Package ‘DALEX’

February 21, 2020

Title  moDel Agnostic Language for Exploration and eXplanation
Version  1.0
Description  Unverified black box model is the path to the failure. Opaqueness leads to distrust. Distrust leads to ignorance. Ignoration leads to rejection.
DALEX package xrays any model and helps to explore and explain its behaviour.
Machine Learning (ML) models are widely used and have various applications in classification or regression. Models created with boosting, bagging, stacking or similar techniques are often used due to their high performance. But such black-box models usually lack of direct interpretability.
DALEX package contains various methods that help to understand the link between input variables and model output. Implemented methods help to explore model on the level of a single instance as well as a level of the whole dataset.
All model explainers are model agnostic and can be compared across different models.
DALEX package is the cornerstone for ‘DrWhy.AI’ universe of packages for visual model exploration.
Find more details in (Biecek 2018) <arXiv:1806.08915>.

License  GPL
Encoding  UTF-8
LazyData  true
RoxygenNote  7.0.2
Depends  R (>= 3.5)
Imports  ggplot2
Suggests  gower, ingredients, iBreakDown, ggpubr, ranger, testthat
URL  https://ModelOriented.github.io/DALEX/,
     https://github.com/ModelOriented/DALEX
BugReports  https://github.com/ModelOriented/DALEX/issues
NeedsCompilation  no
Author  Przemyslaw Biecek [aut, cre] (https://orcid.org/0000-0001-8423-1823),
        Szymon Maksymiuk [ctb]
Maintainer  Przemyslaw Biecek <przemyslaw.biecek@gmail.com>
Repository: CRAN
Date/Publication: 2020-02-21 13:20:02 UTC

R topics documented:

- apartments ................................................. 3
- colors_discrete_drwhy ................................. 3
- dragons .................................................. 4
- explain.default ....................................... 5
- HR ....................................................... 8
- install_dependencies ................................. 9
- loss_cross_entropy .................................. 9
- model_diagnostics .................................. 10
- model_info ............................................. 11
- model_parts ........................................... 13
- model_performance ................................. 14
- model_profile .......................................... 15
- plot.model_diagnostics ......................... 17
- plot.model_performance ...................... 18
- plot.model_profile .................................. 20
- plot.predict_diagnostics .................... 21
- plot.predict_profile ......................... 22
- predict.explainer ................................ 23
- predict_diagnostics ............................... 24
- predict_parts .......................................... 25
- predict_profile ....................................... 27
- print.description .................................. 28
- print.explainer ....................................... 29
- print.model_diagnostics .................... 30
- print.model_info .................................... 30
- print.model_performance .................. 31
- print.model_profile ................................ 31
- print.predict_diagnostics ............... 32
- theme_drwhy ........................................ 32
- titanic ............................................... 33
- update_data ......................................... 34
- update_label .......................................... 35
- variable_effect ..................................... 35
- yhat .................................................... 36

Index 39
Data structure of the dataset is copied from a real dataset from PBImisc package, but they were generated in a way to mimic the effect of the Anscombe quartet for complex black box models.

Usage

data(apartments)

Format

A data frame with 1000 rows and 6 columns

Details

- `m2.price` - price per square meter
- `surface` - apartment area in square meters
- `n.rooms` - number of rooms (correlated with surface)
- `district` - district in which apartment is located, factor with 10 levels
- `floor` - floor
- `construction.date` - construction year

Description

DrWhy color palettes for ggplot objects

Usage

colors_discrete_drwhy(n = 2)
colors_diverging_drwhy()
colors_breakdown_drwhy()

Arguments

- `n` - number of colors for color palette
Value

color palette as vector of characters

<table>
<thead>
<tr>
<th>dragons</th>
<th>Dragon Data</th>
</tr>
</thead>
</table>

Description

Datasets dragons and dragons_test are artificial, generated from the same ground truth model, but with sometimes different data distribution.

Usage

data(dragons)

Format

a data frame with 2000 rows and 8 columns

Details

Values are generated in a way to:

- have nonlinearity in year_of_birth and height
- have concept drift in the test set

- year_of_birth - year in which the dragon was born. Negative year means year BC, eg: -1200 = 1201 BC
- year_of_discovery - year in which the dragon was found.
- height - height of the dragon in yards.
- weight - weight of the dragon in tons.
- scars - number of scars.
- colour - colour of the dragon.
- number_of_lost_teeth - number of teeth that the dragon lost.
- life_length - life length of the dragon.
explain.default

Create Model Explainer

Description

Black-box models may have very different structures. This function creates a unified representation of a model, which can be further processed by various explainers.

Usage

```r
explain.default(
  model,
  data = NULL,
  y = NULL,
  predict_function = NULL,
  residual_function = NULL,
  weights = NULL,
  ...,
  label = NULL,
  verbose = TRUE,
  precalculate = TRUE,
  colorize = TRUE,
  model_info = NULL,
  type = NULL
)
```

```r
explain(
  model,
  data = NULL,
  y = NULL,
  predict_function = NULL,
  residual_function = NULL,
  weights = NULL,
  ...,
  label = NULL,
  verbose = TRUE,
  precalculate = TRUE,
  colorize = TRUE,
  model_info = NULL,
  type = NULL
)
```

Arguments

- `model` - object - a model to be explained
- `data` - data.frame or matrix - data that was used for fitting. If not provided then will be extracted from the model. Data should be passed without target column (this
shall be provided as the \textit{y} argument). NOTE: If target variable is present in the data, some of the functionalities may not work properly.

\textbf{y} \textit{numeric vector with outputs / scores}. If provided then it shall have the same size as data.

\textbf{predict\_function} \textit{function that takes two arguments: model and new data and returns numeric vector with predictions.}

\textbf{residual\_function} \textit{function that takes three arguments: model, data and response vector \textit{y}. It should return a numeric vector with model residuals for given data. If not provided, response residuals \((y − ˆy)\) are calculated.}

\textbf{weights} \textit{numeric vector with sampling weights}. By default it’s \texttt{NULL}. If provided then it shall have the same length as data.

\ldots \textit{other parameters}

\textbf{label} \textit{character - the name of the model}. By default it’s extracted from the 'class' attribute of the model.

