Package 'SHAPforxgboost'

May 7, 2026

Title SHAP Plots for 'XGBoost'

Version 0.2.0

Date 2025-12-23

Description Aid in visual data investigations
using SHAP (SHapley Additive exPlanation) visualization plots for 'XGBoost' and 'LightGBM'.
It provides summary plot, dependence plot, interaction plot, and force plot and relies on
the SHAP implementation provided by 'XGBoost' and 'LightGBM'.

License MIT + file LICENSE

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

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

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)

VignetteBuilder knitr

Imports stats, utils, 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 knitr, rmarkdown, gridExtra (>= 2.3), here, parallel,
lightgbm (>= 2.1)

RoxygenNote 7.3.2

NeedsCompilation no

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

Maintainer Yang Liu <lyhello@gmail.com>

Repository CRAN

Date/Publication 2025-12-13 17:40:02 UTC

Contents

`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

doi:10.5281/zenodo.3568449

`label.feature` *Modify labels for features under plotting*

Description

`label.feature` helps to modify labels. 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

doi:10.5281/zenodo.3568449

`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.importance *Variable importance as measured by mean absolute SHAP value.*

Description

Variable importance as measured by mean absolute SHAP value.

Usage

```
shap.importance(data_long, names_only = FALSE, top_n = Inf)
```

Arguments

data_long	a long format data of SHAP values from shap.prep
names_only	If TRUE, returns variable names only.
top_n	How many variables to be returned?

Value

returns data.table with average absolute SHAP values per variable, sorted in decreasing order of importance.

Examples

```
shap.importance(shap_long_iris)

shap.importance(shap_long_iris, names_only = 1)
```

shap.plot.dependence *SHAP dependence and interaction plots*

Description

Creates scatter plots showing the relationship between feature values (x-axis) and SHAP values (y-axis). Can display:

- Simple dependence: how feature values affect predictions
- Colored by another feature: to explore interactions
- Interaction effects: when data_int is provided, shows pairwise SHAP interaction values

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,
  alpha = NULL,
  jitter_height = 0,
  jitter_width = 0,
  ...
)
```

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. If "auto", will select the feature "c" minimizing the variance of the shap value given x and c, which can be viewed as a heuristic for the strongest interaction.
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 if 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
alpha	point transparancy, default to 1 if nobs<1000 else 0.6
jitter_height	amount of vertical jitter (see hight in geom_jitter)
jitter_width	amount of horizontal jitter (see width in geom_jitter). Use values close to 0, e.g. 0.02
...	additional parameters passed to geom_jitter

Value

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

Examples

```
# Example: SHAP dependence plots

# 1. Simple dependence plot: SHAP values vs feature values
shap.plot.dependence(data_long = shap_long_iris, x="Petal.Length",
                     add_hist = TRUE, add_stat_cor = TRUE)

# 2. Show different SHAP values on y-axis
shap.plot.dependence(data_long = shap_long_iris, x="Petal.Length",
                          y = "Petal.Width")

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

# 4. Customize x, y, and color features
shap.plot.dependence(data_long = shap_long_iris, x="Petal.Length",
                          y = "Petal.Width", color_feature = "Petal.Width")

# 5. Additional options: histogram, smooth line, data dilution
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)

# Create multiple plots at once
plot_list <- lapply(names(iris)[2:3], shap.plot.dependence, data_long = shap_long_iris)

# SHAP interaction effect plot
# First, prepare the model and interaction data
X_iris = as.matrix(iris[,1:4])
y_iris = as.numeric(iris[[5]]) - 1
dtrain = xgboost::xgb.DMatrix(data = X_iris, label = y_iris)
params = list(learning_rate = 1, min_split_loss = 0, reg_lambda = 0,
              objective = 'reg:squarederror', nthread = 1)
mod1 = xgboost::xgb.train(params = params, data = dtrain,
                          nrounds = 1, verbose = 0)

# Get interaction SHAP values (two methods):
data_int <- shap.prep.interaction(xgb_model = mod1, X_train = X_iris)
# Or directly:
shap_int <- predict(mod1, X_iris, predinteraction = TRUE)

# Plot interaction effects (y-axis shows interaction values)
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 *Create SHAP force plot (stacked bar chart)*

Description

Displays feature contributions as stacked bars for individual predictions. Each bar shows how features push the prediction above or below the baseline. Supports optional zoom-in for detailed inspection of observation clusters.

