Package ‘ceterisParibus’

February 24, 2020

Title Ceteris Paribus Profiles
Version 0.4.1
Description Ceteris Paribus Profiles (What-If Plots) are designed to present model responses around selected points in a feature space. For example around a single prediction for an interesting observation. Plots are designed to work in a model-agnostic fashion, they are working for any predictive Machine Learning model and allow for model comparisons. Ceteris Paribus Plots supplement the Break Down Plots from ‘breakDown’ package.

Depends R (>= 3.3), ggplot2, gower
Suggests randomForest, ggiraph, e1071, testthat, rpart
Imports DALEX, knitr
License GPL-2
Encoding UTF-8
LazyData true

URL https://pbiecek.github.io/ceterisParibus/
BugReports https://github.com/pbiecek/ceterisParibus/issues
RoxygenNote 7.0.2
NeedsCompilation no
Author Przemyslaw Biecek [aut, cre] (<https://orcid.org/0000-0001-8423-1823>)
Maintainer Przemyslaw Biecek <przemyslaw.biecek@gmail.com>
Repository CRAN
Date/Publication 2020-02-24 16:50:29 UTC

R topics documented:

+ .plot_ceteris_paribus_explainer ................................. 2
calculate_oscillations ........................................ 3
calculate_profiles ........................................... 3
calculate_profiles_lce ....................................... 5
calculate_variable_splits ................................... 6
**Description**

Add More Layers to a Ceteris Paribus Plot

**Usage**

```r
## S3 method for class 'plot_ceteris_paribus_explainer'
e1 + e2
```

**Arguments**

- `e1`: An object of class 'plot_ceteris_paribus_explainer'.
- `e2`: A plot component
**calculate_oscillations**

*Calculate Oscillations for Ceteris Paribus Explainer*

**Description**

Calculate Oscillations for Ceteris Paribus Explainer

**Usage**

`calculate_oscillations(x, sort = TRUE, ...)`

**Arguments**

- `x`: a ceteris_paribus explainer produced with the `ceteris_paribus()` function
- `sort`: a logical value. If TRUE then rows are sorted along the oscillations
- `...`: other arguments

**Examples**

```r
library("DALEX")
## Not run:
library("randomForest")
set.seed(59)

apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
                                    no.rooms + district, data = apartments)

explainer_rf <- explain(apartments_rf_model,
                         data = apartmentsTest, y = apartmentsTest$m2.price)
apartment <- apartmentsTest[1,]

apartment <- apartmentsTest[1,]
cp_rf <- ceteris_paribus(explainer_rf, apartment)
calculate_oscillations(cp_rf)
## End(Not run)
```

**calculate_profiles**

*Calculate Ceteris Paribus Profiles*

**Description**

This function calculates ceteris paribus profiles, i.e. series of predictions from a model calculated for observations with altered single coordinate.
calculate_profiles

Usage

```r
calculate_profiles(
  data,
  variable_splits,
  model,
  predict_function = predict,
  ...
)
```

Arguments

data  
set of observations. Profile will be calculated for every observation (every row)

variable_splits  
named list of vectors. Elements of the list are vectors with points in which
profiles should be calculated. See an example for more details.

model  
a model that will be passed to the predict_function

predict_function  
function that takes data and model and returns numeric predictions. Note that
the ... arguments will be passed to this function.

...  
other parameters that will be passed to the predict_function

Details

Note that calculate_profiles function is S3 generic. If you want to work on non standard data
sources (like H2O ddf, external databases) you should overload it.

Value

a data frame with profiles for selected variables and selected observations

Examples

```r
library("DALEX")

# Not run:
library("randomForest")
set.seed(59)
apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
                                       no.rooms + district, data = apartments)
vars <- c("construction.year", "surface", "floor", "no.rooms", "district")
variable_splits <- calculate_variable_splits(apartments, vars)
new_apartment <- apartmentsTest[1:10,]
profiles <- calculate_profiles(new_apartment, variable_splits,
                               apartments_rf_model)

# only subset of observations
small_apartments <- select_sample(apartmentsTest, n = 10)
small_apartments
small_profiles <- calculate_profiles(small_apartments, variable_splits,
```
calculate_profiles_lce

Calculate Local Conditional Expectation profiles

Description

This function Local Conditional Expectation profiles

Usage

calculate_profiles_lce(
  data,
  variable_splits,
  model,
  dataset,
  predict_function = predict,
  ...
)

Arguments

data set of observations. Profile will be calculated for every observation (every row)
variable_splits named list of vectors. Elements of the list are vectors with points in which profiles should be calculated. See an example for more details.
model a model that will be passed to the predict_function
dataset a data.frame, usually training data of a model, used for calculation of LCE profiles
predict_function function that takes data and model and returns numeric predictions. Note that the ... arguments will be passed to this function.
... other parameters that will be passed to the predict_function
Details

Note that calculate_profiles_lce function is S3 generic. If you want to work on non standard data sources (like H2O ddf, external databases) you should overload it.