\textbf{verbose} \textit{if TRUE (default) then diagnostic messages will be printed.}

\textbf{precalculate} \textit{if TRUE (default) then \texttt{predicted\_values} and \texttt{residual} are calculated when explainer is created. This will happen also if \texttt{verbose} is TRUE. Set both \texttt{verbose} and \texttt{precalculate} to FALSE to omit calculations.}

\textbf{colorize} \textit{if TRUE (default) then \texttt{WARNINGS}, \texttt{ERRORS} and \texttt{NOTES} are colorized. Will work only in the R console.}

\textbf{model\_info} \textit{a named list (package, version, type) containing information about model}. If \texttt{NULL}, \texttt{DALEX} will seek for information on its own.

\textbf{type} \textit{type of a model, either classification or regression}. If not specified then type will be extracted from \texttt{model\_info}.

\subsection*{Details}

Please \texttt{NOTE}, that the \texttt{model} is the only required argument. But some explainers may require that other arguments will be provided too.

\subsection*{Value}

An object of the class \texttt{explainer}.

It’s a list with following fields:

\begin{itemize}
  \item \texttt{model} the explained model.
  \item \texttt{data} the dataset used for training.
  \item \texttt{y} response for observations from data.
  \item \texttt{weights} sample weights for data. \texttt{NULL} if weights are not specified.
  \item \texttt{y\_hat} calculated predictions.
  \item \texttt{residuals} calculated residuals.
\end{itemize}
• `predict_function` function that may be used for model predictions, shall return a single numerical value for each observation.
• `residual_function` function that returns residuals, shall return a single numerical value for each observation.
• `class` class/classes of a model.
• `label` label of explainer.
• `model_info` named list containing basic information about model, like package, version of package and type.

References


Examples

```r
# simple explainer for regression problem
aps_lm_model4 <- lm(m2.price ~ ., data = apartments)
aps_lm_explainer4 <- explain(aps_lm_model4, data = apartments, label = "model_4v")
aps_lm_explainer4

# various parameters for the explain function
# all defaults
aps_lm <- explain(aps_lm_model4)

# silent execution
aps_lm <- explain(aps_lm_model4, verbose = FALSE)

# user provided predict_function
aps_lm <- explain(aps_lm_model4, data = apartments, label = "model_4v", predict_function = predict)

# set target variable
aps_lm <- explain(aps_lm_model4, data = apartments, label = "model_4v", y = apartments$m2.price)
aps_lm <- explain(aps_lm_model4, data = apartments, label = "model_4v", y = apartments$m2.price, predict_function = predict)

# set model_info
model_info <- list(package = "stats", ver = "3.6.2", type = "regression")
aps_lm_model4 <- lm(m2.price ~ ., data = apartments)
aps_lm_explainer4 <- explain(aps_lm_model4, data = apartments, label = "model_4v", model_info = model_info)

## Not run:
# set model_info
model_info <- list(package = "stats", ver = "3.6.2", type = "regression")
aps_lm_model4 <- lm(m2.price ~ ., data = apartments)
aps_lm_explainer4 <- explain(aps_lm_model4, data = apartments, label = "model_4v", model_info = model_info)

aps_lm_explainer4 <- explain(aps_lm_model4, data = apartments, label = "model_4v",
weights = as.numeric(apartments$construction.year > 2000))
```
# more complex model
library("ranger")
aps_ranger_model4 <- ranger(m2.price ~ ., data = apartments, num.trees = 50)
aps_ranger_explainer4 <- explain(aps_ranger_model4, data = apartments, label = "model_ranger")
aps_ranger_explainer4

## End(Not run)

---

Human Resources Data

Description

Datasets HR and HR_test are artificial, generated from the same model. Structure of the dataset is based on real data, from Human Resources department with information which employees were promoted, which were fired.

Usage

data(HR)

Format

a data frame with 10000 rows and 6 columns

Details

Values are generated in a way to:
- have interaction between age and gender for the ‘fired’ variable
- have non monotonic relation for the salary variable
- have linear effects for hours and evaluation.

- gender - gender of an employee.
- age - age of an employee in the moment of evaluation.
- hours - average number of working hours per week.
- evaluation - evaluation in the scale 2 (bad) - 5 (very good).
- salary - level of salary in the scale 0 (lowest) - 5 (highest).
- status - target variable, either ‘fired’ or ‘promoted’ or ‘ok’.
install_dependencies

Install all dependencies for the DALEX package

Description

By default 'heavy' dependencies are not installed along DALEX. This function silently install all required packages.

Usage

install_dependencies(packages = c("ingredients", "iBreakDown", "ggpubr"))

Arguments

packages which packages shall be installed?

loss_cross_entropy Calculate Loss Functions

Description

Calculate Loss Functions

Usage

loss_cross_entropy(observed, predicted, p_min = 1e-04, na.rm = TRUE)
loss_sum_of_squares(observed, predicted, na.rm = TRUE)
loss_root_mean_square(observed, predicted, na.rm = TRUE)
loss_accuracy(observed, predicted, na.rm = TRUE)
loss_one_minus_auc(observed, predicted)

Arguments

observed observed scores or labels, these are supplied as explainer specific y
predicted predicted scores, either vector of matrix, these are returned from the model specific predict_function()
p_min for cross entropy, minimal value for probability to make sure that log will not explode
na.rm logical, should missing values be removed?
Value

numeric - value of the loss function

Examples

```r
## Not run:
library("ranger")
titanic_ranger_model <- ranger(survived~., data = titanic_imputed, num.trees = 50,
probability = TRUE)
loss_one_minus_auc(titanic_imputed$survived, yhat(titanic_ranger_model, titanic_imputed))

HR_ranger_model_multi <- ranger(status~., data = HR, num.trees = 50, probability = TRUE)
loss_cross_entropy(as.numeric(HR$status), yhat(HR_ranger_model_multi, HR))

## End(Not run)
```

model_diagnostics

Dataset Level Model Diagnostics

Description

This function performs model diagnostic of residuals. Residuals are calculated and plotted against predictions, true y values or selected variables. Find information how to use this function here: https://pbiecek.github.io/ema/residualDiagnostic.html.

Usage

```r
model_diagnostics(explainer, variables = NULL, ...)
```

Arguments

- explainer: a model to be explained, preprocessed by the explain function
- variables: character - name of variables to be explained. Default NULL stands for all variables
- ...: other parameters

Value

An object of the class model_diagnostics. It's a data frame with residuals and selected variables.

