID model as a standalone predictive model
data(airquality, package = "datasets")
mid <- interpret(Ozone ~ .^2, data = airquality, lambda = .5)
plot(mid, "Wind")
plot(mid, "Temp")
plot(mid, "Wind:Temp", main.effects = TRUE)

data(Nile, package = "datasets")
nile <- data.frame(time = 1:length(Nile), flow = as.numeric(Nile))

# A flexible fit with many knots
mid <- interpret(flow ~ time, data = nile, k = 100L)
plot(mid, "time", intercept = TRUE, limits = c(600L, 1300L))
points(x = 1L:100L, y = Nile)

# A smoother fit with fewer knots
mid <- interpret(flow ~ time, data = nile, k = 10L)
plot(mid, "time", intercept = TRUE, limits = c(600L, 1300L))
points(x = 1L:100L, y = Nile)

# A pseudo-smoothed fit using a penalty
mid <- interpret(flow ~ time, data = nile, k = 100L, lambda = 100L)
plot(mid, "time", intercept = TRUE, limits = c(600L, 1300L))
points(x = 1L:100L, y = Nile)
```

---

mid.breakdown *Calculate MID Breakdowns*

Description

mid.breakdown() calculates the contribution of each component function of a fitted MID model to a single prediction. It breaks down the total prediction into the effects of the intercept, main effects, and interactions.

Usage

```
mid.breakdown(
  object,
  data = NULL,
  row = NULL,
  sort = TRUE,
  format = c("%s", "%s, %s")
)
```

Arguments

object	a "mid" object.
data	a data frame containing one or more observations for which to calculate the MID breakdown. If not provided, data is automatically extracted based on the function call.
row	an optional numeric value or character string specifying the row of data to be used for the breakdown. If NULL, and the data contains two or more observations, only the first observation is used.
sort	logical. If TRUE, the output data frame is sorted by the absolute contribution of each effect.
format	a character vector of length two to be used as a format string for sprintf() to display the values of main effects and interactions, respectively.

Details

mid.breakdown() is a method for local interpretability. For a given observation, it provides a clear answer to the question, "How much did each component of the MID model contribute to the final prediction?"

The function calculates the value of each term in the MID model's additive structure for the specified observation. The total prediction is the sum of these individual contributions. The prediction, denoted $\mathcal{F}(\mathbf{x})$, is decomposed as:

$$\mathcal{F}(\mathbf{x}) = f_\phi + \sum_j f_j(x_j) + \sum_{j<k} f_{jk}(x_j, x_k)$$

The output data frame itemizes the numerical value of each main effect ($f_j(x_j)$) and interaction effect ($f_{jk}(x_j, x_k)$), along with the intercept (f_ϕ). This makes the prediction transparent and easy to understand.

Value

mid.breakdown() returns an object of class "mid.breakdown". This is a list with the following components:

breakdown	a data frame containing the breakdown of the prediction.
data	the data frame containing the predictor variable values used for the prediction.
intercept	the intercept of the MID model.
prediction	the predicted value from the MID model.

See Also

interpret, plot.mid.breakdown, ggmid.mid.breakdown

Examples

```
data(airquality, package = "datasets")
mid <- interpret(Ozone ~ .^2, data = airquality, lambda = 1)

# Calculate the breakdown for the first observation in the data
mbd <- mid.breakdown(mid, data = airquality, row = 1)
print(mbd)

# Calculate the breakdown for the third observation in the data
mbd <- mid.breakdown(mid, data = airquality, row = 3)
print(mbd)
```

---

mid.conditional *Calculate MID Conditional Expectations*

Description

`mid.conditional()` calculates the data required to draw Individual Conditional Expectation (ICE) curves from a fitted MID model. ICE curves visualize how a single observation's prediction changes as a specified variable's value varies, while all other variable are held constant.

Usage

```
mid.conditional(
  object,
  variable,
  data = NULL,
  n.samples = 100L,
  max.rows = 100000L,
  type = c("response", "link"),
  keep.effects = TRUE
)
```

Arguments

`object`	a "mid" object.
`variable`	a character string or expression specifying the singlue predictor variable for which to calculate ICE curves.
`data`	a data frame containing the observations to be used for the ICE calculations. If not provided, data is automatically extracted based on the function call.
`n.samples`	the number of sample points for the `varibale`'s range.
`max.rows`	the maximum number of rows for the output data frames. If the number of evaluation points exceeds this limit, the original data is randomly subsampled.

type	the type of prediction to return. "response" (default) for the original scale or "link" for the scale of the linear predictor.
keep.effects	logical. If TRUE, the effects of individual component functions are stored in the output object.

Details

The function generates a set of hypothetical observations by creating copies of the original data and varying the specified variable across a range of sample points. It then obtains a prediction for each of these hypothetical observations from the MID model. The returned object can be plotted to visualize the ICE curves.

Value

mid.conditional() returns an object of class "mid.conditional". This is a list with the following components:

terms	a character vector of relevant terms for the variable.
observed	a data frame of the original observations used, along with their predictions.
conditional	a data frame of the hypothetical observations and their corresponding predictions.
values	a vector of the sample points for the variable used in the ICE calculation

See Also

interpret, plot.mid.conditional, ggmid.mid.conditional

Examples

