Package 'SHAPforxgboost'

May 14, 2020

Title SHAP Plots for 'XGBoost'

Version 0.0.4

Description The aim of 'SHAPforxgboost' is to aid in visual data investigations using SHAP (SHapley Additive exPlanation) visualization plots for 'XGBoost'. It provides summary plot, dependence plot, interaction plot, and force plot. It relies on the 'dmlc/xgboost' package to produce SHAP values. Please refer to 'slundberg/shap' for the original implementation of SHAP in 'Python'.

License MIT + file LICENSE

URL https://github.com/liuyanguu/SHAPforxgboost

BugReports https://github.com/liuyanguu/SHAPforxgboost/issues

Encoding UTF-8

LazyData true

Depends R (>= 3.3.0)

Imports ggplot2 (>= 3.0.0), xgboost (>= 0.81.0.0), data.table (>= 1.12.0), ggforce (>= 0.2.1.9000), ggExtra (>= 0.8), RColorBrewer (>= 1.1.2), ggpubr, BBmisc

Suggests gridExtra (>= 2.3), here, parallel

RoxygenNote 7.0.2

NeedsCompilation no

Author Yang Liu [aut, cre] (<https://orcid.org/0000-0001-6557-6439>), Allan Just [aut, ctb] (<https://orcid.org/0000-0003-4312-5957>)

Maintainer Yang Liu <lyhello@gmail.com>

Repository CRAN

Date/Publication 2020-05-14 04:50:02 UTC

R topics documented:

dataXY_df 2
label.feature 3
labels_within_package 3

new_labels 4
plot.label 4
scatter.plot.diagonal 5
scatter.plot.simple 6
shap.plot.dependence 7
shap.plot.force_plot 9
shap.plot.force_plot_bygroup 10
shap.plot.summary 10
shap.plot.summary.wrap1 12
shap.plot.summary.wrap2 13
shap.prep 14
shap.prep.interaction 16
shap.prep.stack.data 17
shap.values 18
shap_int_iris 19
shap_long_iris 19
shap_score 20
shap_values_iris 20

Index **21**

`dataXY_df` *Terra satellite data (X,Y) for running the xgboost model .*

Description

Data.table, contains 9 features, and about 10,000 observations

Usage

```
dataXY_df
```

Format

An object of class `data.table` (inherits from `data.frame`) with 10148 rows and 10 columns.

References

http://doi.org/10.5281/zenodo.3334713

`label.feature` *helper function to modify labels for features under plotting*

Description

If a list is created in the global environment named **new_labels** (`!is.null(new_labels)`, the plots will use that list to replace default list of labels `labels_within_package`.

Usage

```
label.feature(x)
```

Arguments

x	variable names

Value

a character, e.g. "date", "Time Trend", etc.

`labels_within_package` *labels_within_package: Some labels package auther defined to make his plot, mainly serve the paper publication.*

Description

It contains a list that match each feature to its labels. It is used in the function `label.feature`.

Usage

```
labels_within_package
```

Format

An object of class `list` of length 20.

Details

labels_within_package <- list(dayint = "Time trend", diffcwv = "delta CWV (cm)", date = "", Column_WV = "MAIAC CWV (cm)", AOT_Uncertainty = "Blue band uncertainty", elev = "Elevation (m)", aod = "Aerosol optical depth", RelAZ = "Relative azimuth angle", DevAll_P1km = expression(paste("Proportion developed area in 1",km^2)), dist_water_km = "Distance to water (km)", forestProp_1km = expression(paste("Proportion of forest in 1",km^2)), Aer_optical_depth = "DSCOVR EPIC MAIAC AOD400nm", aer_aod440 = "AERONET AOD440nm", aer_aod500 = "AERONET AOD500nm", diff440 = "DSCOVR MAIAC - AERONET AOD", diff440_pred = "Predicted Error", aer_aod440_hat = "Predicted AERONET AOD440nm", AOD_470nm = "AERONET AOD470nm", Optical_Depth_047_t = "MAIAC AOD470nm (Terra)", Optical_Depth_047_a = "MAIAC AOD470nm (Aqua)")

References

http://doi.org/10.5281/zenodo.3334713

`new_labels` *new_labels: a place holder default to NULL.*

Description

if supplied as a list, it offers user to rename labels

Usage

```
new_labels
```

Format

An object of class `NULL` of length 0.