References

Examples

```r
apartments_lm_model <- lm(m2.price ~ ., data = apartments)
explainer_lm <- explain(apartments_lm_model,
  data = apartments,
  y = apartments$m2.price)
diag_lm <- model_diagnostics(explainer_lm)
diag_lm
plot(diag_lm)
## Not run:
library("ranger")
apartments_ranger_model <- ranger(m2.price ~ ., data = apartments)
explainer_ranger <- explain(apartments_ranger_model,
  data = apartments,
  y = apartments$m2.price)
diag_ranger <- model_diagnostics(explainer_ranger)
diag_ranger
plot(diag_ranger)
plot(diag_ranger, diag_lm)
plot(diag_ranger, diag_lm, variable = "y")
plot(diag_ranger, diag_lm, variable = "construction.year")
plot(diag_ranger, variable = "y", yvariable = "y_hat")
plot(diag_ranger, variable = "y", yvariable = "abs_residuals")
plot(diag_ranger, variable = "ids")
## End(Not run)
```

---

**model_info**

**Extract info from model**

### Description

This generic function let user extract base information about model. The function returns a named list of class `model_info` that contain about package of model, version and task type. For wrappers like `mlr` or `caret` both, package and wrapper information are stored.

### Usage

```r
model_info(model, ...)
```

## S3 method for class 'lm'
```
model_info(model, ...)
```

## S3 method for class 'randomForest'
```
model_info(model, ...)
```

## S3 method for class 'svm'
```
model_info(model, ...)
```
model_info

## S3 method for class 'glm'
model_info(model, ...)

## S3 method for class 'glmnet'
model_info(model, ...)

## S3 method for class 'cv.glmnet'
model_info(model, ...)

## S3 method for class 'ranger'
model_info(model, ...)

## S3 method for class 'gbm'
model_info(model, ...)

## S3 method for class 'model_fit'
model_info(model, ...)

## S3 method for class 'train'
model_info(model, ...)

## S3 method for class 'rpart'
model_info(model, ...)

## Default S3 method:
model_info(model, ...)

Arguments

model - model object
...
- another arguments

Currently supported packages are:

- class 'cv.glmnet' and 'glmnet' - models created with 'glmnet' package
- class 'glm' - generalized linear models
- class 'model_fit' - models created with 'parsnip' package
- class 'lm' - linear models created with 'stats::lm'
- class 'ranger' - models created with 'ranger' package
- class 'randomForest' - random forest models created with 'randomForest' package
- class 'svm' - support vector machines models created with the 'e1071' package
- class 'train' - models created with 'caret' package
- class 'gbm' - models created with 'gbm' package

Value

A named list of class model_info
Examples

```r
aps_lm_model4 <- lm(m2.price ~ ., data = apartments)
model_info(aps_lm_model4)

library("ranger")
model_regr_rf <- ranger::ranger(m2.price~., data = apartments, num.trees = 50)
model_info(model_regr_rf)
```

---

**model_parts**

Dataset Level Variable Importance as Change in Loss Function after Variable Permutations

Description

From DALEX version 1.0 this function calls the feature_importance. Find information how to use this function here: [https://pbiecek.github.io/ema/featureImportance.html](https://pbiecek.github.io/ema/featureImportance.html).

Usage

```r
model_parts(
  explainer,
  loss_function = loss_sum_of_squares,
  ..., 
  type = "variable_importance",
  n_sample = 1000
)
```

Arguments

- **explainer**: a model to be explained, preprocessed by the explain function
- **loss_function**: a function that will be used to assess variable importance
- **...**: other parameters
- **type**: character, type of transformation that should be applied for dropout loss. `variable_importance` and `raw` results raw drop lossess, `ratio` returns `drop_loss/drop_loss_full_model` while `difference` returns `drop_loss - drop_loss_full_model`
- **n_sample**: number of observations that should be sampled for calculation of variable importance. If negative then variable importance will be calculated on whole dataset (no sampling).

Value

An object of the class feature_importance. It’s a data frame with calculated average response.

References

Examples

## Not run:
library("ranger")
titanic_ranger_model <- ranger(survived~., data = titanic_imputed, num.trees = 50,
                                 probability = TRUE)
explainer_ranger <- explain(titanic_ranger_model, data = titanic_imputed[, -8],
                             y = titanic_imputed$survived)
vi_ranger <- model_parts(explainer_ranger, type = "raw")
head(vi_ranger, 8)
plot(vi_ranger)
titanic_glm_model <- glm(survived~., data = titanic_imputed, family = "binomial")
explainer_glm <- explain(titanic_glm_model, data = titanic_imputed[, -8],
                         y = titanic_imputed$survived)
logit <- function(x) exp(x)/(1+exp(x))
vi_glm <- model_parts(explainer_glm, type = "raw",
                      loss_function = function(observed, predicted)
                                      sum((observed - logit(predicted))^2))
head(vi_glm, 8)
plot(vi_glm)
## End(Not run)

model_performance

Dataset Level Model Performance Measures

Description

Function `model_performance()` calculates various performance measures for classification and regression models. For classification models following measures are calculated: F1, accuracy, recall, precision and AUC. For regression models following measures are calculated: mean squared error, R squared, median absolute deviation.

Usage

`model_performance(explainer, ..., cutoff = 0.5)`

Arguments

- `explainer`: a model to be explained, preprocessed by the `explain` function
- `...`: other parameters
- `cutoff`: a cutoff for classification models, needed for measures like recall, precision, ACC, F1. By default 0.5.

