Package ‘cheem’

March 14, 2022

Title Interactively Explore the Support of Local Explanations with the Radial Tour

Version 0.2.0

Description Given a tree-based model, calculate the tree SHAP <arXiv:1802.03888>; <https://github.com/ModelOriented/treeshap> local explanation of every observation. View the data space, explanation space, and residual plot as ensemble graphic interactive on a shiny application. After an observation of interest is identified, the normalized variable importance of the local explanation is used as a 1D projection basis. The support of the local explanation is then explored by changing the basis with the use of the radial tour <doi:10.32614/RJ-2020-027>; <doi:10.1080/10618600.1997.10474754>.

Depends R (>= 3.5.0)

License MIT + file LICENSE

URL https://github.com/nspyrison/cheem/

BugReports https://github.com/nspyrison/cheem/issues

LinkingTo Rcpp

Imports Rcpp, data.table, spinifex (>= 0.3.3), ggplot2, plotly, magrittr, shiny, shinythemes, shinycssloaders, DT

Suggests gbm, lightgbm, randomForest, ranger, xgboost, tourr, lqmm, mvtnorm, gganimate, dplyr, tidyr, tictoc, beepr, knitr, testthat (>= 3.0.0), spelling, rmarkdown

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 7.1.2

Language en-US

Config/testthat/edition 3

NeedsCompilation yes

Author Nicholas Spyrison [aut, cre] (<https://orcid.org/0000-0002-8417-0212>)

Maintainer Nicholas Spyrison <spyrison@gmail.com>

Repository CRAN

Date/Publication 2022-03-14 11:40:02 UTC

R topics documented:

`amesHousing2018` *Ames housing data 2018*

Description

House sales prices from Ames, Iowa, USA between 2006 and 2010. Only complete numeric observations remain.

Usage

```
amesHousing2018

amesHousing2018_raw

amesHousing2018_NorthAmes
```

Format

complete data.frame with 2291 rows and 18 numeric variables, SalesPrice, the response variable, and 3 class variables

An object of class `data.frame` with 2930 rows and 82 columns.

An object of class `data.frame` with 338 rows and 11 columns.

Details

amesHousing2018 Complete data.frame, n = 2291, 18 numeric variable (including 2 temporal: MoSold, YrSold), response variable SalePrice, 3 class factors.

amesHousing2018_NorthAmes A simplified subsample, just North Ames (largest neighborhood). Complete data.frame, n = 338, 9 numeric variables, response variable SalePrice, 1 class factor SubclassMS, a zoning subclass.

amesHousing2018_raw Original data from Kaggle, 2930 rows of 82 variables. Sparse rows (639) and sparse/defaulted columns (64) are removed.

No data dictionary is provided on Kaggle, but amesHousing2018 variables are inferred to be:

- LotFrontage, Length of the front (street facing) side of the lot in yards (0.914m)
- LotArea, Area of the lot in square yards (0.836m^2)
- OverallQual, Overall quality (of the house?), integer in (1, 10)
- OverallCond, Overall condition (of the lot?), integer in (1, 10)
- YearBuild, The year the house was originally built
- BsmtUnfArea, Unfinished basement area, in square yards (0.836m^2)
- TotBsmtArea, Total basement area, in square yards (0.836m^2)
- 1stFlrArea, First (ground) floor living area in square yards (0.836m^2)
- LivingArea, Total living area in square yards (0.836m^2)

- Bathrms, The number of bathrooms
- Bedrms, The number of bedrooms
- TotRms, The total number of rooms
- GarageYrBlt, The year the garage was build
- GarageCars, The number of car spaces in the garage
- GarageArea, The area of the garage in square yards (0.836m^2)
- MoSold, The number of the month of the house sale
- YrSold, The number of the year of the house sale
- SalePrice, The sale of the house in USD (as of the year of sale?)
- SubclassMS, Factor subclass of construction zone, 16 levels
- SubclassMS, Factor major class of construction zone, 7 levels
- Neighborhd, Factor neighborhood of Ames, IA, 28 levels

Source

De Cock, D. (2011). "Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project," *Journal of Statistics Education*, Volume 19, Number 3. http://jse.amstat.org/v19n3/decock/DataDocumentation.txt http://jse.amstat.org/v19n3/decock.pdf

Kaggle, Ames Housing Dataset https://www.kaggle.com/prevek18/ames-housing-dataset

Replicating this dataset:

```
if(FALSE) ## Don't accidentally open the URL.
  browseURL("https://www.kaggle.com/prevek18/ames-housing-dataset")
ames <- readr::read_csv("./buildignore/AmesHousing.csv")
amesHousing2018_raw <- data.frame(ames)
## save(amesHousing2018_raw, file = "./data/amesHousing2018_raw.rda")

## Complete rows and numeric variables
ames1 <- ames[, unlist(lapply(ames, is.numeric))]
ames1$Bathrooms <- ames1$`Full Bath` + ames1$`Half Bath`
ames1 <- ames1[, c(1:18, 38, 19:37)]
col_idx <- !(colnames(ames1) %in% c(
  "Order", "Mas Vnr Area", "BsmtFin SF 1", "BsmtFin SF 2",
  "Bsmt Full Bath", "Bsmt Half Bath", "Fireplaces",
  "Wood Deck SF", "Open Porch SF", "Enclosed Porch",
  "3Ssn Porch", "Screen Porch", "Pool Area", "Misc Val", "2nd Flr SF",
  "Low Qual Fin SF", "Full Bath", "Half Bath", "Kitchen AbvGr"))
row_idx <- !is.na(ames1$"Garage Yr Blt") &
  !is.na(ames1$"Lot Frontage") &
  !is.na(ames1$"Bsmt Unf SF") &
  !is.na(ames1$"Total Bsmt SF")
ames2 <- as.data.frame(ames1[row_idx, col_idx])

## Looking for character classes to keep:
ames_char <- ames[, unlist(lapply(ames, is.character))]
```

```
ames_clas <- as.data.frame(lapply(ames_char, factor))[, -1]
ames_clasint <- data.frame(lapply(ames_clas, as.integer))
col_idx_char <- which(names(ames_clas) %in%
                        c("MS.SubClass", "MS.Zoning", "Neighborhood"))
classes <- ames_clas[row_idx, col_idx_char]

amesHousing2018 <- cbind(ames2, classes)
names(amesHousing2018) <- c(
  "LotFrontage", "LotArea","OverallQual", "OverallCond", "YearBuild",
  "YearRemod", "BsmtUnfArea", "TotBsmtArea", "1stFlrArea", "LivingArea",
  "Bathrms", "Bedrms", "TotRms", "GarageYrBlt", "GarageCars", "GarageArea",
  "MoSold", "YrSold", "SalePrice", "SubclassMS", "ZoneMS", "Neighborhd")
## save(amesHousing2018, file = "./data/amesHousing2018.rda")

.thin_col_idx <- names(amesHousing2018) %in% c(
  "LotArea", "OverallQual", "YearBuild",
  "LivingArea", "Bathrms", "Bedrms", "TotRms",
  "GarageYrBlt", "GarageArea", "SalePrice", "SubclassMS")
amesHousing2018_thin <- amesHousing2018[, .thin_col_idx]

## subset to north ames, and only 5 largest subclasses
r_idx <- amesHousing2018$Neighborhd == "NAmes" &
  amesHousing2018$SubclassMS %in% c("020", "050", "080", "090", "060")
amesHousing2018_NorthAmes <- amesHousing2018_thin[r_idx, ]
amesHousing2018_NorthAmes$SubclassMS <- factor(
  amesHousing2018_NorthAmes$SubclassMS,
  unique(amesHousing2018_NorthAmes$SubclassMS))
## save(amesHousing2018_NorthAmes, file = "./data/amesHousing2018_NorthAmes.rda")
```

Examples

```
library(cheem)

## Regression setup:
dat  <- amesHousing2018_NorthAmes[1:100, ]
X    <- dat[, 1:9]
Y    <- dat$SalePrice
clas <- dat$SubclassMS

## Model, treeSHAP explanation, cheem list:
rf_fit  <- default_rf(X, Y)
shap_df <- attr_df_treeshap(rf_fit, X, noisy = FALSE)
this_ls <- cheem_ls(X, Y, class = clas,
                    model = rf_fit,
                    attr_df = shap_df)

## Visualize:
global_view(this_ls)

## Save for used with shiny app (expects .rds):
```

```
if(FALSE) ## Don't accidentally save.
  saveRDS(this_ls, "./my_cheem_ls.rds")
```

---

as_logical_index *Assure a full length logical index*

Description

Suggests a alpha opacity to plot with as a function of the number of observation.