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

```
# Example: SHAP force plots (stacked bar charts)
# Shows contribution of each feature to individual predictions

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 clusters separately
shap.plot.force_plot_bygroup(plot_data)
```

shap.plot.force_plot_bygroup *Create faceted SHAP force plots by cluster*

Description

Creates a faceted display with one force plot per observation cluster, allowing comparison of prediction patterns across different groups.

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

```
# Example: SHAP force plots (stacked bar charts)
# Shows contribution of each feature to individual predictions

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 clusters separately
shap.plot.force_plot_bygroup(plot_data)
```

shap.plot.summary *SHAP summary plot using long-format SHAP values*

Description

Creates a beeswarm/sina plot or bar chart showing feature importance. The sina plot shows SHAP value distributions for each feature, colored by feature values. For a simpler workflow, use shap.plot.summary.wrap1 (directly from model) or shap.plot.summary.wrap2 (from SHAP matrix).

Usage

```
shap.plot.summary(
  data_long,
  x_bound = NULL,
  dilute = FALSE,
  scientific = FALSE,
  my_format = NULL,
  min_color_bound = "#FFCC33",
  max_color_bound = "#6600CC",
  kind = c("sina", "bar")
)
```

Arguments

data_long	a long format data of SHAP values from shap.prep
x_bound	use to set horizontal axis limit in the plot
dilute	being numeric or logical (TRUE/FALSE), it aims to help 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, and the format will be 0.000
my_format	supply your own number format if you really want
min_color_bound	min color hex code for colormap. Color gradient is scaled between min_color_bound and max_color_bound. Default is "#FFCC33".
max_color_bound	max color hex code for colormap. Color gradient is scaled between min_color_bound and max_color_bound. Default is "#6600CC".
kind	By default, a "sina" plot is shown. As an alternative, set kind = "bar" to visualize mean absolute SHAP values as a barplot. Its color is controlled by max_color_bound. Other arguments are ignored for this kind of plot.

Details

To customize feature labels, define new_labels in the global environment as a named list (see labels_within_package for examples).

Value

returns a ggplot2 object, could add further layers.

Examples

```
# Example: Basic workflow for SHAP summary plot
# Note: For xgboost 3.x, use xgb.DMatrix + xgb.train, and convert factor labels to numeric
```

```
data("iris")
X1 = as.matrix(iris[,1:4])
y1 = as.numeric(iris[[5]]) - 1  # Convert factor to numeric
dtrain = xgboost::xgb.DMatrix(data = X1, label = y1)
params = list(learning_rate = 1, min_split_loss = 0, reg_lambda = 0,
              objective = 'reg:squarederror', nthread = 1)
mod1 = xgboost::xgb.train(params = params, data = dtrain,
                          nrounds = 1, verbose = 0)

# Get SHAP values and feature importance
shap_values <- shap.values(xgb_model = mod1, X_train = X1)
shap_values$mean_shap_score  # Ranked features by mean|SHAP|
shap_values_iris <- shap_values$shap_score

# Prepare long-format data for plotting
shap_long_iris <- shap.prep(xgb_model = mod1, X_train = X1)
# Alternative: use pre-computed SHAP values
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)

# Alternative options:
# Option 1: directly from xgboost model
shap.plot.summary.wrap1(mod1, X = as.matrix(iris[,1:4]), top_n = 3)

# Option 2: from pre-computed SHAP values (useful for 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 model object and predictors*

Description

`shap.plot.summary.wrap1` wraps up function `shap.prep` and `shap.plot.summary`

Usage

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

Arguments

`model`	the model
`X`	the dataset of predictors used for calculating SHAP
`top_n`	how many predictors you want to show in the plot (ranked)

dilute — being numeric or logical (TRUE/FALSE), it aims to help 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

```
# Example: Basic workflow for SHAP summary plot
# Note: For xgboost 3.x, use xgb.DMatrix + xgb.train, and convert factor labels to numeric

data("iris")
X1 = as.matrix(iris[,1:4])
y1 = as.numeric(iris[[5]]) - 1  # Convert factor to numeric
dtrain = xgboost::xgb.DMatrix(data = X1, label = y1)
params = list(learning_rate = 1, min_split_loss = 0, reg_lambda = 0,
              objective = 'reg:squarederror', nthread = 1)
mod1 = xgboost::xgb.train(params = params, data = dtrain,
                          nrounds = 1, verbose = 0)

# Get SHAP values and feature importance
shap_values <- shap.values(xgb_model = mod1, X_train = X1)
shap_values$mean_shap_score  # Ranked features by mean|SHAP|
shap_values_iris <- shap_values$shap_score