Value

An object of the class `model_performance`. 
model_profile

References


Examples

```r
## Not run:
library("ranger")
titanic_ranger_model <- ranger(survived~., data = titanic_imputed, num.trees = 100, probability = TRUE)
# It's a good practice to pass data without target variable
explainer_ranger <- explain(titanic_ranger_model, data = titanic_imputed[, -8], y = titanic_imputed$survived)
# resulting dataframe has predicted values and residuals
mp_ex_rn <- model_performance(explainer_ranger)

titanic_glm_model <- glm(survived~., data = titanic_imputed, family = "binomial")
explainer_glm <- explain(titanic_glm_model, data = titanic_imputed[, -8],
                         y = titanic_imputed$survived,
predict_function = function(m, x) predict.glm(m, x, type = "response"),
label = "glm")
mp_ex_glm <- model_performance(explainer_glm)
plot(mp_ex_glm)
plot(mp_ex_glm, mp_ex_rn)

titanic_lm_model <- lm(survived~., data = titanic_imputed)
explainer_lm <- explain(titanic_lm_model, data = titanic_imputed[, -8], y = titanic_imputed$survived)
mp_ex_lm <- model_performance(explainer_lm)
plot(mp_ex_lm)
plot(mp_ex_glm, mp_ex_rn, mp_ex_lm)
## End(Not run)
```

---

model_profile  

Dataset Level Variable Profile as Partial Dependence or Accumulated Local Dependence Explanations

Description

This function calculates explanations on a dataset level set that explore model response as a function of selected variables. The explanations can be calculated as Partial Dependence Profile or Accumulated Local Dependence Profile. Find information how to use this function here: https://pbiecek.github.io/ema/partialDependenceProfiles.html. The variable_profile function is a copy of model_profile.
Usage

model_profile(
  explainer,
  variables = NULL,
  N = 100,
  ..., 
  groups = NULL,
  k = NULL,
  center = TRUE,
  type = "partial"
)

variable_profile(
  explainer,
  variables = NULL,
  N = 100,
  ..., 
  groups = NULL,
  k = NULL,
  center = TRUE,
  type = "partial"
)

single_variable(explainer, variable, type = "pdp", ...)

Arguments

explainer a model to be explained, preprocessed by the explain function
variables character - names of variables to be explained
N number of observations used for calculation of aggregated profiles. By default 100.
... other parameters that will be passed to ingredients::aggregate_profiles
groups a variable name that will be used for grouping. By default NULL which means that no groups shall be calculated
k number of clusters for the hclust function (for clustered profiles)
center shall profiles be centered before clustering
type the type of variable profile. Either partial, conditional or accumulated.
variable deprecated, use variables instead

Details

Underneath this function calls the partial_dependency or accumulated_dependency functions from the ingredients package.

Value

An object of the class model_profile. It’s a data frame with calculated average model responses.
References


Examples

titanic_glm_model <- glm(survived~., data = titanic_imputed, family = "binomial")
explainer_glm <- explain(titanic_glm_model, data = titanic_imputed)
expl_glm <- model_profile(explainer_glm, "fare")
plot(expl_glm)

## Not run:
library("ranger")
titanic_ranger_model <- ranger(survived~., data = titanic_imputed, num.trees = 50,
probability = TRUE)
explainer_ranger <- explain(titanic_ranger_model, data = titanic_imputed)
expl_ranger <- model_profile(explainer_ranger)
plot(expl_ranger, geom = "profiles")

vp_ra <- model_profile(explainer_ranger, type = "partial", variables = c("age", "fare"))
plot(vp_ra, variables = c("age", "fare"), geom = "points")

vp_ra <- model_profile(explainer_ranger, type = "partial", k = 3)
plot(vp_ra, geom = "profiles")

vp_ra <- model_profile(explainer_ranger, type = "partial", groups = "gender")
plot(vp_ra, geom = "profiles")

vp_ra <- model_profile(explainer_ranger, type = "accumulated")
plot(vp_ra, geom = "profiles")

## End(Not run)

plot.model_diagnostics

Plot Dataset Level Model Diagnostics

Description

Plot Dataset Level Model Diagnostics

Usage

## S3 method for class 'model_diagnostics'
plot(x, ..., variable = "y_hat", yvariable = "residuals", smooth = TRUE)
```
plot.model_performance

Plot Dataset Level Model Performance Explanations

Description

Plot Dataset Level Model Performance Explanations
```

**Arguments**

- **x**: a data.frame to be explained, preprocessed by the `model_diagnostics` function
- **...**: other object to be included to the plot
- **variable**: character - name of the variable on OX axis to be explained, by default `y_hat`
- **yvariable**: character - name of the variable on OY axis, by default `residuals`
- **smooth**: logical shall the smooth line be added

**Value**

an object of the class `model_diagnostics_explainer`.

**Examples**

```r
apartments_lm_model <- lm(m2.price ~ ., data = apartments)
explainer_lm <- explain(apartments_lm_model, data = apartments, y = apartments$m2.price)
diag_lm <- model_diagnostics(explainer_lm)
diag_lm
plot(diag_lm)
## Not run:
library("ranger")
apartments_ranger_model <- ranger(m2.price ~ ., data = apartments)
explainer_ranger <- explain(apartments_ranger_model, data = apartments, y = apartments$m2.price)
diag_ranger <- model_diagnostics(explainer_ranger)
diag_ranger
plot(diag_ranger)
plot(diag_ranger, diag_lm)
plot(diag_ranger, diag_lm, variable = "y")
plot(diag_ranger, diag_lm, variable = "construction.year")
plot(diag_ranger, variable = "y", yvariable = "y_hat")
## End(Not run)
```
Usage

```r
## S3 method for class 'model_performance'
plot(
  x,
  ..., geom = "ecdf", show_outliers = 0,
  ptlabel = "name",
  lossFunction = function(x) sqrt(mean(x^2))
)
```

Arguments

- `x`: a model to be explained, preprocessed by the `explain` function
- `...`: other parameters
- `geom`: either "ecdf", "boxplot", "gain", "lift" or "histogram" determines how residuals shall be summarized
- `show_outliers`: number of largest residuals to be presented (only when `geom = boxplot`).
- `ptlabel`: either "name" or "index" determines the naming convention of the outliers
- `lossFunction`: function that calculates the loss for a model based on model residuals. By default it's the root mean square.

Value

An object of the class `model_performance`.