Value

a data frame with profiles for selected variables and selected observations

Examples

library("DALEX")
## Not run:
library("randomForest")
set.seed(59)
apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor + no.rooms + district, data = apartments)
explainer_rf <- explain(apartments_rf_model,
    data = apartments[,2:6], y = apartments$m2.price)
vars <- c("construction.year", "surface", "floor", "no.rooms", "district")
variable_splits <- calculate_variable_splits(apartments, vars)
new_apartment <- apartments[1, ]
profiles <- calculate_profiles_lce(new_apartment, variable_splits,
    apartments_rf_model, explainer_rf$data)
profiles
## End(Not run)

calculate_variable_splits

Calculate Split Points for Selected Variables

Description

This function calculate candidate splits for each selected variable. For numerical variables splits are calculated as percentiles (in general uniform quantiles of the length grid_points). For all other variables splits are calculated as unique values.

Usage

calculate_variable_splits(data, variables = colnames(data), grid_points = 101)

Arguments

data validation dataset. Is used to determine distribution of observations.
variables names of variables for which splits shall be calculated
grid_points number of points used for response path
Details

Note that `calculate_variable_splits` function is S3 generic. If you want to work on non standard data sources (like H2O ddf, external databases) you should overload it.

Value

A named list with splits for selected variables

Examples

```r
library("DALEX")
## Not run:
library("randomForest")
set.seed(59)
apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
                                      no.rooms + district, data = apartments)
vars <- c("construction.year", "surface", "floor", "no.rooms", "district")
calculate_variable_splits(apartments, vars)
## End(Not run)
```

---

ceteris_paribus  
_Ceteris Paribus Explainer_

Description

This function calculate ceteris paribus profiles for selected data points.

Usage

```r
ceteris_paribus(
    explainer, observations, y = NULL,
    variable_splits = NULL, variables = NULL,
    grid_points = 101
)
```

Arguments

- `explainer` a model to be explained, preprocessed by function ‘DALEX::explain()’.
- `observations` set of observation for which profiles are to be calculated
- `y` true labels for `observations`. If specified then will be added to ceteris paribus plots.
ceteris_paribus_layer

add_layer to the Ceteris Paribus Plot

Description

Function `ceteris_paribus_layer()` adds a layer to a plot created with `plot.ceteris_paribus_explainer()` plots. Various parameters help to decide what should be plotted, profiles, aggregated profiles, points or rugs.

**variable_splits**

named list of splits for variables, in most cases created with `calculate_variable_splits()`. If NULL then it will be calculated based on validation data available in the `explainer`.

**variables**

names of variables for which profiles shall be calculated. Will be passed to `calculate_variable_splits()`. If NULL then all variables from the validation data will be used.

**grid_points**

number of points for profile. Will be passed to `calculate_variable_splits()`.

Value

An object of the class `ceteris_paribus_explainer`. It’s a data frame with calculated average responses.

Examples

```r
library("DALEX")
## Not run:
library("randomForest")
set.seed(59)

apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor + no.rooms + district, data = apartments)

explainer_rf <- explain(apartments_rf_model,
                         data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)

apartments_small <- select_sample(apartmentsTest, 10)

cp_rf <- ceteris_paribus(explainer_rf, apartments_small)
cp_rf

cp_rf <- ceteris_paribus(explainer_rf, apartments_small, y = apartments_small$m2.price)
cp_rf

## End(Not run)
```
**Usage**

ceteris_paribus_layer(
  x,
  ..., 
  size = 1,
  alpha = 0.3,
  color = "black",
  size_points = 2,
  alpha_points = 1,
  color_points = color,
  size_rugs = 0.5,
  alpha_rugs = 1,
  color_rugs = color,
  size_residuals = 1,
  alpha_residuals = 1,
  color_residuals = color,
  only_numerical = TRUE,
  show_profiles = TRUE,
  show_observations = TRUE,
  show_rugs = FALSE,
  show_residuals = FALSE,
  aggregate_profiles = NULL,
  as.gg = FALSE,
  facet_ncol = NULL,
  selected_variables = NULL,
  init_plot = FALSE
)