```
data(airquality, package = "datasets")
mid <- interpret(Ozone ~ .^2, data = airquality, lambda = 1)

# Calculate the ICE values for a fitted MID model
ice <- mid.conditional(mid, variable = "Wind", data = airquality)
print(ice)
```

---

mid.effect *Evaluate Single MID Component Functions*

Description

mid.effect() calculates the contribution of a single component function of a fitted MID model. It serves as a low-level helper function for making predictions or for direct analysis of a term's effect.

mid.f() is a convenient shorthand for mid.effect().

Usage

```
mid.effect(object, term, x, y = NULL)

mid.f(object, term, x, y = NULL)
```

Arguments

object	a "mid" object.
term	a character string specifying the component function (term) to evaluate.
x	a vector of values for the first variable in the term. If a matrix or data frame is provided, values of the related variables are extracted from it.
y	a vector of values for the second variable in an interaction term.

Details

`mid.effect()` is a low-level function designed to calculate the contribution of a single component function. Unlike `predict.mid()`, which is designed to return total model predictions, `mid.effect()` is more flexible. It accepts vectors, as well as data frames, as input for `x` and `y`, making it particularly useful for visualizing a component's effect in combination with other functions, such as `graphics::curve()`.

For a main effect, the function evaluates the component function $f_j(x_j)$ for a vector of values x_j. For an interaction, it evaluates $f_{jk}(x_j, x_k)$ using vectors x_j and x_k.

Value

`mid.effect()` returns a numeric vector of the calculated term contributions, Wwith the same length as `x` and `y`.

See Also

`interpret`, `predict.mid`

Examples

```
data(airquality, package = "datasets")
mid <- interpret(Ozone ~ .^2, data = airquality, lambda = 1)

# Visualize the main effect of "Wind"
curve(mid.effect(mid, term = "Wind", x), from = 0, to = 25)

# Visualize the interaction of "Wind" and "Temp"
curve(mid.f(mid, "Wind:Temp", x, 50), 0, 25)
curve(mid.f(mid, "Wind:Temp", x, 60), 0, 25, add = TRUE, lty = 2)
curve(mid.f(mid, "Wind:Temp", x, 70), 0, 25, add = TRUE, lty = 3)
```

`mid.importance` *Calculate MID Importance*

Description

`mid.importance()` calculates the MID importance of a fitted MID model. This is a measure of feature importance that quantifies the average contribution of each component function across a dataset.

Usage

```
mid.importance(object, data = NULL, weights = NULL, sort = TRUE, measure = 1L)
```

Arguments

`object`	a "mid" object.
`data`	a data frame containing the observations to calculate the importance. If `NULL`, the `fitted.matrix` from the "mid" object is used.
`weights`	an optional numeric vector of sample weights.
`sort`	logical. If `TRUE`, the output data frame is sorted by importance in descending order.
`measure`	an integer specifying the measure of importance. Possible alternatives are `1` for the mean absolute effect, `2` for the root mean square effect, and `3` for the median absolute effect.

Details

The MID importance of a component function (e.g., a main effect or an interaction) is defined as the mean absolute effect on the predictions within the given data. Terms with higher importance have a larger average impact on the model's overall predictions.

Value

`mid.importance()` returns an object of class "mid.importance". This is a list containing the following components:

`importance`	a data frame with the calculated importance values, sorted by default.
`predictions`	the matrix of the fitted or predicted MID values.
`measure`	a character string describing the type of the importance measure used.

See Also

`interpret`, `plot.mid.importance`, `ggmid.mid.importance`

Examples

```
data(airquality, package = "datasets")
mid <- interpret(Ozone ~ .^2, data = airquality, lambda = 1)

# Calculate MID importance using median absolute contribution
imp <- mid.importance(mid)
print(imp)

# Calculate MID importance using root mean square contribution
imp <- mid.importance(mid, measure = 2)
print(imp)
```

---

mid.plots *Plot Multiple MID Component Functions*

Description

`mid.plots()` is a convenience function for applying `ggmid()` or `plot()` to multiple component functions of a "mid" object at once. It can automatically determine common plotting scales and manage the layout.

Usage