# Prepare long-format data for plotting
shap_long_iris <- shap.prep(xgb_model = mod1, X_train = X1)
# Alternative: use pre-computed SHAP values
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)

# Alternative options:
# Option 1: directly from xgboost model
shap.plot.summary.wrap1(mod1, X = as.matrix(iris[,1:4]), top_n = 3)

# Option 2: from pre-computed SHAP values (useful for 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

shap.plot.summary.wrap2 wraps up function shap.prep and shap.plot.summary. Since SHAP matrix could be returned from cross-validation instead of only one model, here the wrapped shap.prep takes the SHAP score matrix shap_score as input

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 calculating SHAP values
top_n	how many predictors you want to show in the plot (ranked)
dilute	being numeric or logical (TRUE/FALSE), it aims to help 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

```
# Example: Basic workflow for SHAP summary plot
# Note: For xgboost 3.x, use xgb.DMatrix + xgb.train, and convert factor labels to numeric

data("iris")
X1 = as.matrix(iris[,1:4])
y1 = as.numeric(iris[[5]]) - 1  # Convert factor to numeric
dtrain = xgboost::xgb.DMatrix(data = X1, label = y1)
params = list(learning_rate = 1, min_split_loss = 0, reg_lambda = 0,
              objective = 'reg:squarederror', nthread = 1)
mod1 = xgboost::xgb.train(params = params, data = dtrain,
                          nrounds = 1, verbose = 0)

# Get SHAP values and feature importance
shap_values <- shap.values(xgb_model = mod1, X_train = X1)
shap_values$mean_shap_score  # Ranked features by mean|SHAP|
shap_values_iris <- shap_values$shap_score

# Prepare long-format data for plotting
shap_long_iris <- shap.prep(xgb_model = mod1, X_train = X1)
# Alternative: use pre-computed SHAP values
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)

# Alternative options:
# Option 1: directly from xgboost model
shap.plot.summary.wrap1(mod1, X = as.matrix(iris[,1:4]), top_n = 3)

# Option 2: from pre-computed SHAP values (useful for cross-validation)
shap.plot.summary.wrap2(shap_score = shap_values_iris, X = X1, top_n = 3)
```

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

Description

Produces a data.table with 6 columns: ID (observation identifier), variable (feature name), value (SHAP value), rfvalue (raw feature value), stdfvalue (standardized feature value for coloring), and mean_value (mean absolute SHAP value per feature). See shap_long_iris for an example.

Usage

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

Arguments

xgb_model	an XGBoost or LightGBM model object (will derive SHAP values from it)
shap_contrib	optional: a matrix of SHAP values (without baseline column). If supplied, will use these values instead of computing from xgb_model
X_train	the predictor matrix used to calculate SHAP values (required)
top_n	number of top features to include, ranked by mean|SHAP| (default: all)
var_cat	optional: name of a categorical variable for grouped/faceted plots

Details

The ID variable (1:nrow(shap_contrib)) is added to track each observation before converting to long format.

Value

a long-format data.table, named as shap_long

Examples

```
# Example: Basic workflow for SHAP summary plot
# Note: For xgboost 3.x, use xgb.DMatrix + xgb.train, and convert factor labels to numeric

data("iris")
X1 = as.matrix(iris[,1:4])
y1 = as.numeric(iris[[5]]) - 1  # Convert factor to numeric
dtrain = xgboost::xgb.DMatrix(data = X1, label = y1)
params = list(learning_rate = 1, min_split_loss = 0, reg_lambda = 0,
```

```
              objective = 'reg:squarederror', nthread = 1)
mod1 = xgboost::xgb.train(params = params, data = dtrain,
                          nrounds = 1, verbose = 0)

# Get SHAP values and feature importance
shap_values <- shap.values(xgb_model = mod1, X_train = X1)
shap_values$mean_shap_score  # Ranked features by mean|SHAP|
shap_values_iris <- shap_values$shap_score

# Prepare long-format data for plotting
shap_long_iris <- shap.prep(xgb_model = mod1, X_train = X1)
# Alternative: use pre-computed SHAP values
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)

# Alternative options:
# Option 1: directly from xgboost model
shap.plot.summary.wrap1(mod1, X = as.matrix(iris[,1:4]), top_n = 3)

# Option 2: from pre-computed SHAP values (useful for cross-validation)
shap.plot.summary.wrap2(shap_score = shap_values_iris, X = X1, top_n = 3)
# Example: Using var_cat to group by a categorical variable
# The var_cat argument creates grouped long-format data for faceted plots

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])
y_train = as.numeric(iris$Species) - 1  # Convert factor to numeric
dtrain = xgboost::xgb.DMatrix(data = X_train, label = y_train)
params = list(learning_rate = 1, min_split_loss = 0, reg_lambda = 0,
              objective = 'reg:squarederror', nthread = 1)
mod1 = xgboost::xgb.train(params = params, data = dtrain,
                          nrounds = 1, verbose = 0)

# Use var_cat to create faceted plots by Group
shap_long2 <- shap.prep(xgb_model = mod1, X_train = X_train, var_cat = "Group")
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 is a convenience wrapper for `predict(xgb_model, X_train, predinteraction = TRUE)`. See `xgboost::predict.xgb.Booster` for details. Note: This functionality is only available for XGBoost models, not LightGBM.

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