Usage

```
as_logical_index(index, n)
```

Arguments

index	A vector, typically a numeric row index of the data to coerce to a logical index.
n	Single numeric, the number of rows of the data use as a replicate return length.

Value

A logical index of length n.

See Also

Other cheem utility: basis_attr_df(), color_scale_of(), does_contain_nonnumeric(), is_discrete(), is_diverging(), linear_tform(), logistic_tform(), manip_var_of_attr_df(), problem_type(), rnorm_from()

Examples

```
library(cheem)

## Coerce a numeric index to logical:
as_logical_index(c(1, 4:10, 15), nrow(mtcars))
```

attr_df_treeshap — *Extract the full treeSHAP data.frame of a randomForest model*

Description

A data.frame of each observations treeSHAP variable attributions of a randomForest model. A wrapper for `treeshap::randomForest.unify` and `treeshap::treeshap`.

Usage

```
attr_df_treeshap(
  model,
  x,
  keep_heavy = FALSE,
  verbose = getOption("verbose"),
  noisy = getOption("verbose")
)
```

Arguments

`model`	A tree based model supported by `treeshap`: a model from `gbm`, `lightgbm`, `randomForest`, `ranger`, or `xgboost`.
`x`	The explanatory data (without response) to extract the local attributions from.
`keep_heavy`	Logical, if the heavy items "interactions", "unified_model", and "observations" should be kept. Defaults to FALSE.
`verbose`	Logical, if runtime should be printed. Defaults to TRUE.
`noisy`	Logical, if a tone should be played on completion. Defaults to TRUE.

Value

A data.frame of the local attributions for each observation.

See Also

Other cheem preprocessing: `cheem_ls()`, `default_rf()`

Examples

```
library(cheem)

## Regression setup:
dat  <- amesHousing2018_NorthAmes
X    <- dat[, 1:9]
Y    <- dat$SalePrice
clas <- dat$SubclassMS

rf_fit  <- default_rf(X, Y)
```

```
## Long runtime for full datasets or complex models:
shap_df <- attr_df_treeshap(rf_fit, X, noisy = FALSE)
this_ls <- cheem_ls(X, Y, class = clas,
                    model = rf_fit,
                    attr_df = shap_df)
global_view(this_ls)
```

---

basis_attr_df *Basis matrix, 1D, of the local attribution basis*

Description

Extract and format the 1D local attribution basis from the provided local explanation's attribution.

Usage

```
basis_attr_df(attr_df, rownum)
```

Arguments

`attr_df`	A data frame of local explanation attributions, such as a return from `attr_df_treeshap()`.
`rownum`	The rownumber of the observation.

Value

A matrix of the 1D basis.

See Also

Other cheem utility: `as_logical_index()`, `color_scale_of()`, `does_contain_nonnumeric()`, `is_discrete()`, `is_diverging()`, `linear_tform()`, `logistic_tform()`, `manip_var_of_attr_df()`, `problem_type()`, `rnorm_from()`

Examples

```
library(cheem)

## Regression setup:
dat  <- amesHousing2018_NorthAmes
X    <- dat[, 1:9]
Y    <- dat$SalePrice
clas <- dat$SubclassMS

## Model and treeSHAP explanation:
rf_fit  <- default_rf(X, Y)
shap_df <- attr_df_treeshap(rf_fit, X, noisy = FALSE)

## Attribution basis of one obs:
basis_attr_df(shap_df, rownum = 1)
```

---

cheem *cheem*

Description

Consider an arbitrary black-box model. Local explanations are a class of point-measures of the variable importance to the model for one location in the explanatory variables. The package `cheem` extracts the local attribution of all observations and first creates a 2D approximated space comparing data- and attribution-spaces. After a primary and comparison observation have been selected the attribution of the primary is used as the 1D basis for a manual tour. This tour changes the contribution from the variable that deviates the most from its expected value. By viewing the positions of the primary and comparison points, the analyst can scrutinize the explanation identify variable attribution leading to a misclassification or large residual.

Details

GitHub: `https://github.com/nspyrison/cheem`

See Also

`cheem_ls()` `run_app()`

---

cheem_ls *Preprocessing for use in shiny app*

Description

Performs the preprocessing steps needs to supply the plot functions `global_view()` and `radial_cheem_tour()` used in the shiny app.

Usage

```
cheem_ls(
  x,
  y,
  class = NULL,
  model,
  attr_df,
  basis_type = c("pca", "olda"),
  layer_name = utils::tail(class(model), 1),
  verbose = getOption("verbose"),
  keep_model = FALSE
)
```

Arguments

x	The explanatory variables of the model.
y	The target variable of the model.
class	The variable to group points by. Originally the *predicted* class.
model	A non-linear model, originally a randomForest::randomForest model fit, or a return from default_rf().
attr_df	A data frame of local explanation attributions, such as a return from attr_df_treeshap().
basis_type	The type of basis used to approximate the data and attribution space from. Expects "pca" or "olda" (requires clas). Defaults to "pca".
layer_name	Character layer name, typically the type of local attribution used. Defaults to the last class of the model.
verbose	Logical, if runtime should be printed. Defaults to TRUE.
keep_model	Logical, whether or not the heavy model object should be kept. Defaults to FALSE.

Value

A list of data.frames needed for the shiny application.

See Also

global_view() radial_cheem_tour() radial_cheem_tour()

Other cheem preprocessing: attr_df_treeshap(), default_rf()

Examples

```
library(cheem)
library(spinifex)

## Classification setup:
X    <- penguins_na.rm[, 1:4]
clas <- penguins_na.rm$species
Y    <- as.integer(clas)

## Model and treeSHAP explanation:
rf_fit  <- default_rf(X, Y)
shap_df <- attr_df_treeshap(rf_fit, X, noisy = FALSE)
this_ls <- cheem_ls(X, Y, class = clas,
                     model = rf_fit,
                     attr_df = shap_df)
global_view(this_ls) ## Preview spaces

## Save for used with shiny app (expects .rds):
if(FALSE){ ## Don't accidentally save.
  saveRDS(this_ls, "./my_cheem_ls.rds")
  run_app() ## Select the saved .rds file from the Data dropdown.
}
```

```
## Regression setup:
dat  <- amesHousing2018_NorthAmes
X    <- dat[, 1:9]
Y    <- dat$SalePrice
clas <- dat$SubclassMS

## Model and treeSHAP explanation:
rf_fit  <- default_rf(X, Y)

shap_df <- attr_df_treeshap(rf_fit, X, noisy = FALSE)
this_ls <- cheem_ls(X, Y, class = clas,
                     model = rf_fit,
                     attr_df = shap_df)
global_view(this_ls) ## Preview spaces


## Save for used with shiny app (expects .rds):
if(FALSE){ ## Don't accidentally save.
  saveRDS(this_ls, "./my_cheem_ls.rds")
  run_app() ## Select the saved .rds file from the Data dropdown.
}
```

chocolates *Chocolates dataset*

Description

The chocolates data was compiled by students at Iowa State University of STAT503 (circa 2015) taught by Dianne Cook. Nutrition label information on the chocolates as listed on manufacturer websites. All numbers were normalized to be equivalent to a 100g serving. Units of measurement are listed in the variable name.