```
mid.plots(
  object,
  terms = mid.terms(object, interactions = FALSE),
  limits = c(NA, NA),
  intercept = FALSE,
  main.effects = FALSE,
  max.plots = NULL,
  engine = c("ggplot2", "graphics"),
  ...
)
```

Arguments

`object`	a "mid" object.
`terms`	a character vector of the terms to be visualized. By default, only the main effect terms are used.
`limits`	a numeric vector of length two specifying the y-axis limits. `NA` values are replaced by the minimum and/or maximum of the plotted MID values. If `intercept = TRUE` is set, the intercept is also included in the limit calculation.
`intercept`	logical. If `TRUE`, the intercept is added to the MID values and the plotting scale is shifted accordingly.
`main.effects`	logical. If `TRUE`, main effects are added to the interaction plots to show conditional effects. This argument disables automatic limit calculations.

max.plots	an integer specifying the maximum number of plots to be generated.
engine	the plotting engine to use, either "ggplot2" or "graphics".
...	optional parameters passed on to plot.mid() or ggmid().

Value

If engine is "ggplot2", mid.plots() returns a list of "ggplot" objects. Otherwise (i.e., if engine is "graphics"), mid.plots() produces plots as side-effects and returns NULL invisibly.

See Also

interpret, plot.mid, ggmid

Examples

```
data(diamonds, package = "ggplot2")
set.seed(42)
idx <- sample(nrow(diamonds), 1e4L)
mid <- interpret(price ~ (carat + cut + color + clarity) ^ 2, diamonds[idx, ])

# Plot selected main effects and interaction using the ggplot2 engine
mid.plots(mid, mid.terms(mid, require = "color", remove = "cut"), limits = NULL)
```

---

mid.terms *Extract Terms from MID Models*

Description

mid.terms() extracts term labels from a fitted MID model or derivative objects. Its primary strength is the ability to filter terms based on their type (main effects vs. interactions) or their associated variable names.

Usage

```
mid.terms(
  object,
  main.effects = TRUE,
  interactions = TRUE,
  require = NULL,
  remove = NULL,
  ...
)
```

Arguments

`object`	a "mid" object or another object that contains model terms. Can be a "mid.importance", "mid.conditional", or "mid.breakdown" object.
`main.effects`	logical. If `FALSE`, the main effect terms are excluded.
`interactions`	logical. If `FALSE`, the interactions terms are excluded.
`require`	a character vector of variable names. Only terms related to at least one of these variables are returned.
`remove`	a character vector of variable names. Terms related to any of these variables are excluded.
`...`	aliases are supported for convenience: "me" for `main.effects` and "ie" for `interactions`.

Details

A "term" in a MID model refers to either a main effect (e.g., "Wind") or an interaction effect (e.g., "Wind:Temp"). This function provides a flexible way to select a subset of these terms, which is useful for plotting, summarizing, or other downstream analyses.

Value

`mid.terms()` returns a character vector of the selected term labels.

Note

This function provides the common underlying logic for the `stats::terms()` S3 methods for "mid", "mid.importance", "mid.conditional", and "mid.breakdown" objects.

See Also

`interpret`

Examples

```
data(airquality, package = "datasets")
mid <- interpret(Ozone ~ .^2, airquality, lambda = 1)

# Get only main effect terms
mid.terms(mid, interactions = FALSE)

# Get terms related to "Wind" or "Temp"
mid.terms(mid, require = c("Wind", "Temp"))

# Get terms related to "Wind" or "Temp", but exclude any with "Day"
mid.terms(mid, require = c("Wind", "Temp"), remove = "Day")

# Get the predicted contributions of only the terms associated with "Wind"
terms_wind <- mid.terms(mid, require = "Wind")
predict(mid, airquality[1:3,], terms = terms_wind, type = "terms")
```

`numeric.encoder` *Encoder for Quantitative Variables*

Description

`numeric.encoder()` creates an encoder function for a quantitative variable. This encoder can then be used to convert a numeric vector into a design matrix using either piecewise linear or one-hot interval encoding, which are core components for modeling effects in a MID model.

`numeric.frame()` is a helper function to create a "numeric.frame" object that defines the encoding scheme.

Usage

```
numeric.encoder(
  x,
  k,
  type = 1L,
  encoding.digits = NULL,
  tag = "x",
  frame = NULL,
  weights = NULL
)

numeric.frame(
  reps = NULL,
  breaks = NULL,
  type = NULL,
  encoding.digits = NULL,
  tag = "x"
)
```

Arguments

`x`	a numeric vector to be encoded.
`k`	an integer specifying the coarseness of the encoding. If not positive, all unique values of `x` are used as knots or bins.
`type`	an integer (`1` or `0`) specifying the encoding method (see the "details" section).
`encoding.digits`	an integer specifying the rounding digits for the piecewise linear encoding (`type = 1`).
`tag`	the name of the variable.
`frame`	a "numeric.frame" object or a numeric vector that explicitly defines the knots or breakes for the encoding.
`weights`	an optional numeric vector of sample weights for `x`.
`reps`	a numeric vector to be used as the representative values (knots).
`breaks`	a numeric vector to be used as the binning breaks.

Details

The primary purpose of the encoder is to transform a single numeric variable into a design matrix for the MID model's linear system formulation. The output of the encoder depends on the `type` argument.

When `type = 1`, the variable's effect is modeled as a piecewise linear function with `k` knots including both ends. For each value, the encoder finds the two nearest knots and assigns a weight to each, based on its relative position. This results in a design matrix where each row has at most two non-zero values that sum to `1`. This approach creates a smooth, continuous representation of the effect.

When `type = 0`, the variable's effect is modeled as a step function by dividing its range into `k` intervals (bins). The encoder determines which interval each value falls into and assigns a `1` to the corresponding column in the design matrix, with all other columns being `0`. This results in a standard one-hot encoded matrix and creates a discrete, bin-based representation of the effect.

Value

`numeric.encoder()` returns an object of class "encoder". This is a list containing the following components:

`frame`	a "numeric.frame" object containing the encoding information.
`encode`	a function to convert a numeric vector `x` into a dummy matrix.
`n`	the number of encoding levels (i.e., columns in the design matrix).
`type`	a character string describing the encoding type: "linear", "constant", or "null".

`numeric.frame()` returns a "numeric.frame" object containing the encoding information.

See Also

`factor.encoder`

Examples