```
# Example: SHAP interaction plots
# Shows how feature interactions affect predictions

X_iris = as.matrix(iris[,1:4])
y_iris = as.numeric(iris[[5]]) - 1  # Convert factor to numeric
dtrain = xgboost::xgb.DMatrix(data = X_iris, label = y_iris)
params = list(learning_rate = 1, min_split_loss = 0, reg_lambda = 0,
              objective = 'reg:squarederror', nthread = 1)
mod1 = xgboost::xgb.train(params = params, data = dtrain,
                          nrounds = 1, verbose = 0)

# Get interaction SHAP values (two methods):
data_int <- shap.prep.interaction(xgb_model = mod1, X_train = X_iris)
# Or directly:
shap_int <- predict(mod1, X_iris, predinteraction = TRUE)

# Plot interaction effects
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 (stacked bar chart)*

Description

Transforms SHAP values into a format suitable for force plots, which show how features contribute to individual predictions. The function:

- Ranks features by importance
- Optionally combines less important features into 'rest_variables'
- Clusters observations for better visualization
- Assigns group labels for faceted plots

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, 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

```
# Example: SHAP force plots (stacked bar charts)
# Shows contribution of each feature to individual predictions

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 clusters separately
shap.plot.force_plot_bygroup(plot_data)
```

---

shap.values *Get SHAP scores from a trained XGBoost or LightGBM model*

Description

`shap.values` returns a list of three objects from XGBoost or LightGBM 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 baseline value (intercept), which is stored in the last column of the SHAP contribution matrix (named "BIAS" in older xgboost versions or "(Intercept)" in newer versions). The rowsum of SHAP values including the baseline would equal to the predicted value (y_hat) generally speaking.

Usage

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

Arguments

`xgb_model`	an XGBoost or LightGBM model object
`X_train`	the data supplied to the `predict` function to get the prediction. It should be a matrix. Notice that coercing the matrix to a dense matrix by using `as.matrix` might lead to wrong behaviors in some cases. See discussion in issues on this topic.

Value

a list of three elements:

`shap_score`	A data.table of SHAP values (without the baseline column)
`mean_shap_score`	Ranked features by mean absolute SHAP value
`BIAS0`	The baseline/intercept value (from the '(Intercept)' column in xgboost 3.x)

Examples

```
# Example: Basic workflow for SHAP summary plot
# Note: For xgboost 3.x, use xgb.DMatrix + xgb.train, and convert factor labels to numeric

data("iris")
X1 = as.matrix(iris[,1:4])
y1 = as.numeric(iris[[5]]) - 1  # Convert factor to numeric
dtrain = xgboost::xgb.DMatrix(data = X1, label = y1)
params = list(learning_rate = 1, min_split_loss = 0, reg_lambda = 0,
              objective = 'reg:squarederror', nthread = 1)
mod1 = xgboost::xgb.train(params = params, data = dtrain,
                          nrounds = 1, verbose = 0)

# Get SHAP values and feature importance
shap_values <- shap.values(xgb_model = mod1, X_train = X1)
shap_values$mean_shap_score  # Ranked features by mean|SHAP|
shap_values_iris <- shap_values$shap_score

# Prepare long-format data for plotting
shap_long_iris <- shap.prep(xgb_model = mod1, X_train = X1)
# Alternative: use pre-computed SHAP values
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)

# Alternative options:
# Option 1: directly from xgboost model
shap.plot.summary.wrap1(mod1, X = as.matrix(iris[,1:4]), top_n = 3)

# Option 2: from pre-computed SHAP values (useful for 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

doi:10.5281/zenodo.3568449

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