Examples

```r
## Not run:
library("ranger")
titanic_ranger_model <- ranger(survived~., data = titanic_imputed, num.trees = 50,
                                probability = TRUE)
explainer_ranger <- explain(titanic_ranger_model, data = titanic_imputed[, -8],
                             y = titanic_imputed$survived)
mp_ranger <- model_performance(explainer_ranger)
plot(mp_ranger)
plot(mp_ranger, geom = "boxplot", show_outliers = 1)
titanic_ranger_model2 <- ranger(survived~gender + fare, data = titanic_imputed,
                                num.trees = 50, probability = TRUE)
explainer_ranger2 <- explain(titanic_ranger_model2, data = titanic_imputed[, -8],
                            y = titanic_imputed$survived,
                            label = "ranger2")
mp_ranger2 <- model_performance(explainer_ranger2)
plot(mp_ranger, mp_ranger2, geom = "roc")
plot(mp_ranger, mp_ranger2, geom = "lift")
plot(mp_ranger, mp_ranger2, geom = "gain")
plot(mp_ranger, mp_ranger2, geom = "boxplot")
plot(mp_ranger, mp_ranger2, geom = "histogram")
```
plot(mp_ranger, mp_ranger2, geom = "ecdf")

```r
titanic_glm_model <- glm(survived~., data = titanic_imputed, family = "binomial")
explainer_glm <- explain(titanic_glm_model, data = titanic_imputed[,-8],
                        y = titanic_imputed$survived, label = "glm",
                        predict_function = function(m,x) predict.glm(m,x,type = "response"))
mp_glm <- model_performance(explainer_glm)
plot(mp_glm)
```

```r
titanic_lm_model <- lm(survived~., data = titanic_imputed)
explainer_lm <- explain(titanic_lm_model, data = titanic_imputed[,-8],
                        y = titanic_imputed$survived, label = "lm")
mp_lm <- model_performance(explainer_lm)
plot(mp_lm)
```

```r
plot(mp_ranger, mp_glm, mp_lm)
plot(mp_ranger, mp_glm, mp_lm, geom = "boxplot")
plot(mp_ranger, mp_glm, mp_lm, geom = "boxplot", show_outliers = 1)
```

## End(Not run)

---

**plot.model_profile**  
*Plot Dataset Level Model Profile Explanations*

### Description

Plot Dataset Level Model Profile Explanations

### Usage

```r
## S3 method for class 'model_profile'
plot(x, ..., geom = "aggregates")
```

### Arguments

- `x`  
a variable profile explanation, created with the `model_profile` function
- `...`  
other parameters
- `geom`  
either "aggregates", "profiles", "points" determines which will be plotted

### Value

An object of the class ggplot.
Examples

titanic_glm_model <- glm(survived~., data = titanic_imputed, family = "binomial")
explainer_glm <- explain(titanic_glm_model, data = titanic_imputed)
expl_glm <- model_profile(explainer_glm, "fare")
plot(expl_glm)

## Not run:
library("ranger")
titanic_ranger_model <- ranger(survived~., data = titanic_imputed, num.trees = 50,
                              probability = TRUE)
explainer_ranger <- explain(titanic_ranger_model, data = titanic_imputed)
expl_ranger <- model_profile(explainer_ranger)
plot(expl_ranger)
plot(expl_ranger, geom = "aggregates")

vp_ra <- model_profile(explainer_ranger, type = "partial", variables = c("age", "fare"))
plot(vp_ra, variables = c("age", "fare"), geom = "points")

vp_ra <- model_profile(explainer_ranger, type = "partial", k = 3)
plot(vp_ra, geom = "profiles")
plot(vp_ra, geom = "points")

vp_ra <- model_profile(explainer_ranger, type = "partial", groups = "gender")
plot(vp_ra, geom = "profiles")
plot(vp_ra, geom = "points")

vp_ra <- model_profile(explainer_ranger, type = "accumulated")
plot(vp_ra, geom = "profiles")
plot(vp_ra, geom = "points")

## End(Not run)
plot.predict_profile

Description
Plot Variable Profile Explanations

Usage
## S3 method for class 'predict_profile'
plot(x, ...)

Arguments
x
an object with instance level residual diagnostics created with predict_diagnostics function
...
other parameters

Value
an ggplot2 object of the class gg.

Examples
library("ranger")
titanic_glm_model <- ranger(survived ~ gender + age + class + fare + sibsp + parch,
data = titanic_imputed)
explainer_glm <- explain(titanic_glm_model,
data = titanic_imputed,
y = titanic_imputed$survived)
johny_d <- titanic_imputed[24, c("gender", "age", "class", "fare", "sibsp", "parch")]

## Not run:
pl <- predict_diagnostics(explainer_glm, johny_d, variables = NULL)
plot(pl)
pl <- predict_diagnostics(explainer_glm, johny_d,
neighbors = 10,
variables = c("age", "fare"))
plot(pl)
pl <- predict_diagnostics(explainer_glm, johny_d,
neighbors = 10,
variables = c("class", "gender"))
plot(pl)
## End(Not run)
predict.explainer

Arguments

- x: a model to be explained, preprocessed by the `explain` function
- ...: other parameters

Value

An object of the class `variable_profile`.

Description

This is a generic `predict()` function works for explainer objects.

Usage

```r
## S3 method for class 'explainer'
predict(object, newdata, ...)

model_prediction(explainer, new_data, ...)
```

Arguments

- object: a model to be explained, object of the class explainer
- newdata: data.frame or matrix - observations for prediction
- ...: other parameters that will be passed to the predict function
- explainer: a model to be explained, object of the class explainer
- new_data: data.frame or matrix - observations for prediction

Value

An numeric matrix of predictions

Examples

```r
HR_glm_model <- glm(status == "fired"~., data = HR, family = "binomial")
explainer_glm <- explain(HR_glm_model, data = HR)
predict(explainer_glm, HR[1:3,])

## Not run:
library("ranger")
HR_ranger_model <- ranger(status~., data = HR, num.trees = 50, probability = TRUE)
explainer_ranger <- explain(HR_ranger_model, data = HR)
predict(explainer_ranger, HR[1:3,])

model_prediction(explainer_ranger, HR[1:3,])

## End(Not run)
```
predict_diagnostics  

Instance Level Residual Diagnostics

Description

This function performs local diagnostic of residuals. For a single instance its neighbors are identified in the validation data. Residuals are calculated for neighbors and plotted against residuals for all data. Find information how to use this function here: https://pbiecek.github.io/ema/localDiagnostics.html.

Usage

predict_diagnostics(
  explainer,
  new_observation,
  variables = NULL,
  ..., nbins = 20,
  neighbors = 50,
  distance = gower::gower_dist
)

individual_diagnostics(
  explainer,
  new_observation,
  variables = NULL,
  ..., nbins = 20,
  neighbors = 50,
  distance = gower::gower_dist
)

Arguments

explainer  a model to be explained, preprocessed by the 'explain' function
new_observation  a new observation for which predictions need to be explained
variables  character - name of variables to be explained
...  other parameters
nbins  number of bins for the histogram. By default 20
neighbors  number of neighbors for histogram. By default 50.
distance  the distance function, by default the gower_dist() function.