```
# Create an encoder for a quantitative variable
data(iris, package = "datasets")
enc <- numeric.encoder(x = iris$Sepal.Length, k = 5L, tag = "Sepal.Length")
enc

# Encode a numeric vector with NA and Inf
enc$encode(x = c(4:8, NA, Inf))

# Create an encoder with a pre-defined encoding frame
frm <- numeric.frame(breaks = c(3, 5, 7, 9), type = 0L)
enc <- numeric.encoder(x = iris$Sepal.Length, frame = frm)
enc$encode(x = c(4:8, NA, Inf))

# Create an encoder with a numeric vector specifying the knots
enc <- numeric.encoder(x = iris$Sepal.Length, frame = c(3, 5, 7, 9))
enc$encode(x = c(4:8, NA, Inf))
```

plot.mid *Plot MID Component Functions*

Description

For "mid" objects (i.e., fitted MID models), `plot()` visualizes a single component function specified by the `term` argument.

Usage

```
## S3 method for class 'mid'
plot(
  x,
  term,
  type = c("effect", "data", "compound"),
  theme = NULL,
  intercept = FALSE,
  main.effects = FALSE,
  data = NULL,
  limits = NULL,
  jitter = 0.3,
  resolution = c(100L, 100L),
  ...
)
```

Arguments

`x`	a "mid" object to be visualized.
`term`	a character string specifying the component function to be plotted.
`type`	the plotting style. One of "effect", "data" or "compound".
`theme`	a character string or object defining the color theme. See `color.theme` for details.
`intercept`	logical. If `TRUE`, the intercept is added to the MID values.
`main.effects`	logical. If `TRUE`, main effects are included in the interaction plot.
`data`	a data frame to be plotted with the corresponding MID values. If not provided, data is automatically extracted from the function call.
`limits`	a numeric vector of length two specifying the limits of the plotting scale.
`jitter`	a numeric value specifying the amount of jitter for the data points.
`resolution`	an integer or vector of two integers specifying the resolution of the raster plot for interactions.
`...`	optional parameters to be passed to the graphing function. Possible arguments are "col", "fill", "pch", "cex", "lty", "lwd" and aliases of them.

Details

This is an S3 method for the `plot()` generic that produces a plot from a "mid" object, visualizing a component function of the fitted MID model.

The `type` argument controls the visualization style. The default, `type = "effect"`, plots the component function itself. In this style, the plotting method is automatically selected based on the effect's type: a line plot for quantitative main effects; a bar plot for qualitative main effects; and a filled contour (level) plot for interactions. The `type = "data"` option creates a scatter plot of `data`, colored by the values of the component function. The `type = "compound"` option combines both approaches, plotting the component function alongside the data points.

Value

`plot.mid()` produces a plot as a side-effect and returns `NULL` invisibly.

See Also

`interpret`, `ggmid`

Examples

```
data(diamonds, package = "ggplot2")
set.seed(42)
idx <- sample(nrow(diamonds), 1e4)
mid <- interpret(price ~ (carat + cut + color + clarity)^2, diamonds[idx, ])

# Plot a quantitative main effect
plot(mid, "carat")

# Plot a qualitative main effect
plot(mid, "clarity")

# Plot an interaction effect with data points and a raster layer
plot(mid, "carat:clarity", type = "compound", data = diamonds[idx, ])

# Use a different color theme
plot(mid, "clarity:color", theme = "RdBu")
```

---

plot.mid.breakdown *Plot MID Breakdowns*

Description

For "mid.breakdown" objects, `plot()` visualizes the breakdown of a prediction by component functions.

Usage