**Arguments**

- **x**
  a ceteris paribus explainer produced with function `ceteris_paribus()`

- **...**
  other explainers that shall be plotted together

- **size**
  a numeric. Size of lines to be plotted

- **alpha**
  a numeric between 0 and 1. Opacity of lines

- **color**
  a character. Either name of a color or name of a variable that should be used for coloring

- **size_points**
  a numeric. Size of points to be plotted

- **alpha_points**
  a numeric between 0 and 1. Opacity of points

- **color_points**
  a character. Either name of a color or name of a variable that should be used for coloring

- **size_rugs**
  a numeric. Size of rugs to be plotted

- **alpha_rugs**
  a numeric between 0 and 1. Opacity of rugs

- **color_rugs**
  a character. Either name of a color or name of a variable that should be used for coloring

- **size_residuals**
  a numeric. Size of line and points to be plotted for residuals
alpha_residuals
a numeric between 0 and 1. Opacity of points and lines for residuals

color_residuals
a character. Either name of a color or name of a variable that should be used for
coloring for residuals

only_numerical
a logical. If TRUE then only numerical variables will be plotted. If FALSE then
only categorical variables will be plotted.

show_profiles
a logical. If TRUE then profiles will be plotted. Either individual or aggregate
(see ‘aggregate_profiles’)

show_observations
a logical. If TRUE then individual observations will be marked as points

show_rugs
a logical. If TRUE then individual observations will be marked as rugs

show_residuals
a logical. If TRUE then residuals will be plotted as a line ended with a point

aggregate_profiles
function. If NULL (default) then individual profiles will be plotted. If a function
(e.g. mean or median) then profiles will be aggregated and only the aggregate
profile will be plotted

as.gg
if TRUE then returning plot will have gg class

facet_ncol
number of columns for the ‘facet_wrap()’.  

selected_variables
if not NULL then only ‘selected_variables’ will be presented

init_plot
technical parameter, do not use.

Value
a ggplot2 object

Examples
library("DALEX")
## Not run:
library("randomForest")
set.seed(59)

apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
   no.rooms + district, data = apartments)

explainer_rf <- explain(apartments_rf_model,
   data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)

apartments_small_1 <- apartmentsTest[1,]
apartments_small_2 <- select_sample(apartmentsTest, n = 20)
apartments_small_3 <- select_neighbours(apartmentsTest, apartments_small_1, n = 20)

cp_rf_y1 <- ceteris_paribus(explainer_rf, apartments_small_1, y = apartments_small_1$m2.price)
cp_rf_y2 <- ceteris_paribus(explainer_rf, apartments_small_2, y = apartments_small_2$m2.price)
cp_rf_y3 <- ceteris_paribus(explainer_rf, apartments_small_3, y = apartments_small_3$m2.price)
local_conditional_expectations

Local Conditional Expectation Explainer

Description

This explainer works for individual observations. For each observation it calculates Local Conditional Expectation (LCE) profiles for selected variables.

Usage

local_conditional_expectations(
  explainer,
  observations,
  y = NULL,
  variable_splits = NULL,
  variables = NULL,
  grid_points = 101
)

Arguments

 explainer a model to be explained, preprocessed by function 'DALEX::explain()'.
 observations set of observation for which profiles are to be calculated
**localconditionalexpectations**

*y*  
true labels for ‘observations’. If specified then will be added to local conditional expectations plots.

**variable_splits**  
named list of splits for variables, in most cases created with ‘calculate_variable_splits()’. If NULL then it will be calculated based on validation data available in the ‘ex- plainer’.

**variables**  
names of variables for which profiles shall be calculated. Will be passed to ‘calculate_variable_splits()’. If NULL then all variables from the validation data will be used.