Value

An object of the class 'predict_diagnostics'. It’s a data frame with calculated distribution of residuals.
References

Examples

```r
library("ranger")
titanic_glm_model <- ranger(survived ~ gender + age + class + fare + sibsp + parch, data = titanic_imputed)
explainer_glm <- explain(titanic_glm_model,
data = titanic_imputed,
                        y = titanic_imputed$survived)
johny_d <- titanic_imputed[24, c("gender", "age", "class", "fare", "sibsp", "parch")]

## Not run:
id_johny <- predict_diagnostics(explainer_glm, johny_d, variables = NULL)
id_johny
plot(id_johny)

id_johny <- predict_diagnostics(explainer_glm, johny_d, neighbors = 10, variables = c("age", "fare"))
id_johny
plot(id_johny)

id_johny <- predict_diagnostics(explainer_glm, johny_d, neighbors = 10, variables = c("class", "gender"))
id_johny
plot(id_johny)

## End(Not run)
```

**predict_parts**  
*Instance Level Parts of the Model Predictions*

**Description**
Instance Level Variable Attributions as Break Down or SHAP Explanations. Model prediction is decomposed into parts that are attributed for particular variables. From DALEX version 1.0 this function calls the `break_down` or `shap` functions from the iBreakDown package. Find information how to use this function here: https://pbiecek.github.io/ema/breakDown.html.

**Usage**

`predict_parts(explainer, new Observation, ..., type = "break_down")`
predict_parts(oscillations(explainer, new_observation, ...)
predict_parts_break_down(explainer, new_observation, ...)
predict_parts_break_down_interactions(explainer, new_observation, ...)
predict_parts_shap(explainer, new_observation, ...)
variable_attribution(explainer, new_observation, ..., type = "break_down")

Arguments

explainer  a model to be explained, preprocessed by the ‘explain’ function
new_observation  a new observation for which predictions need to be explained
...  other parameters that will be passed to iBreakDown::break_down
type  the type of variable attributions. Either shap, oscillations, break_down or break_down_interactions.

Value

Depending on the type there are different classes of the resulting object. It’s a data frame with calculated average response.

References


Examples

new_dragon <- data.frame(year_of_birth = 200,
                          height = 80,
                          weight = 12.5,
                          scars = 0,
                          number_of_lost_teeth = 5)
dragon_lm_model4 <- lm(life_length ~ year_of_birth + height +
                        weight + scars + number_of_lost_teeth,
                        data = dragons)
dragon_lm_explainer4 <- explain(dragon_lm_model4, data = dragons, y = dragons$year_of_birth,
                                 label = "model_4v")
dragon_lm_predict4 <- predict_parts_break_down(dragon_lm_explainer4,
                                               new_observation = new_dragon)
head(dragon_lm_predict4)
plot(dragon_lm_predict4)

## Not run:
library("ranger")
dragon_ranger_model4 <- ranger(life_length ~ year_of_birth + height +
predict_profile

weight + scars + number_of_lost_teeth,
  data = dragons, num.trees = 50)
dragon_ranger_explainer4 <- explain(dragon_ranger_model4, data = dragons, y = dragons$year_of_birth,
  label = "model_ranger")
dragon_ranger_predict4 <- predict_parts_break_down(dragon_ranger_explainer4,
  new_observation = new_dragon)
head(dragon_ranger_predict4)
plot(dragon_ranger_predict4)

## End(Not run)

predict_profile  Instance Level Profile as Ceteris Paribus

Description
This function calculates individual profiles aka Ceteris Paribus Profiles. From DALEX version 1.0 this function calls the ceteris_paribus from the ingredients package. Find information how to use this function here: https://pbiecek.github.io/ema/ceterisParibus.html.

Usage

predict_profile(
  explainer,
  new_observation,
  variables = NULL,
  ...
)

individual_profile(
  explainer,
  new_observation,
  variables = NULL,
  ...
)

Arguments

explainer  a model to be explained, preprocessed by the explain function
new_observation  a new observation for which predictions need to be explained
variables  character - names of variables to be explained
...  other parameters
type  character, currently only the ceteris_paribus is implemented
Value

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

References


Examples

```r
new_dragon <- data.frame(year_of_birth = 200,
                          height = 80,
                          weight = 12.5,
                          scars = 0,
                          number_of_lost_teeth = 5)

dragon_lm_model4 <- lm(life_length ~ year_of_birth + height +
                       weight + scars + number_of_lost_teeth,
                       data = dragons)
dragon_lm_explainer4 <- explain(dragon_lm_model4, data = dragons, y = dragons$year_of_birth,
                       label = "model_4v")
dragon_lm_predict4 <- predict_profile(dragon_lm_explainer4,
                       new_observation = new_dragon,
                       variables = c("year_of_birth", "height", "scars"))
head(dragon_lm_predict4)
plot(dragon_lm_predict4,
     variables = c("year_of_birth", "height", "scars"))

## Not run:
library("ranger")
dragon_ranger_model4 <- ranger(life_length ~ year_of_birth + height +
                                weight + scars + number_of_lost_teeth,
                                data = dragons, num.trees = 50)
dragon_ranger_explainer4 <- explain(dragon_ranger_model4, data = dragons, y = dragons$year_of_birth,
                                label = "model_ranger")
dragon_ranger_predict4 <- predict_profile(dragon_ranger_explainer4,
                                new_observation = new_dragon,
                                variables = c("year_of_birth", "height", "scars"))
head(dragon_ranger_predict4)
plot(dragon_ranger_predict4,
     variables = c("year_of_birth", "height", "scars"))

## End(Not run)
```
Description

Generic function

Usage

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

Arguments

  x an individual explainer produced with the `describe()` function
  ... other arguments

print.explainer Print Explainer Summary

Description

Print Explainer Summary

Usage

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

Arguments

  x a model explainer created with the `explain` function
  ... other parameters

Examples

aps_lm_model4 <- lm(m2.price~., data = apartments)
aps_lm_explainer4 <- explain(aps_lm_model4, data = apartments, y = apartments$m2.price,
                               label = "model_4v")
aps_lm_explainer4