```
## S3 method for class 'mid.breakdown'
plot(
  x,
  type = c("waterfall", "barplot", "dotchart"),
  theme = NULL,
  terms = NULL,
  max.terms = 15L,
  width = NULL,
  vline = TRUE,
  catchall = "others",
  format = c("%t=%v", "%t"),
  ...
)
```

Arguments

x	a "mid.breakdown" object to be visualized.
type	the plotting style. One of "waterfall", "barplot" or "dotchart".
theme	a character string or object defining the color theme. See color.theme for details.
terms	an optional character vector specifying which terms to display.
max.terms	the maximum number of terms to display in the plot. Less important terms will be grouped into a "catchall" category.
width	a numeric value specifying the width of the bars.
vline	logical. If TRUE, a vertical line is drawn at the zero or intercept line.
catchall	a character string for the catchall label.
format	a character string or character vector of length two to be used as the format of the axis labels. Use "%t" for the term name (e.g., "carat") and "%v" for the values (e.g., "0.23").
...	optional parameters passed on to the graphing function. Possible arguments are "col", "fill", "pch", "cex", "lty", "lwd" and aliases of them.

Details

This is an S3 method for the plot() generic that produces a breakdown plot from a "mid.breakdown" object, visualizing the contribution of each component function to a single prediction.

The type argument controls the visualization style. The default, type = "waterfall", creates a waterfall plot that shows how the prediction builds from the intercept, with each term's contribution sequentially added or subtracted. The type = "barplot" option creates a standard bar plot where the length of each bar represents the magnitude of the term's contribution. The type = "dotchart" option creates a dot plot showing the contribution of each term as a point connected to a zero baseline.

Value

plot.mid.breakdown() produces a plot as a side effect and returns NULL invisibly.

See Also

mid.breakdown, ggmid.mid.breakdown

Examples

```
data(diamonds, package = "ggplot2")
set.seed(42)
idx <- sample(nrow(diamonds), 1e4)
mid <- interpret(price ~ (carat + cut + color + clarity)^2, diamonds[idx, ])
mbd <- mid.breakdown(mid, diamonds[1L, ])

# Create a waterfall plot
plot(mbd, type = "waterfall")

# Create a bar plot with a different theme
plot(mbd, type = "barplot", theme = "highlight")

# Create a dot chart
plot(mbd, type = "dotchart", size = 1.5)
```

---

plot.mid.conditional *Plot MID Conditional Expectations*

Description

For "mid.conditional" objects, `plot()` visualizes Individual Conditional Expectation (ICE) curves derived from a fitted MID model.

Usage

```
## S3 method for class 'mid.conditional'
plot(
  x,
  type = c("iceplot", "centered"),
  theme = NULL,
  term = NULL,
  var.alpha = NULL,
  var.color = NULL,
  var.linetype = NULL,
  var.linewidth = NULL,
  reference = 1L,
  dots = TRUE,
  sample = NULL,
  ...
)
```

Arguments

x	a "mid.conditional" object to be visualized.
type	the plotting style. One of "iceplot" or "centered".
theme	a character string or object defining the color theme. See color.theme for details.
term	an optional character string specifying an interaction term. If passed, the ICE curve for the specified term is plotted.
var.alpha	a variable name or expression to map to the alpha aesthetic.
var.color	a variable name or expression to map to the color aesthetic.
var.linetype	a variable name or expression to map to the linetype aesthetic.
var.linewidth	a variable name or expression to map to the linewidth aesthetic.
reference	an integer specifying the index of the sample points to use as the reference for centering the c-ICE plot.
dots	logical. If TRUE, points representing the actual predictions for each observation are plotted.
sample	an optional vector specifying the names of observations to be plotted.
...	optional parameters passed on to the graphing functions.

Details

This is an S3 method for the plot() generic that produces ICE curves from a "mid.conditional" object. ICE plots are a model-agnostic tool for visualizing how a model's prediction for a single observation changes as one feature varies. This function plots one line for each observation in the data.

The type argument controls the visualization style: The default, type = "iceplot", plots the row ICE curves. The type = "centered" option creates the centered ICE (c-ICE) plot, where each curve is shifted so start at zero, which makes it easier to compare the slopes of the curves.

The var.color, var.alpha, etc., arguments allow you to map aesthetics to other variables in your data using (possibly) unquoted expressions.

Value

plot.mid.conditional() produces an ICE plot as a side-effect and invisibly returns the ICE matrix used for the plot.

See Also

mid.conditional, ggmid.mid.conditional

Examples

```
data(airquality, package = "datasets")
library(midr)
mid <- interpret(Ozone ~ .^2, data = airquality, lambda = 0.1)
ice <- mid.conditional(mid, "Temp", data = airquality)
```

```
# Create an ICE plot, coloring lines by 'Wind'
plot(ice, var.color = "Wind")

# Create a centered ICE plot, mapping color and linetype to other variables
plot(ice, type = "centered", theme = "Purple-Yellow",
     var.color = factor(Month), var.linetype = Wind > 10)
```

`plot.mid.importance` *Plot MID Importance*

Description

For "mid.importance" objects, `plot()` visualizes the importance of component functions of the fitted MID model.

Usage