**grid_points**  
number of points for profile. Will be passed to ‘calculate_variable_splits()’.

**Value**
An object of the class ’ceteris_paribus_explainer’. A data frame with calculated LCE profiles.

**Examples**

```r
library("DALEX")
## Not run:
library("randomForest")
set.seed(59)

apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
  no.rooms + district, data = apartments)

explainer_rf <- explain(apartments_rf_model,
  data = apartments[,2:6], y = apartments$m2.price)

new_apartment <- apartments[1, ]

cp_rf <- ceteris_paribus(explainer_rf, new_apartment)
lce_rf <- local_conditional_expectations(explainer_rf, new_apartment)
lce_rf

lce_rf <- local_conditional_expectations(explainer_rf, new_apartment, y = new_apartment$m2.price)
lce_rf

# Plot LCE
sel_vars <- c("surface", "no.rooms")
plot(lce_rf, selected_variables = sel_vars)

# Compare ceteris paribus profiles with LCE profiles
plot(cp_rf, selected_variables = sel_vars) +
  ceteris_paribus_layer(lce_rf, selected_variables = sel_vars, color = "red")

## End(Not run)
```
local_fit  

**Local Fit / Wangkardu Explanations**

**Description**

Local Fit / Wangkardu Explanations

**Usage**

```r
def local_fit(
    explainer,
    observation,
    selected_variable,
    grid_points = 101,
    select_points = 0.1
)
```

**Arguments**

- **explainer**: a model to be explained, preprocessed by the 'DALEX::explain' function
- **observation**: a new observavration for which predictions need to be explained
- **selected_variable**: variable to be presented in the local fit plot
- **grid_points**: number of points used for response path
- **select_points**: fraction of points fromvalidation data to be presented in local fit plots

**Value**

An object of the class 'local_fit_explainer'. It’s a data frame with calculated average responses.

**Examples**

```r
library("DALEX")
## Not run:
library("randomForest")
set.seed(59)

apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
  no.rooms + district, data = apartments)

explainer_rf <- explain(apartments_rf_model,
  data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)

new_apartment <- apartmentsTest[1,]
new_apartment

cr_rf <- local_fit(explainer_rf, observation = new_apartment,
```

```
Function `plot.ceteris_paribus_explainer` plots Ceteris Paribus Plots for selected observations. Various parameters help to decide what should be plotted, profiles, aggregated profiles, points or rugs.

Usage

```r
## S3 method for class 'ceteris_paribus_explainer'
plot(
  x,
  ..., size = 1,
  alpha = 0.3,
  color = "black",
  size_points = 2,
  alpha_points = 1,
  color_points = color,
  size_rugs = 0.5,
  alpha_rugs = 1,
  color_rugs = color,
  size_residuals = 1,
  alpha_residuals = 1,
  color_residuals = color,
  only_numerical = TRUE,
  show_profiles = TRUE,
  show_observations = TRUE,
  show_rugs = FALSE,
  show_residuals = FALSE,
  aggregate_profiles = NULL,
  as.gg = FALSE,
  facet_ncol = NULL,
  selected_variables = NULL
)
```

Arguments

- `x`: a ceteris paribus explainer produced with function `ceteris_paribus()`
- `...`: other explainers that shall be plotted together
plot.ceteris_paribus_explainer

size  a numeric. Size of lines to be plotted
alpha  a numeric between 0 and 1. Opacity of lines
color  a character. Either name of a color or name of a variable that should be used for
coloring
size_points  a numeric. Size of points to be plotted
alpha_points  a numeric between 0 and 1. Opacity of points
color_points  a character. Either name of a color or name of a variable that should be used for
coloring
size_rugs  a numeric. Size of rugs to be plotted
alpha_rugs  a numeric between 0 and 1. Opacity of rugs
color_rugs  a character. Either name of a color or name of a variable that should be used for
coloring
size_residuals  a numeric. Size of line and points to be plotted for residuals
alpha_residuals  a numeric between 0 and 1. Opacity of points and lines for residuals
color_residuals  a character. Either name of a color or name of a variable that should be used for
coloring for residuals
only_numerical  a logical. If TRUE then only numerical variables will be plotted. If FALSE then
only categorical variables will be plotted.
show_profiles  a logical. If TRUE then profiles will be plotted. Either individual or aggregate
(see ‘aggregate_profiles’)
show_observations  a logical. If TRUE then individual observations will be marked as points
show_rugs  a logical. If TRUE then individual observations will be marked as rugs
show_residuals  a logical. If TRUE then residuals will be plotted as a line ended with a point
aggregate_profiles  function. If NULL (default) then individual profiles will be plotted. If a function
(e.g. mean or median) then profiles will be aggregated and only the aggregate
profile will be plotted
as.gg  if TRUE then returning plot will have gg class
facet_ncol  number of columns for the ‘facet_wrap()’
selected_variables  if not NULL then only ‘selected_variables’ will be presented

Value

a ggplot2 object
Examples