Usage

```
chocolates
```

Format

A complete data.frame with 88 observations and 10 numeric variables, name of the chocolate, manufacturer, country, and type of the chocolate.

- Name, the name of the chocolate
- MFR, chocolate manufacturer
- Country, the country the manufacturer is incorporated.
- Type, the type of chocolate according to the website, either 'Dark' or 'Milk"
- Calories, the number of calories per 100 grams

- CalFat, calories from fat per 100 grams
- TotFat_g, grams of total fat per 100 grams
- SatFat_g, grams of saturated fat per 100 grams
- Chol_mg, milligrams of cholesterol per 100 grams
- Na_mg, milligrams of sodium (salt) per 100 grams
- Carbs_g, grams of carbohydrates per 100 grams
- Fiber_g, grams of fiber per 100 grams
- Sugars_g, grams of sugar per 100 grams
- Protein_g, grams of sugar per 100 grams

Source

Monash University, Introduction to Machine Learning course https://iml.numbat.space/

Replicating this dataset:

```
if(FALSE) ## Don't accidentally open the URL.
  browseURL("https://iml.numbat.space/")
## Accessed Jan 2022
chocolates <- readr::read_csv("https://iml.numbat.space/data/chocolates.csv")
chocolates <- data.frame(chocolates)
chocolates[, 2] <- factor(chocolates[, 2])
chocolates[, 3] <- factor(chocolates[, 3])
chocolates[, 4] <- factor(chocolates[, 4])
## save(chocolates, file = "./data/chocolates.rda")
```

Examples

```
library(cheem)

## Classification setup:
X    <- chocolates[, 5:14]
Y    <- as.integer(chocolates$Type)
clas <- chocolates$Type

## Model, treeSHAP explanation, cheem list:
rf_fit  <- default_rf(X, Y)
shap_df <- attr_df_treeshap(rf_fit, X, noisy = FALSE)
this_ls <- cheem_ls(X, Y, class = clas,
                    model = rf_fit,
                    attr_df = shap_df)

## Visualize:
global_view(this_ls)

## Save for used with shiny app (expects .rds):
if(FALSE) ## Don't accidentally save.
  saveRDS(this_ls, "./my_cheem_ls.rds")
```

`color_scale_of` *Suggest a color and fill scale.*

Description

Whether or not a vector is a diverges a value, returns a logical. Used to help default a scale_color for ggplot2.

Usage

```
color_scale_of(x, mid_pt = 0, limits = NULL, ...)
```

Arguments

`x`	A vector to to scale the color to.
`mid_pt`	A single number checking divergence from. Defaults to 0.
`limits`	A vector of the min and max values for the scale. Useful for setting an absolute range, such as (-1, 1) for attribution/y correlation of each point. Points outside of limits show as default grey. Defaults to NULL; the range of x.
`...`	Optional other arguments passed to ggplot2::continuous_scale or ggplot2::discrete_scale.

Value

A list containing a scale_color, and scale_fill; the suggested color/fill scale for a ggplot.

See Also

Other cheem utility: `as_logical_index()`, `basis_attr_df()`, `does_contain_nonnumeric()`, `is_discrete()`, `is_diverging()`, `linear_tform()`, `logistic_tform()`, `manip_var_of_attr_df()`, `problem_type()`, `rnorm_from()`

Examples

```
library(cheem)
library(ggplot2)
g <- ggplot(mtcars, aes(disp, mpg))

## Discrete
g + geom_point(aes(color = factor(vs))) +
  color_scale_of(mtcars$vs)
## Sequential increasing
g + geom_point(aes(color = mpg)) +
  color_scale_of(mtcars$mpg)
## Dummy sequential decr
g + geom_point(aes(color = -1 *mpg)) +
  color_scale_of(-1 * mtcars$mpg)
## Dummy diverging
g + geom_point(aes(color = mpg - median(mpg))) +
  color_scale_of(mtcars$mpg - median(mtcars$mpg))
## Dummy limits
g + geom_point(aes(color = mpg - median(mpg))) +
  color_scale_of(mtcars$mpg - median(mtcars$mpg), limits = c(-5, 5))
```

default_rf *Random forest model via randomForest*

Description

A wrapper function for `randomForest::randomForest` with more modest hyperparameter defaults and arguments consistent with `cheem`.

Usage

```
default_rf(
  x,
  y,
  verbose = getOption("verbose"),
  hp_ntree = 125,
  hp_mtry = ifelse(is_discrete(y), sqrt(ncol(x)), ncol(x)/3),
  hp_nodesize = max(ifelse(is_discrete(y), 1, 5), nrow(x)/500)
)
```

Arguments

`x`	The explanatory variables of the model.
`y`	The target variable of the model.
`verbose`	Logical, if runtime should be printed. Defaults to TRUE.
`hp_ntree`	Hyperparameter, the number of trees to grow.
`hp_mtry`	Hyperparameter, the number variables randomly sampled at each split.
`hp_nodesize`	Hyperparameter, the minimum size of terminal nodes. Setting this number larger causes smaller trees to be grown (and thus take less time).

Value

A randomForest model.

See Also

Other cheem preprocessing: `attr_df_treeshap()`, `cheem_ls()`

Examples

```
library(cheem)

## Regression setup:
dat  <- amesHousing2018_NorthAmes
X    <- dat[, 1:9]
Y    <- dat$SalePrice
clas <- dat$SubclassMS

## Model, treeSHAP, cheem list, visualize
rf_fit  <- default_rf(X, Y)
shap_df <- attr_df_treeshap(rf_fit, X, noisy = FALSE)
this_ls <- cheem_ls(X, Y, class = clas,
                    model = rf_fit,
                    attr_df = shap_df)
global_view(this_ls)
```

`devMessage` *Development message*

Description

Send a message if the 4th chunk of the package version is 9000.

Usage

```
devMessage(text)
```

Arguments

`text`	A character string to message() if package version is 9000.

`does_contain_nonnumeric` *Check if a vector contains non-numeric character*

Description

Returns a logical, whether or not a vector contains any non-numeric characters. Typically used to test if row names hold non-index information.

Usage

```
does_contain_nonnumeric(x)
```

Arguments

`x`	A vector to be tested for existence of non-numeric characters.

Value

Logical, whether or not x contains any non-numeric characters.

See Also

Other cheem utility: `as_logical_index()`, `basis_attr_df()`, `color_scale_of()`, `is_discrete()`, `is_diverging()`, `linear_tform()`, `logistic_tform()`, `manip_var_of_attr_df()`, `problem_type()`, `rnorm_from()`

Examples

```
library(cheem)

does_contain_nonnumeric(mtcars$mpg)
does_contain_nonnumeric(rownames(mtcars)) ## Meaningful info to use in tooltip
does_contain_nonnumeric(rownames(cars)) ## Assume no meaningful info to use in tooltip
```

`gbm.unify` *Unify GBM model*

Description

Convert your GBM model into a standardized representation. The returned representation is easy to be interpreted by the user and ready to be used as an argument in `treeshap()` function.