```
## S3 method for class 'mid.importance'
plot(
  x,
  type = c("barplot", "dotchart", "heatmap", "boxplot"),
  theme = NULL,
  max.terms = 30L,
  ...
)
```

Arguments

`x`	a "mid.importance" object to be visualized.
`type`	the plotting style. One of "barplot", "dotchart", "heatmap", or "boxplot".
`theme`	a character string or object defining the color theme. See `color.theme` for details.
`max.terms`	the maximum number of terms to display in the bar, dot and box plots.
`...`	optional parameters passed on to the graphing functions. Possible arguments are "col", "fill", "pch", "cex", "lty", "lwd" and aliases of them.

Details

This is an S3 method for the `plot()` generic that produces an importance plot from a "mid.importance" object, visualizing the average contribution of component functions to the fitted MID model.

The `type` argument controls the visualization style. The default, `type = "barplot"`, creates a standard bar plot where the length of each bar represents the overall importance of the term. The `type = "dotchart"` option creates a dot plot, offering a clean alternative to the bar plot for visualizing term importance. The `type = "heatmap"` option creates a matrix-shaped heat map where the color of each cell represents the importance of the interaction between a pair of variables, or the main

effect on the diagonal. The `type = "boxplot"` option creates a box plot where each box shows the distribution of a term's contributions across all observations, providing insight into the varibability of each term's effect.

Value

`plot.mid.importance()` produces a plot as a side effect and returns `NULL` invisibly.

See Also

`mid.importance`, `ggmid.mid.importance`

Examples

```
data(diamonds, package = "ggplot2")
set.seed(42)
idx <- sample(nrow(diamonds), 1e4)
mid <- interpret(price ~ (carat + cut + color + clarity)^2, diamonds[idx, ])
imp <- mid.importance(mid)

# Create a bar plot (default)
plot(imp)

# Create a dot chart
plot(imp, type = "dotchart", theme = "Okabe-Ito", size = 1.5)

# Create a heatmap
plot(imp, type = "heatmap")

# Create a boxplot to see the distribution of effects
plot(imp, type = "boxplot")
```

---

predict.mid *Predict Method for fitted MID Models*

Description

`predict.mid()` is an S3 method for "mid" objects that obtains predictions from a fitted MID model. It can be used to predict on new data or to retrieve the fitted values from the original data.

Usage

```
## S3 method for class 'mid'
predict(
  object,
  newdata = NULL,
  na.action = "na.pass",
  type = c("response", "link", "terms"),
  terms = object$terms,
```

```
  ...
)
```

Arguments

object	a "mid" object to be used to make predictions.
newdata	a data frame of the new observations. If NULL, the original fitted values are extracted and returned.
na.action	a function or character string specifying what should happen when the data contain NA values.
type	the type of prediction required. One of "response", "link", or "terms".
terms	a character vector of term labels, specifying a subset of component functions to use for predictions.
...	arguments to be passed to other methods (not used in this method).

Details

The type argument allows you to specify the scale of the prediction. By default (type = "response"), the function returns predictions on the original scale of the response variable. Alternatively, you can obtain predictions on the scale of the linear predictor by setting type = "link". For a detailed breakdown, setting type = "terms" returns a matrix where each column represents the contribution of a specific model term on the linear predictor scale.

The terms argument allows for predictions based on a subset of the model's component functions, excluding others.

Value

predict.mid() returns a numeric vector of MID model predictions, or a matrix if type = "terms".

See Also

interpret, mid.effect, get.yhat

Examples

```
data(airquality, package = "datasets")
test <- 1:10
mid <- interpret(Ozone ~ .^2, airquality[-test, ], lambda = 1, link = "log")

# Predict on new data
predict(mid, airquality[test, ])

# Get predictions on the link scale
predict(mid, airquality[test, ], type = "link")

# Get the contributions of specific terms
predict(mid, airquality[test, ], terms = c("Temp", "Wind"), type = "terms")
```

print.mid *Print MID Models*

Description

print.mid() is an S3 method for "mid" objects that prints a concise summary of a fitted MID model.

Usage

```
## S3 method for class 'mid'
print(x, digits = max(3L, getOption("digits") - 2L), main.effects = FALSE, ...)
```

Arguments

x	a "mid" object to be printed.
digits	an integer specifying the number of significant digits for printing.
main.effects	logical. If TRUE, the MID values of each main effect are also printed.
...	arguments to be passed to other methods (not used in this method).

Details

By default, the print() method for "mid" objects provides a quick overview of the model structure by listing the number of main effect and interaction terms. If main.effects = TRUE is specified, the method will also print the contribution of each main effect at its sample points, providing a more detailed look at the model's components.

Value

print.mid() returns the original "mid" object invisibly.

See Also

interpret, summary.mid

Examples

```
data(cars, package = "datasets")
mid <- interpret(dist ~ speed, cars)