`plot.label` *internal-function to revise axis label for each feature*

Description

This function further fine-tune the format of each feature

Usage

```
## S3 method for class 'label'
plot(plot1, show_feature)
```

Arguments

`plot1`	ggplot2 object
`show_feature`	feature to plot

Value

returns ggplot2 object with further mordified layers based on the feature

`scatter.plot.diagonal` *make customized scatter plot with diagonal line and R2 printed.*

Description

make customized scatter plot with diagonal line and R2 printed.

Usage

```
scatter.plot.diagonal(
  data,
  x,
  y,
  size0 = 0.2,
  alpha0 = 0.3,
  dilute = FALSE,
  add_abline = FALSE,
  add_hist = TRUE,
  add_stat_cor = TRUE
)
```

Arguments

`data`	dataset
`x`	x
`y`	y
`size0`	point size, default to 1 of nobs<1000, 0.4 if nobs>1000
`alpha0`	alpha of point
`dilute`	a number or logical, dafault to TRUE, will plot `nrow(data_long)/dilute` data. For example, if dilute = 5 will plot 1/5 of the data. if dilute = TRUE will plot half of the data.
`add_abline`	default to FALSE, add a diagonal line `ggExtra::ggMarginal` but notice if add histogram, what is returned is no longer a ggplot2 object
`add_hist`	optional to add marginal histogram using `ggExtra::ggMarginal` but notice if add histogram, what is returned is no longer a ggplot2 object
`add_stat_cor`	add correlation and p-value from `ggpubr::stat_cor`

Value

ggplot2 object if `add_hist = FALSE`

Examples

```
scatter.plot.diagonal(data = iris, x = "Sepal.Length", y = "Petal.Length")
```

`scatter.plot.simple` *simple scatter plot, adding marginal histogram by default.*

Description

simple scatter plot, adding marginal histogram by default.

Usage

```
scatter.plot.simple(
  data,
  x,
  y,
  size0 = 0.2,
  alpha0 = 0.3,
  dilute = FALSE,
  add_hist = TRUE,
  add_stat_cor = FALSE
)
```

Arguments

`data`	dataset
`x`	x
`y`	y
`size0`	point size, default to 1 of nobs<1000, 0.4 if nobs>1000
`alpha0`	alpha of point
`dilute`	a number or logical, dafault to TRUE, will plot `nrow(data_long)/dilute` data. For example, if dilute = 5 will plot 1/5 of the data. if dilute = TRUE will plot half of the data.
`add_hist`	optional to add marginal histogram using `ggExtra::ggMarginal` but notice if add histogram, what is returned is no longer a ggplot2 object
`add_stat_cor`	add correlation and p-value from `ggpubr::stat_cor`

Value

ggplot2 object if `add_hist = FALSE`

Examples

```
scatter.plot.simple(data = iris, x = "Sepal.Length", y = "Petal.Length")
```

`shap.plot.dependence` *SHAP dependence plot and interaction plot, optional to be colored by a selected feature*

Description

This function by default makes a simple dependence plot with feature values on the x-axis and SHAP values on the y-axis, optional to color by another feature. It is optional to use a different variable for SHAP values on the y-axis, and color the points by the feature value of a designated variable. Not colored if `color_feature` is not supplied. If `data_int` (the SHAP interaction values dataset) is supplied, it will plot the interaction effect between `y` and `x` on the y-axis. Dependence plot is easy to make if you have the SHAP values dataset from `predict.xgb.Booster`. It is not necessary to start with the long format data, but since that is used for the summary plot, we just continue to use it here.