```r
library("DALEX")
## Not run:
library("randomForest")
set.seed(59)
apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
                                   no.rooms + district, data = apartments)
explainer_rf <- explain(apartments_rf_model,
                        data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)
apartments_small <- apartmentsTest[1:20,]
apartments_small_1 <- apartmentsTest[1,]
apartments_small_2 <- select_sample(apartmentsTest, n = 20)
apartments_small_3 <- select_neighbours(apartmentsTest, apartments_small_1, n = 20)
cp_rf <- ceteris_paribus(explainer_rf, apartments_small)
cp_rf_1 <- ceteris_paribus(explainer_rf, apartments_small_1)
cp_rf_2 <- ceteris_paribus(explainer_rf, apartments_small_2)
cp_rf_3 <- ceteris_paribus(explainer_rf, apartments_small_3)
cp_rf

cp_rf_y <- ceteris_paribus(explainer_rf, apartments_small, y = apartments_small$m2.price)
cp_rf_y1 <- ceteris_paribus(explainer_rf, apartments_small_1, y = apartments_small_1$m2.price)
cp_rf_y2 <- ceteris_paribus(explainer_rf, apartments_small_2, y = apartments_small_2$m2.price)
cp_rf_y3 <- ceteris_paribus(explainer_rf, apartments_small_3, y = apartments_small_3$m2.price)

plot(cp_rf_y, show_profiles = TRUE, show_observations = TRUE,
     show_residuals = TRUE, color = "black",
     alpha = 0.3, alpha_points = 1, alpha_residuals = 0.5,
     size_points = 2, size_rugs = 0.5)

plot(cp_rf_y, show_profiles = TRUE, show_observations = TRUE,
     show_residuals = TRUE, color = "black",
     selected_variables = c("construction_year", "surface"),
     alpha = 0.3, alpha_points = 1, alpha_residuals = 0.5,
     size_points = 2, size_rugs = 0.5)

plot(cp_rf_y1, show_profiles = TRUE, show_observations = TRUE, show_rugs = TRUE,
     show_residuals = TRUE, alpha = 0.5, size_points = 3,
     alpha_points = 1, size_rugs = 0.5)

plot(cp_rf_y2, show_profiles = TRUE, show_observations = TRUE, show_rugs = TRUE,
     alpha = 0.2, alpha_points = 1, size_rugs = 0.5)

plot(cp_rf_y3, show_profiles = TRUE, show_observations = TRUE, show_rugs = TRUE,
     show_residuals = TRUE, alpha = 0.2, color_residuals = "orange", size_rugs = 0.5)

plot(cp_rf_y, show_profiles = TRUE, show_observations = TRUE, show_rugs = TRUE,
     show_residuals = TRUE, alpha = 0.5, color = "surface", as.gg = TRUE) +
     scale_color_gradient(low = "darkblue", high = "darkred")
```
plot.ceteris_paribus_oscillations

Plot Ceteris Paribus Oscillations

Description

Function `plot.ceteris_paribus_oscillations` plots variable importance plots.

Usage

```r
## S3 method for class 'ceteris_paribus_oscillations'
plot(x, ...)
```

Arguments

- `x` a ceteris paribus oscillation explainer produced with function `calculate_oscillations()`
- `...` other explainers that shall be plotted together

Value

A ggplot2 object

Examples

```r
library("DALEX")
## Not run:
library("randomForest")
set.seed(59)
apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
  no.rooms + district, data = apartments)
explainer_rf <- explain(apartments_rf_model,
  data = apartmentsTest, y = apartmentsTest$m2.price)
apartment <- apartmentsTest[1:2,]
```
Function `plot.local_fit_explainer` plots Local Fit Plots for a single prediction / observation.

### Usage

```r
## S3 method for class 'local_fit_explainer'
plot(x, ..., plot_residuals = TRUE, palette = "default")
```

### Arguments

- **x**: a local fit explainer produced with the `local_fit` function
- **...**: other explainers that shall be plotted together
- **plot_residuals**: if TRUE (default) then residuals are plotted as red/blue bars
- **palette**: color palette. Currently the choice is limited to `wangkardu` and `default`

### Value

a ggplot2 object

### Examples

```r
library("DALEX")
## Not run:
library("randomForest")
set.seed(59)

apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor + no.rooms + district, data = apartments)

explainer_rf <- explain(apartments_rf_model,
                         data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)

new_apartment <- apartmentsTest[1,]
new_apartment
```
plot.what_if_2d_explainer

Plot What If 2D Explanations

Description

Function `plot.what_if_2d_explainer` plots What-If Plots for a single prediction / observation.

Usage