Usage

```
gbm.unify(gbm_model, data)
```

Arguments

`gbm_model`	An object of `gbm` class. At the moment, models built on data with categorical features are not supported - please encode them before training.
`data`	Reference dataset. A `data.frame` or `matrix` with the same columns as in the training set of the model. Usually dataset used to train model.

Value

a unified model representation - a `model_unified.object` object

Author(s)

Konrad Komisarczyk, Przemyslaw Biecek, et al.

Source

treeshap, https://github.com/ModelOriented/treeshap

See Also

`unify_tree_model`, a wrapper function unifying these models. `lightgbm.unify` for `LightGBM models` `xgboost.unify` for `XGBoost models` `ranger.unify` for `ranger models` `randomForest.unify` for `randomForest models`

Examples

```
library(cheem)

## Regression setup:
dat  <- amesHousing2018_NorthAmes
X    <- dat[, 1:9]
Y    <- dat$SalePrice
clas <- dat$SubclassMS

## Fit a model:
gbm_model <- gbm::gbm(
  formula = Y ~ .,
  data = data.frame(X, Y),
  distribution = "gaussian",
  n.trees = 50,
  interaction.depth = 4,
  n.cores = 1)
unified_model <- gbm.unify(gbm_model, X)

## Calculate treeSHAP:
shaps <- treeshap(unified_model, X[1:2,])
str(shaps)
```

`global_view` *Linked* `plotly` *display, global view of data and attribution space.*

Description

from a cheem_ls() list, create a linked `plotly` of the global data- and attribution- spaces. Typically consumed directly by shiny app.

Usage

```
global_view(
  cheem_ls,
  primary_obs = NULL,
  comparison_obs = NULL,
  color = c("default", "residual", "log_maha.data", "cor_attr_proj.y"),
  height_px = 480,
  width_px = 1440,
  as_ggplot = FALSE
)

global_view_subplots(
  cheem_ls,
  primary_obs = NULL,
  comparison_obs = NULL,
  color = c("default", "residual", "log_maha.data", "cor_attr_proj.y"),
  height_px = 480,
  width_px = 1440
)
```

Arguments

`cheem_ls`	A return from `cheem_ls()`, a list of data frames.
`primary_obs`	The rownumber of the primary observation. Its local attribution becomes the 1d projection basis, and the point it highlighted as a dashed line. Defaults to NULL, no highlighting applied.
`comparison_obs`	The rownumber of the comparison observation. Point is highlighted as a dotted line. Defaults to NULL, no highlighting applied.
`color`	The name of the column in cheem_ls$global_view_df to map to color. Expects c("default", "residual", "log_maha.data", "cor_attr_proj.y"). Defaults to "default"; predicted_class for classification, dummy class for regression.
`height_px`	The height in pixels of the returned `plotly` plot. Defaults to 480.
`width_px`	The width in pixels of the returned `plotly` plot. Defaults to 1440.
`as_ggplot`	Logical, if TRUE returns the plots before being passed to `plotly` functions.

Value

A `plotly` plot, an interactive html widget of the global view, first two components of the basis of the data- and attribution- spaces.

See Also

Other cheem consumers: `radial_cheem_tour()`, `run_app()`

Examples

```
library(cheem)

## Regression setup:
dat  <- amesHousing2018_NorthAmes
X    <- dat[, 1:9]
Y    <- dat$SalePrice
clas <- dat$SubclassMS

## Model, explanation, cheem list, global view:
```

```
rf_fit <- default_rf(X, Y)
shap_df <- attr_df_treeshap(rf_fit, X, noisy = FALSE)
this_ls <- cheem_ls(X, Y, class = clas,
                     model = rf_fit,
                     attr_df = shap_df)

## Visualize
global_view(this_ls)                   ## uses ggplot facets %>% plotly
global_view(this_ls, as_ggplot = TRUE) ## early return of ggplot
global_view_subplots(this_ls)          ## uses plotly::subplots

## Different color mappings, especially for regression

global_view_subplots(this_ls, color = "residual")
global_view_subplots(this_ls, color = "log_maha.data")
global_view_subplots(this_ls, color = "cor_attr_proj.y")
```

global_view_df_1layer *Create the plot data.frame for the global linked plotly display.*

Description

Internal function consumed in the cheem workflow. Produces the plot data.frame of 1 layer. consumed downstream in cheem_ls.

Usage

```
global_view_df_1layer(
  x,
  class = NULL,
  basis_type = c("pca", "olda"),
  layer_name = utils::tail(class(x), 1)
)
```

Arguments

x	The explanatory variables of the model.
class	The variable to group points by. Originally the *predicted* class.
basis_type	The type of basis used to approximate the data and attribution space from. Defaults to "pca".
layer_name	Character layer name, typically the type of local attribution used. Defaults to the name of the last class of x.

Value

A data.frame, for the global linked **plotly** display.

`ifDev` *Evaluate if development*

Description

Evaluate the expression if the 4th chunk of the package version is 9000.

Usage

```
ifDev(expr)
```

Arguments

`expr`	A character string to message() if package version is 9000.

`is.model_unified` *Check whether object is a valid model_unified object*

Description

Does not check correctness of representation, only basic checks

Usage

```
is.model_unified(x)
```

Arguments

`x`	an object to check

Value

boolean

`is_discrete` *Check if a vector is discrete*

Description

Whether or not a vector is a discrete variable, returns a logical. Typically used on the Y variable of a model.

Usage

```
is_discrete(x, na.rm = TRUE)
```

Arguments

`x`	A vector to check the discreteness of.
`na.rm`	Whether or not to remove NA values before testing discreteness. Defaults to TRUE.

Value

Logical, whether or not `x` is a discrete variable.

See Also

Other cheem utility: `as_logical_index()`, `basis_attr_df()`, `color_scale_of()`, `does_contain_nonnumeric()`, `is_diverging()`, `linear_tform()`, `logistic_tform()`, `manip_var_of_attr_df()`, `problem_type()`, `rnorm_from()`

Examples

```
library(cheem)

is_discrete(mtcars$mpg) ## Numeric column, with more than 25 unique values.
is_discrete(mtcars$cyl) ## Numeric column, labeled as discrete, because less than 25 unique values
is_discrete(letters)    ## Characters and factors labeled discrete.
```

`is_diverging` *Check if a vector diverges a value*

Description

Whether or not a vector is a diverges a value, returns a logical. Used to help default a scale_color for ggplot2.

Usage

```
is_diverging(x, mid_pt = 0)
```

Arguments

`x`	A vector to check the divergence of.
`mid_pt`	A single number checking divergence from. Defaults to 0.

Value

Logical, whether or not `x` is a diverges `mid_pt`.

See Also

Other cheem utility: `as_logical_index()`, `basis_attr_df()`, `color_scale_of()`, `does_contain_nonnumeric()`, `is_discrete()`, `linear_tform()`, `logistic_tform()`, `manip_var_of_attr_df()`, `problem_type()`, `rnorm_from()`

Examples

```
library(cheem)

is_diverging(-10:10)
is_diverging(-10:-5)
is_diverging(mtcars$mpg, 25)
is_diverging(mtcars$mpg, 40)
```

`is_randomForest` *Check model type*

Description

Check whether or not the model is a certain model. Checks if a model is made with: `randomForest::randomForest`, `ranger::ranger`, `gbm::gbm`, `xgboost::xgb.train`, `lightgbm::lightgbm`.

Usage

```
is_randomForest(model)

is_ranger(model)

is_gbm(model)

is_xgboost(model)

is_lightgbm(model)
```

Arguments

`model`	A model object to check the class/origin.

Value

A logical, whether or not the model is of a certain class.