Usage

```
shap.plot.dependence(
  data_long,
  x,
  y = NULL,
  color_feature = NULL,
  data_int = NULL,
  dilute = FALSE,
  smooth = TRUE,
  size0 = NULL,
  add_hist = FALSE,
  add_stat_cor = FALSE
)
```

Arguments

`data_long`	the long format SHAP values from `shap.prep`
`x`	which feature to show on x-axis, it will plot the feature value
`y`	which shap values to show on y-axis, it will plot the SHAP value of that feature. y is default to x, if y is not provided, just plot the SHAP values of x on the y-axis
`color_feature`	which feature value to use for coloring, color by the feature value
`data_int`	the 3-dimention SHAP interaction values array. if `data_int` is supplied, y-axis will plot the interaction values of y (vs. x). `data_int` is obtained from either `predict.xgb.Booster` or `shap.prep.interaction`
`dilute`	a number or logical, dafault to TRUE, will plot `nrow(data_long)/dilute` data. For example, if dilute = 5 will plot 20% of the data. As long as dilute != FALSE, will plot at most half the data
`smooth`	optional to add a *loess* smooth line, default to TRUE.
`size0`	point size, default to 1 of nobs<1000, 0.4 if nobs>1000

add_hist	whether to add histogram using ggMarginal, default to TRUE. But notice the plot after adding histogram is a ggExtraPlot object instead of ggplot2 so cannot add geom to that anymore. Turn the histogram off if you wish to add more ggplot2 geoms
add_stat_cor	add correlation and p-value from ggpubr::stat_cor

Value

be default a ggplot2 object, based on which you could add more geom layers.

Examples

```
# **SHAP dependence plot**

# 1. simple dependence plot with SHAP values of x on the y axis
shap.plot.dependence(data_long = shap_long_iris, x="Petal.Length",
                     add_hist = TRUE, add_stat_cor = TRUE)

# 2. can choose a different SHAP values on the y axis
shap.plot.dependence(data_long = shap_long_iris, x="Petal.Length",
                           y = "Petal.Width")

# 3. color by another feature's feature values
shap.plot.dependence(data_long = shap_long_iris, x="Petal.Length",
                           color_feature = "Petal.Width")

# 4. choose 3 different variables for x, y, and color
shap.plot.dependence(data_long = shap_long_iris, x="Petal.Length",
                           y = "Petal.Width", color_feature = "Petal.Width")

# Optional to add hist or remove smooth line, optional to plot fewer data (make plot quicker)
shap.plot.dependence(data_long = shap_long_iris, x="Petal.Length",
                     y = "Petal.Width", color_feature = "Petal.Width",
                     add_hist = TRUE, smooth = FALSE, dilute = 3)

# to make a list of plot
plot_list <- lapply(names(iris)[2:3], shap.plot.dependence, data_long = shap_long_iris)

# **SHAP interaction effect plot **

# To get the interaction SHAP dataset for plotting, need to get `shap_int` first:
mod1 = xgboost::xgboost(
  data = as.matrix(iris[,-5]), label = iris$Species,
  gamma = 0, eta = 1, lambda = 0,nrounds = 1, verbose = FALSE)
# Use either:
data_int <- shap.prep.interaction(xgb_mod = mod1,
                                  X_train = as.matrix(iris[,-5]))
# or:
shap_int <- predict(mod1, as.matrix(iris[,-5]),
                    predinteraction = TRUE)

# if data_int is supplied, y axis will plot the interaction values of y (vs. x)
```

```
shap.plot.dependence(data_long = shap_long_iris,
                           data_int = shap_int_iris,
                           x="Petal.Length",
                           y = "Petal.Width",
                           color_feature = "Petal.Width")
```

`shap.plot.force_plot` *make the SHAP force plot*

Description

The force/stack plot, optional to zoom in at certain x-axis location or zoom in a specific cluster of observations.