## Not run:
library("ranger")
titanic_ranger_model <- ranger(survived~., data = titanic_imputed, num.trees = 50,
                                probability = TRUE)
explainer_ranger <- explain(titanic_ranger_model, data = titanic_imputed[, -8],
                            y = titanic_imputed$survived,
                            label = "model_ranger")
explainer_ranger

## End(Not run)
print.model_diagnostics

Print Dataset Level Model Diagnostics

Description

Generic function

Usage

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

Arguments

x 
- an object with dataset level residual diagnostics created with `model_diagnostics` function

... 
- other parameters

print.model_info

Print model_info

Description

Function prints object of class `model_info` created with `model_info`

Usage

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

Arguments

x 
- an object of class `model_info`

... 
- other parameters
print.model_performance

Print Dataset Level Model Performance Summary

Description
Print Dataset Level Model Performance Summary

Usage

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

Arguments

x            a model to be explained, object of the class 'model_performance_explainer'
...           other parameters

Examples

## Not run:
library("ranger")
titanic_ranger_model <- ranger(survived~., data = titanic_imputed, num.trees = 100,
                               probability = TRUE)
# It's a good practice to pass data without target variable
explainer_ranger <- explain(titanic_ranger_model, data = titanic_imputed[, -8],
                             y = titanic_imputed$survived)
# resulting dataframe has predicted values and residuals
mp_ex_rn <- model_performance(explainer_ranger)
mp_ex_rn
plot(mp_ex_rn)

## End(Not run)

print.model_profile

Print Dataset Level Model Profile

Description
Generic function

Usage

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


Arguments

x an object with dataset level profile created with `model_profile` function

... other parameters

---

`print.predict_diagnostics`

*Print Instance Level Residual Diagnostics*

---

Description

Generic function

Usage

## S3 method for class 'predict_diagnostics'

```r
print(x, ...)
```

Arguments

x an object with instance level residual diagnostics created with `predict_diagnostics` function

... other parameters

---

`theme_drwhy`

*DrWhy Theme for ggplot objects*

---

Description

DrWhy Theme for ggplot objects

Usage

```r
theme_drwhy()
theme_drwhy_vertical()
```

Value

theme for ggplot2 objects
**titanic**

---

**Passengers and Crew on the RMS Titanic Data**

**Description**

The `titanic` data is a complete list of passengers and crew members on the RMS Titanic. It includes a variable indicating whether a person did survive the sinking of the RMS Titanic on April 15, 1912.

**Usage**

```r
data(titanic)
data(titanic_imputed)
```

**Format**

a data frame with 2207 rows and 9 columns

**Details**

This dataset was copied from the `stablelearner` package and went through few variable transformations. Levels in `embarked` was replaced with full names, `sibsp`, `parch` and `fare` were converted to numerical variables and values for crew were replaced with 0. If you use this dataset please cite the original package.

From `stablelearner`: The website [https://www.encyclopedia-titanica.org](https://www.encyclopedia-titanica.org) offers detailed information about passengers and crew members on the RMS Titanic. According to the website 1317 passengers and 890 crew member were abord. 8 musicians and 9 employees of the shipyard company are listed as passengers, but travelled with a free ticket, which is why they have `NA` values in `fare`. In addition to that, `fare` is truely missing for a few regular passengers.

- gender a factor with levels `male` and `female`.
- age a numeric value with the persons age on the day of the sinking.
- class a factor specifying the class for passengers or the type of service aboard for crew members.
- embarked a factor with the persons place of of embarkment (Belfast/Cherbourg/Queenstown/Southampton).
- country a factor with the persons home country.
- fare a numeric value with the ticket price (0 for crew members, musicians and employees of the shipyard company).
- `sibsp` an ordered factor specifying the number if siblings/spouses aboard; adopted from Vanderbild data set (see below).
- `parch` an ordered factor specifying the number of parents/children aboard; adopted from Vanderbild data set (see below).
- survived a factor with two levels (`no` and `yes`) specifying whether the person has survived the sinking.
NOTE: The titanic_imputed dataset uses the following imputation rules.

- Missing ‘age’ is replaced with the mean of the observed ones, i.e., 30.
- For sibsp and parch, missing values are replaced by the most frequently observed value, i.e., 0.
- For fare, mean fare for a given class is used, i.e., 0 pounds for crew, 89 pounds for the 1st, 22 pounds for the 2nd, and 13 pounds for the 3rd class.

Source

This dataset was copied from the stablelearner package and went through few variable transformations. The complete list of persons on the RMS titanic was downloaded from https://www.encyclopedia-titanica.org on April 5, 2016. The information given in sibsp and parch was adopted from a data set obtained from http://biostat.mc.vanderbilt.edu/DataSets.

References


update_data

Update data of an explainer object

Description

Function allows users to update data and y of any explainer in a unified way. It doesn’t require knowledge about structure of an explainer.

Usage

update_data(explainer, data, y = NULL, verbose = TRUE)

Arguments

explainer - explainer object that is supposed to be updated.
data - new data, is going to be passed to an explainer
y - new y, is going to be passed to an explainer
verbose - logical, indicates if information about update should be printed

Value

updated explainer object

Examples

aps_lm_model4 <- lm(m2.price ~ ., data = apartments)
apls_lm_explainer4 <- explain(aps_lm_model4, data = apartments, label = "model_4v")
explainer <- update_data(aps_lm_explainer4, data = apartmentsTest, y = apartmentsTest$m2.price)
**update_label**  
*Update label of explainer object*

**Description**  
Function allows users to update label of any explainer in a unified way. It doesn’t require knowledge about structure of an explainer.

**Usage**  
`update_label(explainer, label, verbose = TRUE)`

**Arguments**  
- `explainer`: explainer object that is supposed to be updated.
- `label`: new label, is going to be passed to an explainer
- `verbose`: logical, indicates if information about update should be printed

**Value**  
updated explainer object

**Examples**  
```r  
aps_lm_model4 <- lm(m2.price ~ ., data = apartments)  
aps_lm_explainer4 <- explain(aps_lm_model4, data = apartments, label = "model_4v")  
explaner <- update_label(aps_lm_explainer4, label = "lm")  
```  

---

**variable_effect**  
*Dataset Level Variable Effect as Partial Dependency Profile or Accumulated Local Effects*

**Description**  
From DALEX version 1.0 this function calls the `accumulated_dependency` or `partial_dependency` from the ingredients package. Find information how to use this function here: [https://pbiecek.github.io/ema/partialDependenceProfiles.html](https://pbiecek.github.io/ema/partialDependenceProfiles.html).