See Also

Other cheem unify: `unify_tree_model()`

Examples

```
library(cheem)

## Regression setup:
dat  <- amesHousing2018_NorthAmes
X    <- dat[, 1:9]
Y    <- dat$SalePrice
clas <- dat$SubclassMS


# Treeshap handles various tree-based models:

## randomForest model
fit <- randomForest::randomForest(X, Y, ntree = 25)
is_randomForest(fit)

## gbm model
if(require(gbm, quietly = TRUE)){
  fit <- gbm::gbm(Y ~ ., "gaussian", data.frame(X, Y), n.trees = 25)
  is_gbm(fit)
}

## lightgbm
if(require(lightgbm, quietly = TRUE)){
  param_lgbm <- list(num_leaves = 25, objective = "regression")
  fit <- lightgbm::lightgbm(
    as.matrix(X), Y, params = param_lgbm,
    nrounds = 2, verbose = 0)
  ## Delete model file if it exists
  if(file.exists("lightgbm.model"))
    file.remove("lightgbm.model")
  is_lightgbm(fit)
}

## ranger model
if(require(ranger, quietly = TRUE)){
  fit <- ranger::ranger(Y ~ ., data.frame(X, Y), num.trees = 25)
  is_ranger(fit)
}

## xgboost
if(require(xgboost, quietly = TRUE)){
  fit <- xgboost::xgboost(as.matrix(X), Y, nrounds = 25, verbose = 0,
                          params = list(objective = "reg:squarederror"))
  is_xgboost(fit)
}


# Continue cheem workflow with tree-based models:

## treeSHAP, cheem list, visualize:
shap_df <- attr_df_treeshap(fit, X, verbose = TRUE, noisy = FALSE)
this_ls <- cheem_ls(X, Y, class = clas,
                    model = fit,
                    attr_df = shap_df)
global_view(this_ls)
```

lightgbm.unify *Unify LightGBM model*

Description

Convert your LightGBM model into a standardized representation. The returned representation is easy to be interpreted by the user and ready to be used as an argument in `treeshap()` function.

Usage

```
lightgbm.unify(lgb_model, data, recalculate = FALSE)
```

Arguments

`lgb_model`	A lightgbm model - object of class `lgb.Booster`
`data`	Reference dataset. A `data.frame` or `matrix` with the same columns as in the training set of the model. Usually dataset used to train model.
`recalculate`	logical indicating if covers should be recalculated according to the dataset given in data. Keep it `FALSE` if training data are used.

Value

a unified model representation - a `model_unified.object` object

Author(s)

Konrad Komisarczyk, Przemyslaw Biecek, et al.

Source

treeshap, `https://github.com/ModelOriented/treeshap`

See Also

`unify_tree_model`, a wrapper function unifying these models. `gbm.unify` for `GBM models` `xgboost.unify` for `XGBoost models` `ranger.unify` for `ranger models` `randomForest.unify` for `randomForest models`

Examples

```
library(cheem)

## Regression setup:
dat  <- amesHousing2018_NorthAmes
X    <- dat[, 1:9]
Y    <- dat$SalePrice
clas <- dat$SubclassMS

## Fit a model:
param_lgbm <- list(num_leaves = 50, objective = "regression")
lightgbm_model <- lightgbm::lightgbm(
  as.matrix(X), Y, params = param_lgbm,
  nrounds = 2, verbose = 0)
unified_model <- lightgbm.unify(lightgbm_model, X)
## Delete model file if it exists
if(file.exists("lightgbm.model"))
  file.remove("lightgbm.model")

## Calculate treeSHAP:
shaps <- treeshap(unified_model, X[1:2, ])
str(shaps)
```

`linear_tform` *Linear function to help set alpha opacity*

Description

Suggests a alpha opacity to plot with as a function of the number of observation.

Usage

```
linear_tform(n, appox_max_n = 5000L, ceiling = 1, floor = 0.3)
```

Arguments

`n`	Number of observations to plot.
`appox_max_n`	The number of observation to reach floor opacity.
`ceiling`	The highest number returned. Defaults to 1.
`floor`	The lowest number returned. Defaults to 0.3.

Value

A scalar numeric, suggested value to set alpha opacity.

See Also

Other cheem utility: as_logical_index(), basis_attr_df(), color_scale_of(), does_contain_nonnumeric(), is_discrete(), is_diverging(), logistic_tform(), manip_var_of_attr_df(), problem_type(), rnorm_from()

Examples

```
library(cheem)

## Suggest an opacity to use in plotting:
(my_alpha <- linear_tform(nrow(spinifex::penguins_na.rm)))

## Visualize
x <- 1:2000
plot(x, sapply(x, linear_tform), col = "blue")
```

---

logistic_tform *Logistic function to help set alpha opacity*

Description

Suggests a alpha opacity to plot with as a function of the number of observation.

Usage

```
logistic_tform(n, mid_pt = 600, k_attenuation = 5, ceiling = 1, floor = 0.3)
```

Arguments

n	Number of observations to plot.
mid_pt	Inflection point that the logistic curve. Defaults to 600.
k_attenuation	The steepness of the transition, larger is a sharper transition. Quite sensitive and defaults to 5.
ceiling	The highest number returned. Defaults to 1.
floor	The lowest number returned. Defaults to 0.3.

Value

A scalar numeric, suggested value to set alpha opacity.

See Also

Other cheem utility: as_logical_index(), basis_attr_df(), color_scale_of(), does_contain_nonnumeric(), is_discrete(), is_diverging(), linear_tform(), manip_var_of_attr_df(), problem_type(), rnorm_from()

Examples

```
library(cheem)

## Suggest an opacity to use in plotting:
(my_alpha <- logistic_tform(nrow(spinifex::penguins_na.rm)))

## Visualize
x <- 1:2000
plot(x, logistic_tform(x), col = "blue")
```

`manip_var_of_attr_df` *Find the manip var from a given attr_df*

Description

Find the number of the variable with the largest difference between the primary and comparison observations.

Usage

```
manip_var_of_attr_df(attr_df, primary_obs, comparison_obs)
```

Arguments

`attr_df`	A data frame of local explanation attributions, such as a return from `attr_df_treeshap()`.
`primary_obs`	The rownumber of the primary observation. Its local attribution becomes the 1d projection basis, and the point it highlighted as a dashed line.
`comparison_obs`	The rownumber of the comparison observation. Point is highlighted as a dotted line.

Value

A single number of the variable with the largest difference.

See Also

Other cheem utility: `as_logical_index()`, `basis_attr_df()`, `color_scale_of()`, `does_contain_nonnumeric()`, `is_discrete()`, `is_diverging()`, `linear_tform()`, `logistic_tform()`, `problem_type()`, `rnorm_from()`

Examples

```
library(cheem)

## Regression setup:
dat  <- amesHousing2018_NorthAmes
X    <- dat[, 1:9]
Y    <- dat$SalePrice
clas <- dat$SubclassMS
```

```
## Model and treeSHAP explanation:
rf_fit  <- default_rf(X, Y)
shap_df <- attr_df_treeshap(rf_fit, X, noisy = FALSE)

## Suggest the number of a variable to manipulate:
manip_var_of_attr_df(shap_df, primary_obs = 1, comparison_obs = 2)
```

`model_performance_df` *Extract higher level model performance statistics*

Description

Internal function, used downstream in cheem_ls.

Usage

```
model_performance_df(model, x = NULL, y = NULL)
```

Arguments

`model`	A non-linear model, originally a `randomForest::randomForest` model fit, or a return from `default_rf()`.
`x`	Data to predict, required by ranger models.
`y`	Observed response, required by ranger models.

Value

A data.frame of model performance statistics.