Usage

```
shap.plot.force_plot(
  shapobs,
  id = "sorted_id",
  zoom_in_location = NULL,
  y_parent_limit = NULL,
  y_zoomin_limit = NULL,
  zoom_in = TRUE,
  zoom_in_group = NULL
)
```

Arguments

`shapobs`	The dataset obtained by `shap.prep.stack.data`.
`id`	the id variable.
`zoom_in_location`	where to zoom in, default at place of 60 percent of the data.
`y_parent_limit`	set y-axis limits.
`y_zoomin_limit`	`c(a,b)` to limit the y-axis in zoom-in.
`zoom_in`	default to TRUE, zoom in by `ggforce::facet_zoom`.
`zoom_in_group`	optional to zoom in certain cluster.

Examples

```
# **SHAP force plot**
plot_data <- shap.prep.stack.data(shap_contrib = shap_values_iris,
                                  n_groups = 4)
shap.plot.force_plot(plot_data)
shap.plot.force_plot(plot_data,  zoom_in_group = 2)

# plot all the clusters:
shap.plot.force_plot_bygroup(plot_data)
```

shap.plot.force_plot_bygroup

make the stack plot, optional to zoom in at certain x or certain cluster

Description

A collective display of zoom-in plots: one plot for every group of the clustered observations.

Usage

```
shap.plot.force_plot_bygroup(shapobs, id = "sorted_id", y_parent_limit = NULL)
```

Arguments

shapobs	The dataset obtained by shap.prep.stack.data.
id	the id variable.
y_parent_limit	set y-axis limits.

Examples

```
# **SHAP force plot**
plot_data <- shap.prep.stack.data(shap_contrib = shap_values_iris,
                                  n_groups = 4)
shap.plot.force_plot(plot_data)
shap.plot.force_plot(plot_data,  zoom_in_group = 2)

# plot all the clusters:
shap.plot.force_plot_bygroup(plot_data)
```

shap.plot.summary

SHAP summary plot core function using the long format SHAP values

Description

The summary plot (a sina plot) uses a long format data of SHAP values. The SHAP values could be obtained from either an xgboost model or a SHAP value matrix using shap.values. So this summary plot function normally follows the long format dataset obtained using shap.values. If you want to start with a xgboost model and data_X, use shap.plot.summary.wrap1. If you want to use a self-derived dataset of SHAP values, use shap.plot.summary.wrap2. If a list named **new_labels** is provided in the environment (new_labels is pre-loaded by the package as NULL), the plots will use that list to label the variables, here is an example of such a list (the default labels): labels_within_package.

Usage

```
shap.plot.summary(
  data_long,
  x_bound = NULL,
  dilute = FALSE,
  scientific = FALSE,
  my_format = NULL
)
```

Arguments

data_long	a long format data of SHAP values from shap.prep
x_bound	to set horizontal axis limit in the plot
dilute	a number or logical, aim to make the test plot for large amount of data faster. If dilute = 5 will plot 1/5 of the data. If dilute = TRUE or a number, will plot at most half points per feature, so the plotting won't be too slow. If you put dilute too high, at least 10 points per feature would be kept. If the dataset is too small after dilution, will just plot all the data
scientific	show the mean|SHAP| in scientific format. If TRUE, label format is 0.0E-0, default to FALSE, whose format is 0.000
my_format	supply your own number format if you really want

Value

returns a ggplot2 object, could add further layers.