# Default print provides a concise summary
print(mid)

# Setting main.effects = TRUE prints the contributions of each main effect
print(mid, main.effects = TRUE)
```

scale_color_theme *Color Theme Scales for ggplot2 Graphics*

Description

scale_color_theme() and its family of functions provide a unified interface to apply custom color themes to the colour and fill aesthetics of "ggplot" objects.

Usage

```
scale_color_theme(
  theme,
  ...,
  discrete = NULL,
  middle = 0,
  aesthetics = "colour"
)

scale_colour_theme(
  theme,
  ...,
  discrete = NULL,
  middle = 0,
  aesthetics = "colour"
)

scale_fill_theme(theme, ..., discrete = NULL, middle = 0, aesthetics = "fill")
```

Arguments

theme	a color theme name (e.g., "Viridis"), a character vector of color names, or a palette/ramp function. See ?color.theme for more details.
...	optional arguments to be passed to ggplot2::continuous_scale() or ggplot2::discrete_scale().
discrete	logical. If TRUE, a discrete scale is used regardless of the theme type.
middle	a numeric value specifying the middle point for the diverging color themes.
aesthetics	the aesthetic to be scaled. Can be "colour", "color", or "fill".

Details

This function automatically determines the appropriate **ggplot2** scale based on the theme's type. If the theme is "qualitative", a discrete scale is used by default to assign distinct colors to categorical data. The discrete argument is automatically set to TRUE if not specified. If the theme is "sequential" or "diverging", a continuous scale is used by default. The "diverging" themes are handled specially by scales::rescale_mid() to correctly center the gradient around the middle value.

Value

scale_color_theme() returns a ggplot2 scale object (either a "ScaleContinuous" or "ScaleDiscrete" object) that can be added to a "ggplot" object.

See Also

color.theme

Examples

```
data(txhousing, package = "ggplot2")
cities <- c("Houston", "Fort Worth", "San Antonio", "Dallas", "Austin")
df <- subset(txhousing, city %in% cities)
d <- ggplot2::ggplot(data = df, ggplot2::aes(x = sales, y = median)) +
  ggplot2::geom_point(ggplot2::aes(colour = city))

# Plot with a qualitative theme
d + scale_color_theme("Set 1")

# Use a sequential theme as a discrete scale
d + scale_color_theme("SunsetDark", discrete = TRUE)

data(faithfuld, package = "ggplot2")
v <- ggplot2::ggplot(faithfuld) +
  ggplot2::geom_tile(ggplot2::aes(waiting, eruptions, fill = density))

# Plot with continuous themes
v + scale_fill_theme("Plasma")

# Use a diverging theme with a specified midpoint
v + scale_fill_theme("midr", middle = 0.017)
```

---

set.color.theme *Register Color Themes*

Description

set.color.theme() registers a custom color theme in the package's theme registry.

Usage

```
set.color.theme(
  kernel,
  kernel.args = list(),
  options = list(),
  name = "newtheme",
  source = "custom",
  type = NULL,
  env = color.theme.env()
)
```

Arguments

kernel	a color vector, a palette function, or a ramp function to be used as a color kernel. It can also be a character vector or a list (see the "Details" section). A "color.theme" object can also be passed.
kernel.args	a list of arguments to be passed to the color kernel.
options	a list of option values to control the color theme's behavior.
name	a character string for the color theme name.
source	a character string for the source name of the color theme.
type	a character string specifying the type of the color theme. One of "sequential", "diverging", or "qualitative".
env	an environment where the color themes are registered.

Details

This function takes a color vector, a color-generating function, or an existing "color.theme" object and registers it under a specified `name` and `source` (default is "custom/newtheme"). The registered color theme can then be easily retrieved using the "Theme Name Syntax" (see `help(color.theme)`).

To keep the registry environment size small, the `kernel` argument supports a form of lazy loading. To use this feature, provide a vector or list containing two character strings. The first is an R expression that returns a color kernel (e.g., "rainbow"), and the second is the namespace in which to evaluate the expression (e.g., "grDevices"). The expression is evaluated only when the color theme is loaded by `color.theme()`.

Value

`set.color.theme()` returns the metadata of the previous theme that was overwritten (or `NULL` if none existed) invisibly.

See Also

`color.theme`, `color.theme.info`

---

shapviz.mid *Calculate MID-Derived Shapley Values*

Description

`shapviz.mid()` is an S3 method for the `shapviz::shapviz()` generic, which calculates MID-derived Shapley values from a fitted MID model.

Usage

```
## S3 method for class 'mid'
shapviz(object, data = NULL)
```

Arguments

object	a "mid" object.
data	a data frame containing the observations for which to calculate MID-derived Shapley values. If not passed, data is automatically extracted based on the function call.

Details

The function calculates MID-derived Shapley values by attributing the contribution of each component function to its respective variables as follows: first, each main effect is fully attributed to its corresponding variable; and then, each second-order interaction effect is split equally between the two variables involved.

Value

`shapviz.mid()` returns an object of class "shapviz".

---

summary.mid *Summarize MID Models*

Description

For "mid" objects, an S3 method of `summary()` prints a comprehensive summary of a fitted MID Model.

Usage