`model_unified.object` *Unified model representation*

Description

`model_unified` object produced by `*.unify` function.

Value

List consisting of two elements:

model - A `data.frame` representing model with following columns:

`Tree`	0-indexed ID of a tree
`Node`	0-indexed ID of a node in a tree. In a tree the root always has ID 0
`Feature`	In case of an internal node - name of a feature to split on. Otherwise - NA

`Decision.type`	A factor with two levels: "<" and "<=". In case of an internal node - predicate used for splitting observations. Otherwise - NA
`Split`	For internal nodes threshold used for splitting observations. All observations that satisfy the predicate Decision.type(Split) ('< Split' / '<= Split') are proceeded to the node marked as 'Yes'. Otherwise to the 'No' node. For leaves - NA
`Yes`	Index of a row containing a child Node. Thanks to explicit indicating the row it is much faster to move between nodes
`No`	Index of a row containing a child Node
`Missing`	Index of a row containing a child Node where are proceeded all observations with no value of the dividing feature
`Prediction`	For leaves: Value of prediction in the leaf. For internal nodes: NA
`Cover`	Number of observations seen by the internal node or collected by the leaf for the reference dataset

data - Dataset used as a reference for calculating SHAP values. A dataset passed to the `*.unify` or `set_reference_dataset` function with `data` argument. A `data.frame`.

Object has two also attributes set:

`model`	A string. By what package the model was produced.
`missing_support`	A boolean. Whether the model allows missing values to be present in explained dataset.

See Also

`unify_tree_model`, a wrapper function unifying these models. `lightgbm.unify` for `LightGBM models` `gbm.unify` for `GBM models` `xgboost.unify` for `XGBoost models` `ranger.unify` for `ranger models` `randomForest.unify` for `randomForest models`

`print.model_unified` *Prints model_unified objects*

Description

Prints model_unified objects

Usage

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

Arguments

`x`	a model_unified object
`...`	other arguments

Value

Prints a data.frame of the $model of the treeshap unified model.

`print.treeshap` *Prints treeshap objects*

Description

Prints treeshap objects

Usage

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

Arguments

`x`	a treeshap object
`...`	other arguments

Value

A data.frame of $shaps of the treeshap object. Also prints interactions if used.

`problem_type` *The type of model for a given Y variable*

Description

Whether the Y is a "classification", "regression" or ill-defined problem. Returns a character: "classification", "regression", or an error for strange classes. Minor redundancy with is_discrete, though explicit. Could be useful for `DALEX::explain(type)` as it also expects "classification" or "regression".

Usage

```
problem_type(y)
```

Arguments

`y`	Response variable to be modeled

Value

Character either c("classification", "regression") specifying the assumed model task based on the discreteness of y.

See Also

Other cheem utility: `as_logical_index()`, `basis_attr_df()`, `color_scale_of()`, `does_contain_nonnumeric()`, `is_discrete()`, `is_diverging()`, `linear_tform()`, `logistic_tform()`, `manip_var_of_attr_df()`, `rnorm_from()`

Examples

```
library(cheem)

problem_type(mtcars$mpg)
problem_type(mtcars$cyl) ## Numeric column, labeled as discrete, because less than 25 unique values
problem_type(letters)
```

`proto_basis1d_distribution`
Adds the distribution of the row local attributions to a ggtour

---

Description

A spinifex proto_*-like function, that adds the distribution of orthonormalized row values of the specified local explanation `attr_df`. Does not draw the basis bars; use in conjunction with `proto_basis1d()`.

Usage

```
proto_basis1d_distribution(
  attr_df,
  primary_obs = NULL,
  comparison_obs = NULL,
  position = c("top1d", "floor1d", "bottom1d", "off"),
  group_by = as.factor(FALSE),
  pcp_shape = c(3, 142, 124),
  do_add_pcp_segments = TRUE,
  inc_var_nms = NULL,
  row_index = NULL
)
```

Arguments

`attr_df`	An data frame, the attributions of a local explanation, such as a return from `attr_df_treeshap()`.
`primary_obs`	The rownumber of the primary observation. Its local attribution becomes the 1d projection basis, and the point it highlighted as a dashed line. Defaults to NULL, no highlighting.
`comparison_obs`	The rownumber of the comparison observation. Point is highlighted as a dotted line. Defaults to NULL, no highlighting.
`position`	The position for the basis, one of: c("top1d", "floor1d", "bottom1d", "off"). Defaults to "top1d"; basis above the density curves.

`group_by`	Vector to group densities by. Originally *predicted* class.
`pcp_shape`	The number of the shape character to add. Expects 3, 142, or 124, '+', '|' in `plotly`, and '|' in `gganimate`, respectively. Defaults to 3, '+' in either output.
`do_add_pcp_segments`	Logical, whether or not to add to add faint parallel coordinate lines on the 1D basis. Defaults to TRUE.
`inc_var_nms`	A character vector, the names of the variables to keep. Defaults to NULL, all variables kept.
`row_index`	A numeric or logical vector, the index of the rows to keep. Defaults to NULL, all rows kept.

Value

A `ggplot` object of the the distribution of the local explanation's attributions.

Examples

```
library(cheem)
library(spinifex)

## Regression setup:
dat  <- amesHousing2018_NorthAmes
X    <- dat[, 1:9]
Y    <- dat$SalePrice
clas <- dat$SubclassMS

## Model and treeSHAP explanation:
rf_fit  <- default_rf(X, Y)
shap_df <- attr_df_treeshap(rf_fit, X, noisy = FALSE)

## Basis, manipulation var, manual tour path, & predictions to fix to y-axis
bas     <- basis_attr_df(shap_df, 1)
mv      <- manip_var_of_attr_df(shap_df, 1, 2)
mt_path <- manual_tour(bas, mv)
pred    <- unify_predict(rf_fit, X)

## Compose and animate the tour
ggt <- ggtour(mt_path, scale_sd(X), angle = .3) +
  append_fixed_y(fixed_y = scale_sd(pred)) +
  proto_point(list(color = clas, shape = clas)) +
  proto_basis1d_distribution(
    attr_df = shap_df,
    primary_obs = 1, comparison_obs = 2,
    position = "top1d", group_by = clas) +
  proto_basis1d(position = "bottom1d") +
  proto_origin()

animate_plotly(ggt)
```

radial_cheem_tour *Cheem tour; 1D manual tour on the selected attribution*

Description

Create a linked `plotly`of the global data- and attribution- spaces. Typically consumed directly by shiny app.

Usage

```
radial_cheem_tour(
  cheem_ls,
  basis,
  manip_var,
  primary_obs = NULL,
  comparison_obs = NULL,
  do_add_pcp_segments = TRUE,
  pcp_shape = c(3, 142, 124),
  angle = 0.15,
  row_index = NULL,
  inc_var_nms = NULL,
  do_center_frame = TRUE,
  do_add_residual = FALSE
)

radial_cheem_tour_subplots(
  cheem_ls,
  basis,
  manip_var,
  primary_obs = NULL,
  comparison_obs = NULL,
  do_add_pcp_segments = TRUE,
  pcp_shape = c(3, 142, 124),
  angle = 0.15,
  row_index = NULL,
  inc_var_nms = NULL,
  do_center_frame = TRUE
)
```

Arguments

`cheem_ls`	A return from `cheem_ls()`, a list of data frames.
`basis`	A 1D projection basis, typically a return of `basis_attr_df()`.
`manip_var`	The , *number* of the manipulation variable.
`primary_obs`	The rownumber of the primary observation. Its local attribution becomes the 1d projection basis, and the point it highlighted as a dashed line. Defaults to NULL, no primary observation highlighted.