Examples

```
data("iris")
X1 = as.matrix(iris[,-5])
mod1 = xgboost::xgboost(
  data = X1, label = iris$Species, gamma = 0, eta = 1,
  lambda = 0, nrounds = 1, verbose = FALSE)

# shap.values(model, X_dataset) returns the SHAP
# data matrix and ranked features by mean|SHAP|
shap_values <- shap.values(xgb_model = mod1, X_train = X1)
shap_values$mean_shap_score
shap_values_iris <- shap_values$shap_score

# shap.prep() returns the long-format SHAP data from either model or
shap_long_iris <- shap.prep(xgb_model = mod1, X_train = X1)
# is the same as: using given shap_contrib
shap_long_iris <- shap.prep(shap_contrib = shap_values_iris, X_train = X1)

# **SHAP summary plot**
shap.plot.summary(shap_long_iris, scientific = TRUE)
shap.plot.summary(shap_long_iris, x_bound  = 1.5, dilute = 10)
```

```
# Alternatives options to make the same plot:
# option 1: from the xgboost model
shap.plot.summary.wrap1(mod1, X = as.matrix(iris[,-5]), top_n = 3)

# option 2: supply a self-made SHAP values dataset
# (e.g. sometimes as output from cross-validation)
shap.plot.summary.wrap2(shap_score = shap_values_iris, X = X1, top_n = 3)
```

`shap.plot.summary.wrap1` *A wrapped function to make summary plot from xgb model object and predictors*

Description

wraps up function `shap.prep` and `shap.plot.summary`

Usage

```
shap.plot.summary.wrap1(model, X, top_n, dilute = FALSE)
```

Arguments

`model`	the xgboost model
`X`	the dataset of predictors used for the xgboost model
`top_n`	how many predictors you want to show in the plot (ranked)
`dilute`	a number or logical, aim to make the test plot for large amount of data faster. If dilute = 5 will plot 1/5 of the data. If dilute = TRUE or a number, will plot at most half points per feature, so the plotting won't be too slow. If you put dilute too high, at least 10 points per feature would be kept. If the dataset is too small after dilution, will just plot all the data

Examples

```
data("iris")
X1 = as.matrix(iris[,-5])
mod1 = xgboost::xgboost(
  data = X1, label = iris$Species, gamma = 0, eta = 1,
  lambda = 0, nrounds = 1, verbose = FALSE)

# shap.values(model, X_dataset) returns the SHAP
# data matrix and ranked features by mean|SHAP|
shap_values <- shap.values(xgb_model = mod1, X_train = X1)
shap_values$mean_shap_score
shap_values_iris <- shap_values$shap_score

# shap.prep() returns the long-format SHAP data from either model or
shap_long_iris <- shap.prep(xgb_model = mod1, X_train = X1)
```

```
# is the same as: using given shap_contrib
shap_long_iris <- shap.prep(shap_contrib = shap_values_iris, X_train = X1)

# **SHAP summary plot**
shap.plot.summary(shap_long_iris, scientific = TRUE)
shap.plot.summary(shap_long_iris, x_bound  = 1.5, dilute = 10)

# Alternatives options to make the same plot:
# option 1: from the xgboost model
shap.plot.summary.wrap1(mod1, X = as.matrix(iris[,-5]), top_n = 3)

# option 2: supply a self-made SHAP values dataset
# (e.g. sometimes as output from cross-validation)
shap.plot.summary.wrap2(shap_score = shap_values_iris, X = X1, top_n = 3)
```

`shap.plot.summary.wrap2` *A wrapped function to make summary plot from given SHAP values matrix*

Description

Sometimes the SHAP matrix is returned from cross-validation. This function wraps up function `shap.prep` and `shap.plot.summary`.