```r
## S3 method for class 'what_if_2d_explainer'
plot(
  x,
  ...,
  split_ncol = NULL,
  add_raster = TRUE,
  add_contour = TRUE,
  add_observation = TRUE,
  bins = 3
)
```
plot.what_if_2d_explainer

Arguments

- `x`: a ceteris paribus explainer produced with the `what_if_2d` function
- `...`: currently will be ignored
- `split_ncol`: number of columns for the 'facet_wrap'
- `add_raster`: if TRUE then 'geom_raster' will be added to present levels with diverging colors
- `add_contour`: if TRUE then 'geom_contour' will be added to present contours
- `add_observation`: if TRUE then 'geom_point' will be added to present observation that is explained
- `bins`: number of contours to be added

Value

a ggplot2 object

Examples

```r
library("DALEX")
## Not run:
library("randomForest")
set.seed(59)

apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
    no.rooms + district, data = apartments)

explainer_rf <- explain(apartments_rf_model,
    data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)

new_apartment <- apartmentsTest[1, ]

wi_rf_2d <- what_if_2d(explainer_rf, observation = new_apartment)

plot(wi_rf_2d)
plot(wi_rf_2d, add_contour = FALSE)
plot(wi_rf_2d, add_observation = FALSE)
plot(wi_rf_2d, add_raster = FALSE)

# HR data
model <- randomForest(status ~ gender + age + hours + evaluation + salary, data = HR)
pred1 <- function(m, x) predict(m, x, type = "prob")[,1]
explainer_rf_fired <- explain(model, data = HR[,1:5],
    y = HR$status == "fired",
    predict_function = pred1, label = "fired")

new_emp <- HR[1, ]

new_emp
```
plot.what_if_explainer

```r
wi_rf_2d <- what_if_2d(explainer_rf_fired, observation = new_emp)
wi_rf_2d

plot(wi_rf_2d)
```

## End(Not run)

---

**plot.what_if_explainer**

*Plot What If Explanations*

---

**Description**

Function `plot.what_if_explainer` plots What-If Plots for a single prediction / observation.

**Usage**

```r
## S3 method for class 'what_if_explainer'
plot(
  x,
  ..., 
  quantiles = TRUE,
  split = "models",
  split_ncol = NULL,
  color = "variables"
)
```

**Arguments**

- `x` a ceteris paribus explainer produced with the `what_if` function
- `...` other explainers that shall be plotted together
- `quantiles` if TRUE (default) then quantiles will be presented on OX axis. If FALSE then original values will be presented on OX axis
- `split` a character, either 'models' or 'variables'. Sets the variable for faceting
- `split_ncol` number of columns for the 'facet_wrap'
- `color` a character, either 'models' or 'variables'. Sets the variable for coloring

**Value**

a ggplot2 object
Examples

library("DALEX")
## Not run:
library("randomForest")
set.seed(59)

apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
no.rooms + district, data = apartments)

explainer_rf <- explain(apartments_rf_model,
data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)

new_apartment <- apartmentsTest[1,]

wi_rf <- what_if(explainer_rf, observation = new_apartment)

plot(wi_rf, split = "variables", color = "variables")
plot(wi_rf)

## End(Not run)

plot_interactive

Plots Interactive What-If Explanations

Description

Function 'plot_interactive.what_if_explainer' plots Ceteris Paribus Plots for a single prediction.

Usage

## S3 method for class 'what_if_explainer'
plot_interactive(x, ..., split = "models", color = "variables")

plot_interactive(x, ...)

## Default S3 method:
plot_interactive(x, ..., split = "models", color = "variables")

Arguments

x a ceteris_paribus explainer produced with the 'ceteris_paribus' function
...
other explainers that shall be plotted together
split a character, either 'models' or 'variables'. Sets the variable for faceting
color a character, either 'models' or 'variables'. Sets the variable for coloring
Value

a ggiraph object

Examples

library("DALEX")
## Not run:
library("ggiraph")
library("randomForest")
set.seed(59)

apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
  no.rooms + district, data = apartments)

explainer_rf <- explain(apartments_rf_model,
  data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)

new_apartment <- apartmentsTest[1, ]
new_apartment

wi_rf <- what_if(explainer_rf, observation = new_apartment)
wi_rf

plot_interactive(wi_rf, split = "variables", color = "variables")
## End(Not run)

print.ceteris_paribus_explainer

Print Ceteris Paribus Explainer Summary

Description

Print Ceteris Paribus Explainer Summary

Usage

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

Arguments

x a ceteris_paribus explainer produced with the 'ceteris_paribus()' function

... other arguments that will be passed to 'head()'
Examples