`comparison_obs`	The rownumber of the comparison observation. Point is highlighted as a dotted line. Defaults to NULL, no comparison observation highlighted.
`do_add_pcp_segments`	Logical, whether or not to add parallel coordinate line segments to the basis display.
`pcp_shape`	The number of the shape character to add. Expects 3, 142, or 124, '+', '|' in `plotly`, and '|' in `gganimate`, respectively. Defaults to 3, '+' in either output.
`angle`	The step size between interpolated frames, in radians. Defaults to .15.
`row_index`	Numeric index of selected observations. Defaults to TRUE; 1:n.
`inc_var_nms`	A vector of the names of the variables to include in the projection.
`do_center_frame`	Whether or not to scale by standard deviations away from the mean within each frame or not. Defaults to TRUE, helping to keep the animation centered.
`do_add_residual`	Whether of not to add a facet with a fixed y on residual. Doing so may cause issues with animation. Defaults to FALSE.

Value

ggtour (`ggplot2` object with frame info) animation frames of a radial tour manipulating the contribution of a selected tour. Animated with spinifex::animate_* functions.

See Also

Other cheem consumers: `global_view()`, `run_app()`

Examples

```
library(cheem)
library(spinifex)

## Classification setup:
X    <- penguins_na.rm[, 1:4]
clas <- penguins_na.rm$species
Y    <- as.integer(clas)

## Model and tree SHAP explanation:
rf_fit  <- default_rf(X, Y)
shap_df <- attr_df_treeshap(rf_fit, X, noisy = FALSE)
this_ls <- cheem_ls(X, Y, class = clas,
                     model = rf_fit,
                     attr_df = shap_df)

## Basis & suggest manipulation var
bas <- basis_attr_df(shap_df, rownum = 1)
mv  <- manip_var_of_attr_df(shap_df, primary_obs = 1, comparison_obs = 2)

## Radial tour with ggplot facets & animate
ggt <- radial_cheem_tour(this_ls, basis = bas, manip_var = 1)
```

```
animate_plotly(ggt)
if(FALSE) ## or animate with gganimate
  animate_gganimate(ggt, render = gganimate::av_renderer())

## Radial tour using plotly::subplots, not compatible with gganimate.
ggt <- radial_cheem_tour_subplots(this_ls, basis = bas, manip_var = 1)
animate_plotly(ggt)



## Regression setup:
dat  <- amesHousing2018_NorthAmes
X    <- dat[, 1:9]
Y    <- dat$SalePrice
clas <- dat$SubclassMS

## Model and tree SHAP explanation:
rf_fit  <- default_rf(X, Y)
shap_df <- attr_df_treeshap(rf_fit, X, noisy = FALSE)
this_ls <- cheem_ls(X, Y, class = clas,
                     model = rf_fit,
                     attr_df = shap_df)

## Basis & suggest manipulation var
bas <- basis_attr_df(shap_df, rownum = 1)
mv  <- manip_var_of_attr_df(shap_df, primary_obs = 1, comparison_obs = 2)

## Radial tour with ggplot facets & animate
ggt <- radial_cheem_tour(this_ls, basis = bas, manip_var = 1)

animate_plotly(ggt)
if(FALSE) ## or animate with gganimate
  animate_gganimate(ggt, render = gganimate::av_renderer())

## Radial tour using plotly::subplots, not compatible with gganimate.
ggt <- radial_cheem_tour_subplots(this_ls, basis = bas, manip_var = 1)
animate_plotly(ggt)
```

`randomForest.unify` *Unify randomForest model*

Description

Convert your randomForest model into a standardized representation. The returned representation is easy to be interpreted by the user and ready to be used as an argument in `treeshap()` function.

Usage

```
randomForest.unify(rf_model, data)
```

Arguments

`rf_model`	An object of `randomForest` class. At the moment, models built on data with categorical features are not supported - please encode them before training.
`data`	Reference dataset. A `data.frame` or `matrix` with the same columns as in the training set of the model. Usually dataset used to train model.

Value

a unified model representation - a `model_unified.object` object

Author(s)

Konrad Komisarczyk, Przemyslaw Biecek, et al.

Source

treeshap, `https://github.com/ModelOriented/treeshap`

See Also

`unify_tree_model`, a wrapper function unifying these models. `lightgbm.unify` for `LightGBM models` `gbm.unify` for `GBM models` `xgboost.unify` for `XGBoost models` `ranger.unify` for `ranger models`

Examples

```
library(cheem)

## Regression setup:
dat  <- amesHousing2018_NorthAmes
X    <- dat[, 1:9]
Y    <- dat$SalePrice
clas <- dat$SubclassMS

## Fit a model:
rf_model <- default_rf(X, Y)
unified_model <- randomForest.unify(rf_model, X)

## Calculate treeSHAP:
shaps <- treeshap(unified_model, X[1:2, ])
str(shaps)
```

`ranger.unify` *Unify ranger model*

Description

Convert your ranger model into a standardized representation. The returned representation is easy to be interpreted by the user and ready to be used as an argument in `treeshap()` function.

Usage

```
ranger.unify(rng_model, data)
```

Arguments

`rng_model`	An object of `ranger` class. At the moment, models built on data with categorical features are not supported - please encode them before training.
`data`	Reference dataset. A `data.frame` or `matrix` with the same columns as in the training set of the model. Usually dataset used to train model.

Value

a unified model representation - a `model_unified.object` object

Author(s)

Konrad Komisarczyk, Przemyslaw Biecek, et al.

Source

treeshap, `https://github.com/ModelOriented/treeshap`

See Also

`unify_tree_model`, a wrapper function unifying these models. `lightgbm.unify` for `LightGBM models` `gbm.unify` for `GBM models` `xgboost.unify` for `XGBoost models` `randomForest.unify` for `randomForest models`

Examples

```
library(cheem)

## Regression setup:
dat  <- amesHousing2018_NorthAmes
X    <- dat[, 1:9]
Y    <- dat$SalePrice
clas <- dat$SubclassMS

## Fit a model:
```

```
ranger_model  <- ranger::ranger(Y ~ ., data.frame(X, Y), num.trees = 25)
unified_model <- ranger.unify(ranger_model, X)

## Calculate treeSHAP:
shaps <- treeshap(unified_model, X[1:2, ])
str(shaps)
```

rnorm_from — *Draw new samples from the supplied data given its mean and covariances.*

Description

Creates new observation of the data given its specific means and shapes. typically applied to a cluster subset of data. *ie* draw from cluster 'a', then assign to cluster 'b'.

Usage

```
rnorm_from(
  data,
  n_obs = 1,
  var_coeff = 1,
  method = c("pearson", "kendall", "spearman")
)
```

Arguments

`data`	A data.frame or matrix to sample from.
`n_obs`	Number of new observations to draw. Defaults to 1.
`var_coeff`	Variance coefficient, closer to 0 make points near the median, above 1 makes more points further away from the median. Defaults to 1.
`method`	The method of the covariance matrix. Expects "person" (continuous numeric), "kendall" or "spearman" (latter two are ranked based ordinal).

Value

A data.frame, sampled observations given the means and covariance of the data based on with column names kept.

See Also

Other cheem utility: `as_logical_index()`, `basis_attr_df()`, `color_scale_of()`, `does_contain_nonnumeric()`, `is_discrete()`, `is_diverging()`, `linear_tform()`, `logistic_tform()`, `manip_var_of_attr_df()`, `problem_type()`

Examples

```
library(cheem)

sub <- mtcars[mtcars$cyl == 6, ]
## Draw 3 new observations in the shape of 6 cylinder vehicles, with reduced variance.
rnorm_from(data = sub, n_obs = 3, var_coeff = .5)
```

---

run_app *Runs a shiny app demonstrating manual tours*

Description

Runs a local shiny app that demonstrates manual tour and comparable traditional techniques for static projections of multivariate data sets.