Usage

```
shap.plot.summary.wrap2(shap_score, X, top_n, dilute = FALSE)
```

Arguments

`shap_score`	the SHAP values dataset, could be obtained by `shap.prep`
`X`	the dataset of predictors used for the xgboost model
`top_n`	how many predictors you want to show in the plot (ranked)
`dilute`	a number or logical, aim to make the test plot for large amount of data faster. If dilute = 5 will plot 1/5 of the data. If dilute = TRUE or a number, will plot at most half points per feature, so the plotting won't be too slow. If you put dilute too high, at least 10 points per feature would be kept. If the dataset is too small after dilution, will just plot all the data

Examples

```
data("iris")
X1 = as.matrix(iris[,-5])
mod1 = xgboost::xgboost(
  data = X1, label = iris$Species, gamma = 0, eta = 1,
  lambda = 0, nrounds = 1, verbose = FALSE)
```

```
# shap.values(model, X_dataset) returns the SHAP
# data matrix and ranked features by mean|SHAP|
shap_values <- shap.values(xgb_model = mod1, X_train = X1)
shap_values$mean_shap_score
shap_values_iris <- shap_values$shap_score

# shap.prep() returns the long-format SHAP data from either model or
shap_long_iris <- shap.prep(xgb_model = mod1, X_train = X1)
# is the same as: using given shap_contrib
shap_long_iris <- shap.prep(shap_contrib = shap_values_iris, X_train = X1)

# **SHAP summary plot**
shap.plot.summary(shap_long_iris, scientific = TRUE)
shap.plot.summary(shap_long_iris, x_bound  = 1.5, dilute = 10)

# Alternatives options to make the same plot:
# option 1: from the xgboost model
shap.plot.summary.wrap1(mod1, X = as.matrix(iris[,-5]), top_n = 3)

# option 2: supply a self-made SHAP values dataset
# (e.g. sometimes as output from cross-validation)
shap.plot.summary.wrap2(shap_score = shap_values_iris, X = X1, top_n = 3)
```

---

shap.prep *prep SHAP values into long format for plotting*

Description

Produce a dataset of 6 columns: ID of each observation, variable name, SHAP value, variable values (feature value), deviation of the feature value for each observation (for coloring the point), and the mean SHAP values for each variable. You can view this example dataset included in the package: shap_long_iris

Usage

```
shap.prep(
  xgb_model = NULL,
  shap_contrib = NULL,
  X_train,
  top_n = NULL,
  var_cat = NULL
)
```

Arguments

xgb_model	a xgboost model object, will derive the SHAP values from it
shap_contrib	optional to directly supply a SHAP values dataset. If supplied, it will overwrite the xgb_model if xgb_model is also supplied

X_train	the dataset of predictors used for the xgboost model, it provides feature values to the plot, must be supplied
top_n	to choose top_n variables ranked by mean|SHAP| if needed
var_cat	if supplied, will provide long format data, grouped by this categorical variable

Details

The ID variable is added for each observation in the shap_contrib dataset for better tracking, it is created as 1:nrow(shap_contrib) before melting shap_contrib into long format.

Value

a long format data.table, named as shap_long

Examples

```
data("iris")
X1 = as.matrix(iris[,-5])
mod1 = xgboost::xgboost(
  data = X1, label = iris$Species, gamma = 0, eta = 1,
  lambda = 0, nrounds = 1, verbose = FALSE)

# shap.values(model, X_dataset) returns the SHAP
# data matrix and ranked features by mean|SHAP|
shap_values <- shap.values(xgb_model = mod1, X_train = X1)
shap_values$mean_shap_score
shap_values_iris <- shap_values$shap_score

# shap.prep() returns the long-format SHAP data from either model or
shap_long_iris <- shap.prep(xgb_model = mod1, X_train = X1)
# is the same as: using given shap_contrib
shap_long_iris <- shap.prep(shap_contrib = shap_values_iris, X_train = X1)

# **SHAP summary plot**
shap.plot.summary(shap_long_iris, scientific = TRUE)
shap.plot.summary(shap_long_iris, x_bound  = 1.5, dilute = 10)

# Alternatives options to make the same plot:
# option 1: from the xgboost model
shap.plot.summary.wrap1(mod1, X = as.matrix(iris[,-5]), top_n = 3)

# option 2: supply a self-made SHAP values dataset
# (e.g. sometimes as output from cross-validation)
shap.plot.summary.wrap2(shap_score = shap_values_iris, X = X1, top_n = 3)
####
#
# use `var_cat` to add a categorical variable, output the long-format data differently:
library("data.table")
data("iris")
set.seed(123)
iris$Group <- 0
iris[sample(1:nrow(iris), nrow(iris)/2), "Group"] <- 1
```

```
data.table::setDT(iris)
X_train = as.matrix(iris[,c(colnames(iris)[1:4], "Group"), with = FALSE])
mod1 = xgboost::xgboost(
  data = X_train, label = iris$Species, gamma = 0, eta = 1,
  lambda = 0, nrounds = 1, verbose = FALSE)

shap_long2 <- shap.prep(xgb_model = mod1, X_train = X_train, var_cat = "Group")
# **SHAP summary plot**
shap.plot.summary(shap_long2, scientific = TRUE) +
  ggplot2::facet_wrap(~ Group)
```