```r
library("DALEX")
## Not run:
library("randomForest")
set.seed(59)

apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
             no.rooms + district, data = apartments)

explainer_rf <- explain(apartments_rf_model,
             data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)

apartments_small <- select_sample(apartmentsTest, 10)

cp_rf <- ceteris_paribus(explainer_rf, apartments_small)
cp_rf
## End(Not run)
```

---

**print.ceteris_paribus_profile**

*Print Ceteris Paribus Profiles*

**Description**

Print Ceteris Paribus Profiles

**Usage**

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

**Arguments**

- `x` : a ceteris paribus profile produced with the `calculate_profiles` function
- `...` : other arguments that will be passed to `head()`

**Examples**

```r
library("DALEX")
## Not run:
library("randomForest")
set.seed(59)

apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
             no.rooms + district, data = apartments)

vars <- c("construction.year", "surface", "floor", "no.rooms", "district")
variable_splits <- calculate_variable_splits(apartments, vars)
new_apartment <- apartmentsTest[1:10, ]
profiles <- calculate_profiles(new_apartment, variable_splits,
```
# only subset of observations
small_apartments <- select_sample(apartmentsTest, n = 10)
small_apartments
small_profiles <- calculate_profiles(small_apartments, variable_splits, apartments_rf_model)
small_profiles

# neighbors for a selected observation
new_apartment <- apartments[1, 2:6]
small_apartments <- select_neighbours(apartmentsTest, new_apartment, n = 10)
small_apartments
small_profiles <- calculate_profiles(small_apartments, variable_splits, apartments_rf_model)
new_apartment
small_profiles

## End(Not run)

---

**print.local_fit_explainer**

*Prints Local Fit / Wangkardu Summary*

### Description

Prints Local Fit / Wangkardu Summary

### Usage

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

### Arguments

- `x`: a local fit explainer produced with the `local_fit` function
- `...`: other arguments that will be passed to `head` function

### Examples

```r
library("DALEX")
## Not run:
library("randomForest")
apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
  no.rooms + district, data = apartments)
explainer_rf <- explain(apartments_rf_model,
  data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)
new_apartment <- apartmentsTest[1, ]
```
new_apartment

cr_rf <- local_fit(explainer_rf, observation = new_apartment,
    select_points = 0.002, selected_variable = "surface")

## End(Not run)

---

**print.plot_ceteris_paribus_explainer**

*Print Ceteris Paribus Explainer Summary*

**Description**

See more examples in the ceteris_paribus_layer function

**Usage**

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

**Arguments**

- `x`    
  a plot_ceteris_paribus_explainer object to plot

- `...`  
  other arguments that will be passed to `print.ggplot()`

---

**print.what_if_2d_explainer**

*Print What If 2D Explainer Summary*

**Description**

Print What If 2D Explainer Summary

**Usage**

```
## S3 method for class 'what_if_2d_explainer'
print(x, ...)
```

**Arguments**

- `x`    
  a what_if_2d explainer produced with the 'what_if_2d' function

- `...`  
  other arguments that will be passed to head()
Examples

library("DALEX")
## Not run:
library("randomForest")
apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
                                    no.rooms + district, data = apartments)
explainer_rf <- explain(apartments_rf_model,
                        data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)
new_apartment <- apartmentsTest[1, ]
new_apartment
## End(Not run)

print.what_if_explainer

Description

Print What If Explainer Summary

Usage

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

Arguments

x a what_if explainer produced with the 'what_if' function
...
other arguments that will be passed to head()

Examples

library("DALEX")
## Not run:
library("randomForest")
apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
                                    no.rooms + district, data = apartments)
explainer_rf <- explain(apartments_rf_model,
                        data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)
new_apartment <- apartmentsTest[1, ]
new_apartment
## End(Not run)
select_neighbours  
Select Subset of Rows Closest to a Specified Observation

Description
This function selects subset of rows from data set. This is useful if data is large and we need just a sample to calculate profiles.

Usage
```r
select_neighbours(
  data,
  observation,
  variables = NULL,
  distance = gower::gower_dist,
  n = 20,
  frac = NULL
)
```

Arguments
- **data**: set of observations
- **observation**: single observation
- **variables**: variables that shall be used for calculation of distance. By default these are all variables present in ‘data’ and ‘observation’
- **distance**: distance function, by default the ‘gower_dist’ function.
- **n**: number of neighbours to select
- **frac**: if ‘n’ is not specified (NULL), then will be calculated as ‘frac’ * number of rows in ‘data’. Either ‘n’ or ‘frac’ need to be specified.

Details
Note that `select_neighbours` function is S3 generic. If you want to work on non standard data sources (like H2O ddf, external databases) you should overload it.

Value
a data frame with selected rows

Examples
```r
library("DALEX")