```
## S3 method for class 'mid'
summary(object, digits = max(3L, getOption("digits") - 2L), top.n = 10L, ...)
```

Arguments

object	a "mid" object to be summarized.
digits	the number of significant digits for printing numeric values.
top.n	the maximum number of top-ranked terms to be printed in the MID importance table.
...	arguments to be passed to other methods (not used in this method).

Details

The S3 method `summary.mid()` generates a comprehensive overview of the fitted MID model. The output includes the following components: (1) "Call" - the function call used to fit the MID model. (2) "Uninterpreted Variation Ratio" - a key metric indicating the proportion of the target model's variance that is not explained by the MID model. Lower values suggest a better fit. (3) "Residuals" - a five-number summary (Min, 1Q, Median, 3Q, Max) of the working residuals. This aids in assessing model fit and identifying potential biases. (4) "Encoding" - a summary of the encoding schemes used for each variable in the MID model. (5) "Importance" - a list of the top terms ranked by their MID importance, which quantifies their average contribution to the model's predictions.

Value

`summary.mid()` returns the original "mid" object invisibly.

See Also

`interpret`, `print.mid`

Examples

```
# Summarize a fitted MID model
data(cars, package = "datasets")
mid <- interpret(dist ~ speed, cars)
summary(mid)
```

---

theme_midr *Default Plotting Themes*

Description

`theme_midr()` returns a complete theme for "ggplot" objects, providing a consistent visual style for **ggplot2** plots.

`par.midr()` can be used to set graphical parameters for base R graphics.

Usage

```
theme_midr(
  grid_type = c("none", "x", "y", "xy"),
  base_size = 11,
  base_family = "serif",
  base_line_size = base_size/22,
  base_rect_size = base_size/22
)

par.midr(...)
```

Arguments

`grid_type`	the type of grid lines to display, one of "none", "x", "y" or "xy".
`base_size`	base font size, given in pts.
`base_family`	base font family.
`base_line_size`	base size for line elements.
`base_rect_size`	base size for rect elements.
`...`	for `par.midr()`, optional arguments in `tag = value` form to be passed to `graphics::par()`.

Value

`theme_midr()` provides a **ggplot2** theme customized for the **midr** package.

`par.midr()` returns the previous values of the changed parameters in an invisible named list.

Examples

```
# Use theme_midr() with ggplot2
X <- data.frame(x = 1:10, y = 1:10)
ggplot2::ggplot(X) +
  ggplot2::geom_point(ggplot2::aes(x, y)) +
  theme_midr()
ggplot2::ggplot(X) +
  ggplot2::geom_col(ggplot2::aes(x, y)) +
  theme_midr(grid_type = "y")
ggplot2::ggplot(X) +
  ggplot2::geom_line(ggplot2::aes(x, y)) +
  theme_midr(grid_type = "xy")

# Use par.midr() for base R graphics
old.par <- par.midr()
plot(y ~ x, data = X)
par(old.par)
```

---

weighted.loss *Weighted Loss Function*

Description

`weighted.loss()` computes various loss metrics (e.g., RMSE, MAE) between two numeric vectors, or for the deviations from the weighted mean of a numeric vector.

Usage

```
weighted.loss(x, y = NULL, w = NULL, na.rm = FALSE, method = "rmse")
```

Arguments

`x`	a numeric vector.
`y`	an optional numeric vector. If `NULL`, `x` is compared against its weighted mean.
`w`	a numeric vector of sample weights for each value in `x`.
`na.rm`	logical. If `TRUE`, any `NA` and `NaN`s are removed from all input vectors before the calculation.
`method`	the loss measure. One of "mse" (mean square error), "rmse" (root mean square error), "mae" (mean absolute error), or "medae" (median absolute error).

Value

`weighted.loss()` returns a single numeric value.

Examples

```
# Calculate loss metrics between x and y with weights
weighted.loss(x = c(0, 10), y = c(0, 0), w = c(99, 1), method = "rmse")
weighted.loss(x = c(0, 10), y = c(0, 0), w = c(99, 1), method = "mae")
weighted.loss(x = c(0, 10), y = c(0, 0), w = c(99, 1), method = "medae")

# Verify uninterpreted variation ratio of a fitted MID model without weights
mid <- interpret(dist ~ speed, cars)
RSS <- weighted.loss(cars$dist, predict(mid, cars), method = "mse")
TSS <- weighted.loss(cars$dist, method = "mse")
RSS / TSS
mid$ratio

# Verify uninterpreted variation ratio of a fitted MID model with weights
w <- 1:nrow(cars)
mid <- interpret(dist ~ speed, cars, weights = w)
RSS <- weighted.loss(cars$dist, predict(mid, cars), w = w, method = "mse")
TSS <- weighted.loss(cars$dist, w = w, method = "mse")
RSS / TSS
mid$ratio
```

Index