`shap.prep.interaction` *prepare the interaction SHAP values from predict.xgb.Booster*

Description

This function just runs `shap_int <-predict(xgb_mod,as.matrix(X_train),predinteraction = TRUE)`, thus it may not be necessary. Read more about the xgboost predict function at `xgboost::predict.xgb.Booster`.

Usage

```
shap.prep.interaction(xgb_model, X_train)
```

Arguments

`xgb_model`	a xgboost model object
`X_train`	the dataset of predictors used for the xgboost model

Value

a 3-dimention array: #obs x #features x #features

Examples

```
# To get the interaction SHAP dataset for plotting:
# fit the xgboost model
mod1 = xgboost::xgboost(
  data = as.matrix(iris[,-5]), label = iris$Species,
  gamma = 0, eta = 1, lambda = 0,nrounds = 1, verbose = FALSE)
# Use either:
data_int <- shap.prep.interaction(xgb_mod = mod1,
                                  X_train = as.matrix(iris[,-5]))
# or:
shap_int <- predict(mod1, as.matrix(iris[,-5]),
                    predinteraction = TRUE)

# **SHAP interaction effect plot **
shap.plot.dependence(data_long = shap_long_iris,
```

```
                        data_int = shap_int_iris,
                        x="Petal.Length",
                        y = "Petal.Width",
                        color_feature = "Petal.Width")
```

---

shap.prep.stack.data *Prepare data for SHAP force plot (stack plot)*

Description

Make force plot for top_n features, optional to randomly plot certain portion of the data in case the dataset is large.

Usage

```
shap.prep.stack.data(
  shap_contrib,
  top_n = NULL,
  data_percent = 1,
  cluster_method = "ward.D",
  n_groups = 10L
)
```

Arguments

shap_contrib	shap_contrib is the SHAP value data returned from predict.xgb.booster, here an ID variable is added for each observation in the shap_contrib dataset for better tracking, it is created in the begining as 1:nrow(shap_contrib). The ID matches the output from shap.prep
top_n	integer, optional to show only top_n features, combine the rest
data_percent	what percent of data to plot (to speed up the testing plot). The accepted input range is (0,1], if observations left is too few, there will be an error from the clustering function
cluster_method	default to ward.D, please refer to stats::hclust for details
n_groups	a integer, how many groups to plot in shap.plot.force_plot_bygroup

Value

a dataset for stack plot

Examples

```
# **SHAP force plot**
plot_data <- shap.prep.stack.data(shap_contrib = shap_values_iris,
                                  n_groups = 4)
shap.plot.force_plot(plot_data)
```

```
shap.plot.force_plot(plot_data,  zoom_in_group = 2)

# plot all the clusters:
shap.plot.force_plot_bygroup(plot_data)
```

---

shap.values *return SHAP contribution from xgboost model*

Description

The shap.values returns a list of three objects from xgboost model: 1. a dataset (data.table) of SHAP scores. It has the same dimension as the X_train); 2. the ranked variable vector by each variable's mean absolute SHAP value, it ranks the predictors by their importance in the model; and 3. The BIAS, which is like an intercept. The rowsum of SHAP values including the BIAS would equal to the predicted value (y_hat).