Usage

```
run_app(app_nm = "cheem_initial", ...)
```

Arguments

app_nm	Name of the shiny app to run. Expects "cheem_initial".
...	Other arguments passed into `shiny::runApp()`. Such as display.mode = "showcase".

Value

Runs a locally hosted shiny app.

See Also

Other cheem consumers: `global_view()`, `radial_cheem_tour()`

Examples

```

## Runs the app
run_app("cheem_initial")

## Run with app code displayed
run_app(app_nm = "cheem_initial", display.mode = "showcase")
```

set_reference_dataset *Set reference dataset*

Description

Change a dataset used as reference for calculating SHAP values. Reference dataset is initially set with `data` argument in unifying function. Usually reference dataset is dataset used to train the model. Important property of reference dataset is that SHAPs for each observation add up to its deviation from mean prediction of reference dataset.

Usage

```
set_reference_dataset(unified_model, x)
```

Arguments

`unified_model`	Unified model representation of the model created with a (model).unify function. (`model_unified.object`).
`x`	Reference dataset. A `data.frame` or `matrix` with the same columns as in the training set of the model.

Value

`model_unified.object`. Unified representation of the model as created with a (model).unify function, but with changed reference dataset (Cover column containing updated values).

See Also

`unify_tree_model`, a wrapper function unifying these models. `lightgbm.unify` for `LightGBM models` `gbm.unify` for `GBM models` `xgboost.unify` for `XGBoost models` `ranger.unify` for `ranger models` `randomForest.unify` for `randomForest models`

Examples

```
library(cheem)

## Regression setup:
dat  <- amesHousing2018_NorthAmes
X    <- dat[, 1:9]
Y    <- dat$SalePrice
clas <- dat$SubclassMS

## Fit a model:
gbm_model <- gbm::gbm(
  formula = Y ~ .,
  data = data.frame(X, Y),
  distribution = "gaussian",
  n.trees = 50,
    interaction.depth = 4,
    n.cores = 1)
unified <- gbm.unify(gbm_model, X)
set_reference_dataset(unified, X[50:100, ])
```

---

treeshap *Calculate SHAP values of a tree ensemble model.*

Description

Calculate SHAP values and optionally SHAP Interaction values.

Usage

```
treeshap(unified_model, x, interactions = FALSE, verbose = TRUE)
```

Arguments

`unified_model`	Unified data.frame representation of the model created with a (model).unify function. A `model_unified.object` object.
`x`	Observations to be explained. A `data.frame` or `matrix` object with the same columns as in the training set of the model. Keep in mind that objects different than `data.frame` or plain `matrix` will cause an error or unpredictable behavior.
`interactions`	Whether to calculate SHAP interaction values. By default is `FALSE`. Basic SHAP values are always calculated.
`verbose`	Whether to print progress bar to the console. Should be logical. Progress bar will not be displayed on Windows.

Value

A `treeshap.object` object. SHAP values can be accessed with `$shaps`. Interaction values can be accessed with `$interactions`.

Author(s)

Konrad Komisarczyk, Przemyslaw Biecek, et al.

Source

treeshap, `https://github.com/ModelOriented/treeshap`

See Also

`unify_tree_model`, a wrapper function unifying these models. `xgboost.unify` for `XGBoost models` `lightgbm.unify` for `LightGBM models` `gbm.unify` for `GBM models` `randomForest.unify` for `randomForest models` `ranger.unify` for `ranger models`

Examples

```
library(cheem)

## Regression setup:
dat  <- amesHousing2018_NorthAmes
X    <- dat[, 1:9]
Y    <- dat$SalePrice
clas <- dat$SubclassMS

## Fit a model:
rf_fit <- default_rf(X, Y)
unified_model <- unify_tree_model(rf_fit, X)

## Calculate SHAP explanation for X, a subset, or OOS observations.
shaps <- treeshap(unified_model, head(X, 3))
str(shaps)
```

`treeshap.object` *treeshap results*

Description

`treeshap` object produced by `treeshap` function.

Value

List consisting of four elements:

shaps A `data.frame` with M columns, X rows (M - number of features, X - number of explained observations). Every row corresponds to SHAP values for a observation.

interactions An `array` with dimensions (M, M, X) (M - number of features, X - number of explained observations). Every `[, , i]` slice is a symmetric matrix - SHAP Interaction values for a observation. `[a, b, i]` element is SHAP Interaction value of features `a` and `b` for observation `i`. Is `NULL` if interactions where not calculated (parameter `interactions` set `FALSE`.)

unified_model An object of type `model_unified.object`. Unified representation of a model for which SHAP values were calculated. It is used by some of the plotting functions.

observations Explained dataset. `data.frame` or `matrix`. It is used by some of the plotting functions.

See Also

`treeshap`,

`unify_tree_model` *Unify various models/predictions to a standard format*

Description

Unifies models/prediction functions for supported by tree-based models into a standard format.

Usage

```
unify_tree_model(model, x)

unify_predict(model, x)
```

Arguments

`model`	A tree based model supported by `treeshap`: a model from `randomForest::randomForest`, `ranger::ranger`, `gbm::gbm`, `xgboost::xgb.train`, or `lightgbm::lightgbm`.
`x`	The explanatory data (without response) to extract the local attributions from.

Value

A vector of predicted values

See Also

Other cheem unify: `is_randomForest()`

Examples

```
library(cheem)

## Regression setup:
dat  <- amesHousing2018_NorthAmes
X    <- dat[, 1:9]
Y    <- dat$SalePrice
clas <- dat$SubclassMS

## Model, unified prediction from any model (not unified)
rf_fit <- default_rf(X, Y)
unify_predict(rf_fit, X)

## Applies the correct treeshap::*.unify for the model type.
unified_model <- unify_tree_model(rf_fit, X)
str(unified_model)
```

xgboost.unify *Unify xgboost model*

Description

Convert your xgboost model into a standardized representation. The returned representation is easy to be interpreted by the user and ready to be used as an argument in `treeshap()` function.

Usage

```
xgboost.unify(xgb_model, data, recalculate = FALSE)
```

Arguments

`xgb_model`	A xgboost model - object of class `xgb.Booster`
`data`	Reference dataset. A `data.frame` or `matrix` with the same columns as in the training set of the model. Usually dataset used to train model.
`recalculate`	logical indicating if covers should be recalculated according to the dataset given in data. Keep it `FALSE` if training data are used.

Value

a unified model representation - a `model_unified.object` object

Author(s)

Konrad Komisarczyk, Przemyslaw Biecek, et al.

Source

treeshap, `https://github.com/ModelOriented/treeshap`

See Also

`unify_tree_model`, a wrapper function unifying these models. `lightgbm.unify` for `LightGBM models` `gbm.unify` for `GBM models` `ranger.unify` for `ranger models` `randomForest.unify` for `randomForest models`

Examples

```
library(cheem)

## Regression setup:
dat  <- amesHousing2018_NorthAmes
X    <- dat[, 1:9]
Y    <- dat$SalePrice
clas <- dat$SubclassMS
```

```
## Fit a model:
xgb_model <- xgboost::xgboost(as.matrix(X), Y, nrounds = 25, verbose = 0,
                              params = list(objective = "reg:squarederror"))
unified_model <- xgboost.unify(xgb_model, X)

## Calculate treeSHAP:
shaps <- treeshap(unified_model, X[1:2, ])
str(shaps)
```

Index