new_apartment <- apartments[1, 2:6]
small_apartments <- select_neighbours(apartmentsTest, new_apartment, n = 10)
new_apartment
small_apartments
```
**select_sample**

*Select Subset of Rows*

**Description**

This function selects subset of rows from data set. This is useful if data is large and we need just a sample to calculate profiles.

**Usage**

```r
select_sample(data, n = 100, seed = 1313)
```

**Arguments**

- `data`: set of observations. Profile will be calculated for every observation (every row)
- `n`: named list of vectors. Elements of the list are vectors with points in which profiles should be calculated. See an example for more details.
- `seed`: seed for random number generator.

**Details**

Note that `select_subsample` function is S3 generic. If you want to work on non standard data sources (like H2O ddf, external databases) you should overload it.

**Value**

a data frame with selected rows

**Examples**

```r
library("DALEX")
small_apartments <- select_sample(apartmentsTest)
head(small_apartments)
```

---

**what_if**

*What-If Plot*

**Description**

What-If Plot

**Usage**

```r
what_if(explainer, observation, grid_points = 101, selected_variables = NULL)
```
what_if_2d

## Arguments

- **explainer**: a model to be explained, preprocessed by the `DALEX::explain` function
- **observation**: a new observation for which predictions need to be explained
- **grid_points**: number of points used for response path
- **selected_variables**: if specified, then only these variables will be explained

## Value

An object of the class `what_if_explainer`. It’s a data frame with calculated average responses.

## Examples

```r
library("DALEX")
## Not run:
library("randomForest")
set.seed(59)

apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
                                    no.rooms + district, data = apartments)

explainer_rf <- explain(apartments_rf_model,
                         data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)

new_apartment <- apartmentsTest[1,]
new_apartment

wi_rf <- what_if(explainer_rf, observation = new_apartment)
wi_rf
wi_rf <- what_if(explainer_rf, observation = new_apartment,
                 selected_variables = c("surface", "floor", "no.rooms"))
wi_rf

## End(Not run)
```

## Description

This function calculates what if scores for grid of values spanned by two variables.

## Usage

```r
what_if_2d(
  explainer,
  observation,
  grid_points = 101,
  selected_variables = NULL
)
```
Arguments

- **explainer**: a model to be explained, preprocessed by the 'DALEX::explain' function
- **observation**: a new observation for which predictions need to be explained
- **grid_points**: number of points used for response path. Will be used for both variables
- **selected_variables**: if specified, then only these variables will be explained

Value

An object of the class 'what_if_2d_explainer'. It’s a data frame with calculated average responses.

Examples

```r
library("DALEX")
## Not run:
library("randomForest")
set.seed(59)

apartments_rf_model <- randomForest(m2.price ~ construction.year + surface + floor +
    no.rooms + district, data = apartments)

explainer_rf <- explain(apartments_rf_model,
    data = apartmentsTest[,2:6], y = apartmentsTest$m2.price)

new_apartment <- apartmentsTest[1,]

wi_rf_2d <- what_if_2d(explainer_rf, observation = new_apartment,
                         selected_variables = c("surface", "floor", "no.rooms"))

plot(wi_rf_2d)

## End(Not run)
```
Index

+ .plot_ceteris_paribus_explainer, 2
calculate_oscillations, 3
calculate_profiles, 3
calculate_profiles_ice, 5
calculate_variable_splits, 6
ceteris_paribus, 7
ceteris_paribus_layer, 8
local_conditional_expectations, 11
local_fit, 13
plot.ceteris_paribus_explainer, 14
plot.ceteris_paribus_oscillations, 17
plot.local_fit_explainer, 18
plot.what_if_2d_explainer, 19
plot.what_if_explainer, 21
plot_interactive, 22
print.ceteris_paribus_explainer, 23
print.ceteris_paribus_profile, 24
print.local_fit_explainer, 25
print.plot_ceteris_paribus_explainer, 26
print.what_if_2d_explainer, 26
print.what_if_explainer, 27
select_neighbours, 28
select_sample, 29
what_if, 29
what_if_2d, 30