Usage

```
shap.values(xgb_model, X_train)
```

Arguments

xgb_model	a xgboost model object
X_train	the dataset of predictors (independent variables) used for the xgboost model

Value

a list of three elements: the SHAP values as data.table, ranked mean|SHAP|, and BIAS

Examples

```
data("iris")
X1 = as.matrix(iris[,-5])
mod1 = xgboost::xgboost(
  data = X1, label = iris$Species, gamma = 0, eta = 1,
  lambda = 0, nrounds = 1, verbose = FALSE)

# shap.values(model, X_dataset) returns the SHAP
# data matrix and ranked features by mean|SHAP|
shap_values <- shap.values(xgb_model = mod1, X_train = X1)
shap_values$mean_shap_score
shap_values_iris <- shap_values$shap_score

# shap.prep() returns the long-format SHAP data from either model or
shap_long_iris <- shap.prep(xgb_model = mod1, X_train = X1)
# is the same as: using given shap_contrib
shap_long_iris <- shap.prep(shap_contrib = shap_values_iris, X_train = X1)

# **SHAP summary plot**
```

```
shap.plot.summary(shap_long_iris, scientific = TRUE)
shap.plot.summary(shap_long_iris, x_bound  = 1.5, dilute = 10)

# Alternatives options to make the same plot:
# option 1: from the xgboost model
shap.plot.summary.wrap1(mod1, X = as.matrix(iris[,-5]), top_n = 3)

# option 2: supply a self-made SHAP values dataset
# (e.g. sometimes as output from cross-validation)
shap.plot.summary.wrap2(shap_score = shap_values_iris, X = X1, top_n = 3)
```

---

shap_int_iris *The interaction effect SHAP values example using iris dataset.*

Description

The interaction effect SHAP values example using iris dataset.

Usage

```
shap_int_iris
```

Format

An object of class `array` of dimension 150 x 5 x 5.

---

shap_long_iris *The long-format SHAP values example using iris dataset.*

Description

The long-format SHAP values example using iris dataset.

Usage

```
shap_long_iris
```

Format

An object of class `data.table` (inherits from `data.frame`) with 600 rows and 6 columns.

shap_score *SHAP values example from dataXY_df .*

Description

SHAP values example from dataXY_df .

Usage

```
shap_score
```

Format

An object of class `data.table` (inherits from `data.frame`) with 10148 rows and 9 columns.

References

http://doi.org/10.5281/zenodo.3334713

shap_values_iris *SHAP values example using iris dataset.*

Description

SHAP values example using iris dataset.

Usage

```
shap_values_iris
```

Format

An object of class `data.table` (inherits from `data.frame`) with 150 rows and 4 columns.

Index

∗Topic **Labels**
    labels_within_package, 3
    new_labels, 4
∗Topic **Terra**
    dataXY_df, 2
    shap_score, 20
∗Topic **iris**
    shap_int_iris, 19
    shap_long_iris, 19
    shap_values_iris, 20

dataXY_df, 2

label.feature, *3*, 3
labels_within_package, *3*, 3, *10*

new_labels, 4

plot.label, 4

scatter.plot.diagonal, 5
scatter.plot.simple, 6
shap.plot.dependence, 7
shap.plot.force_plot, 9
shap.plot.force_plot_bygroup, 10, *17*
shap.plot.summary, 10, *12, 13*
shap.plot.summary.wrap1, *10*, 12
shap.plot.summary.wrap2, *10*, 13
shap.prep, *7*, *11–13*, 14, *17*
shap.prep.interaction, *7*, 16
shap.prep.stack.data, 17
shap.values, *10*, 18
shap_int_iris, 19
shap_long_iris, *14*, 19
shap_score, 20
shap_values_iris, 20