**Usage**  
```r  
variable_effect(explainer, variables, ..., type = "partial_dependency")  
variable_effect_partial_dependency(explainer, variables, ...)  
variable_effect_accumulated_dependency(explainer, variables, ...)  
```
Arguments

explainer a model to be explained, preprocessed by the `explain` function
variables character - names of variables to be explained
... other parameters
type character - type of the response to be calculated. Currently following options are implemented: 'partial_dependency' for Partial Dependency and 'accumulated_dependency' for Accumulated Local Effects

Value

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

References


Examples

titanic_glm_model <- glm(survived~., data = titanic_imputed, family = "binomial")
explainer_glm <- explain(titanic_glm_model, data = titanic_imputed)
expl_glm <- variable_effect(explainer_glm, "fare", "partial_dependency")
plot(expl_glm)

## Not run:
library("ranger")
titanic_ranger_model <- ranger(survived~., data = titanic_imputed, num.trees = 50, probability = TRUE)
explainer_ranger <- explain(titanic_ranger_model, data = titanic_imputed)
expl_ranger <- variable_effect(explainer_ranger, variables = "fare", type = "partial_dependency")
plot(expl_ranger)
plot(expl_ranger, expl_glm)

# Example for factor variable (with factorMerger)
expl_ranger_factor <- variable_effect(explainer_ranger, variables = "class")
plot(expl_ranger_factor)

## End(Not run)
**Description**

This function is a wrapper over various predict functions for different models and different model structures. The wrapper returns a single numeric score for each new observation. To do this it uses different extraction techniques for models from different classes, like for classification random forest it forces the output to be probabilities not classes itself.

**Usage**

```r
yhat(X.model, newdata, ...)  
## S3 method for class 'lm'  
yhat(X.model, newdata, ...)  
## S3 method for class 'randomForest'  
yhat(X.model, newdata, ...)  
## S3 method for class 'svm'  
yhat(X.model, newdata, ...)  
## S3 method for class 'gbm'  
yhat(X.model, newdata, ...)  
## S3 method for class 'glm'  
yhat(X.model, newdata, ...)  
## S3 method for class 'cv.glmnet'  
yhat(X.model, newdata, ...)  
## S3 method for class 'glmnet'  
yhat(X.model, newdata, ...)  
## S3 method for class 'ranger'  
yhat(X.model, newdata, ...)  
## S3 method for class 'model_fit'  
yhat(X.model, newdata, ...)  
## S3 method for class 'train'  
yhat(X.model, newdata, ...)  
## S3 method for class 'lrm'  
yhat(X.model, newdata, ...)  
## S3 method for class 'rpart'  
yhat(X.model, newdata, ...)  
## Default S3 method:  
yhat(X.model, newdata, ...)
```
Arguments

- **X.model** object - a model to be explained
- **newdata** data.frame or matrix - observations for prediction
- **...** other parameters that will be passed to the predict function

Details

Currently supported packages are:

- **class cv.glmnet and glmnet** - models created with **glmnet** package,
- **class glm** - generalized linear models created with **glm**, 
- **class model_fit** - models created with **parsnip** package,
- **class lm** - linear models created with **lm**, 
- **class ranger** - models created with **ranger** package,
- **class randomForest** - random forest models created with **randomForest** package,
- **class svm** - support vector machines models created with the **e1071** package,
- **class train** - models created with **caret** package,
- **class gbm** - models created with **gbm** package,
- **class lrm** - models created with **rms** package,
- **class rpart** - models created with **rpart** package.

Value

An numeric matrix of predictions
Index

*Topic HR
  HR, 8

*Topic apartments
  apartments, 3

*Topic dragons
  dragons, 4

*Topic titanic
  titanic, 33

accumulated_dependency, 16, 35
apartments, 3
apartments_test (apartments), 3
apartmentsTest (apartments), 3
break_down, 25
ceteris_paribus, 27
colors_breakdown_drwhy
  (colors_discrete_drwhy), 3
colors_discrete_drwhy, 3
colors_diverging_drwhy
  (colors_discrete_drwhy), 3
dragons, 4
dragons_test (dragons), 4
explain, 14, 19, 23
explain (explain.default), 5
explain.default, 5

feature_importance, 13
feature_importance (model_parts), 13
glm, 38
HR, 8
HR_test (HR), 8
HRTest (HR), 8

individual_diagnostics
  (predict_diagnostics), 24
install_dependencies, 9
lm, 38
loss_accuracy (loss_cross_entropy), 9
loss_cross_entropy, 9
loss_one_minus_auc
  (loss_cross_entropy), 9
loss_root_mean_square
  (loss_cross_entropy), 9
loss_sum_of_squares
  (loss_cross_entropy), 9
model_diagnostics, 10, 18, 30
model_info, 11, 30
model_parts, 13
model_performance, 14
model_prediction (predict.explainer), 23
model_profile, 15, 20, 32
partial_dependency, 16, 35
plot_model_diagnostics, 17
plot.model_performance, 18
plot.model_profile, 20
plot.predict_diagnostics, 21
plot.predict_profile, 22
predict.explainer, 23
predict_diagnostics, 22, 24, 32
predict_parts, 25
predict_parts_break_down
  (predict_parts), 25
predict_parts_break_down_interactions
  (predict_parts), 25
predict_parts_ibreak_down
  (predict_parts), 25
predict_parts_oscillations
  (predict_parts), 25
predict_parts_shap (predict_parts), 25
predict_profile, 27
print.description, 28
print.explainer, 29
print.model_diagnostics, 30
print.model_info, 30
print.model_performance, 31
print.model_profile, 31
print.predict_diagnostics, 32

shap, 25
single_variable (model_profile), 15

theme_drwhy, 32
theme_drwhy_vertical (theme_drwhy), 32
titanic, 33
titanic_imputed (titanic), 33

update_data, 34
update_label, 35

variable_attribution (predict_parts), 25
variable_effect, 35
variable_effect_accumulated_dependency (variable_effect), 35
variable_effect_partial_dependency (variable_effect), 35
variable_importance (model_parts), 13
variable_profile (model_profile), 15

yhat, 36