

# Package ‘DALEXtra’

July 29, 2020

**Title** Extension for 'DALEX' Package

**Version** 1.3.2

**Description** Provides wrapper of various machine learning models.

In applied machine learning, there is a strong belief that we need to strike a balance between interpretability and accuracy. However, in field of the interpretable machine learning, there are more and more new ideas for explaining black-box models, that are implemented in 'R'. 'DALEXtra' creates 'DALEX' Biecek (2018) <arXiv:1806.08915> explainer for many type of models including those created using 'python' 'scikit-learn' and 'keras' libraries, and 'java' 'h2o' library. Important part of the package is Champion-Challenger analysis and innovative approach to model performance across subsets of test data presented in Funnel Plot. Third branch of 'DALEXtra' package is aspect importance analysis that provides instance-level explanations for the groups of explanatory variables.

**Depends** R (>= 3.5.0), DALEX (>= 1.3)

**License** GPL

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.0

**Imports** reticulate, ggplot2, glmnet, ggdendro, gridExtra

**Suggests** auditor, ingredients, gbm, ggrepel, h2o, mlr, mlr3, randomForest, rmarkdown, rpart, xgboost, testthat, covr

**URL** <https://ModelOriented.github.io/DALEXtra/>,  
<https://github.com/ModelOriented/DALEXtra>

**BugReports** <https://github.com/ModelOriented/DALEXtra/issues>

**NeedsCompilation** no

**Author** Szymon Maksymiuk [aut, cre] (<<https://orcid.org/0000-0002-3120-1601>>),  
Przemysław Biecek [aut] (<<https://orcid.org/0000-0001-8423-1823>>),  
Katarzyna Pekala [aut],

Anna Kozak [ctb],  
Hubert Baniecki [ctb]

**Maintainer** Szymon Maksymiuk <sz.maksymiuk@gmail.com>

**Repository** CRAN

**Date/Publication** 2020-07-28 23:42:12 UTC

## R topics documented:

aspect_importance . . . . .	3
aspect_importance_single . . . . .	5
champion_challenger . . . . .	7
create_env . . . . .	8
explain_h2o . . . . .	9
explain_keras . . . . .	11
explain_mlr . . . . .	14
explain_mlr3 . . . . .	16
explain_scikitlearn . . . . .	18
explain_xgboost . . . . .	21
funnel_measure . . . . .	23
get_sample . . . . .	25
group_variables . . . . .	26
model_info.WrappedModel . . . . .	26
overall_comparison . . . . .	28
plot.aspect_importance . . . . .	29
plot.funnel_measure . . . . .	30
plot.overall_comparison . . . . .	31
plot.training_test_comparison . . . . .	33
plot_aspects_importance_grouping . . . . .	34
plot_group_variables . . . . .	35
print.funnel_measure . . . . .	36
print.overall_comparison . . . . .	37
print.scikitlearn_set . . . . .	38
print.training_test_comparison . . . . .	39
training_test_comparison . . . . .	40
triplet . . . . .	41
yhat.WrappedModel . . . . .	43
<b>Index</b>	<b>45</b>

---

aspect_importance	<i>Calculates the feature groups importance (called aspects importance) for a selected observation</i>
-------------------	--

---

## Description

Aspect Importance function takes a sample from a given dataset and modifies it. Modification is made by replacing part of its aspects by values from the observation. Then function is calculating the difference between the prediction made on modified sample and the original sample. Finally, it measures the impact of aspects on the change of prediction by using the linear model or lasso.

## Usage

```
aspect_importance(x, ...)  
  
## S3 method for class 'explainer'  
aspect_importance(  
  x,  
  new_observation,  
  aspects,  
  N = 100,  
  sample_method = "default",  
  n_var = 0,  
  f = 2,  
  show_cor = FALSE,  
  ...  
)  
  
## Default S3 method:  
aspect_importance(  
  x,  
  data,  
  predict_function = predict,  
  new_observation,  
  aspects,  
  N = 100,  
  label = class(x)[1],  
  sample_method = "default",  
  n_var = 0,  
  f = 2,  
  show_cor = FALSE,  
  ...  
)  
  
lime(x, ...)
```

**Arguments**

x	an explainer created with the <code>DALEX::explain()</code> function or a model to be explained.
...	other parameters
new_observation	selected observation with columns that corresponds to variables used in the model
aspects	list containing grouping of features into aspects
N	number of observations to be sampled (with replacement) from data
sample_method	sampling method in <code>get_sample</code>
n_var	maximum number of non-zero coefficients after lasso fitting, if zero than linear regression is used
f	frequency in <code>get_sample</code>
show_cor	show if all features in aspect are pairwise positively correlated, works only if dataset contains solely numeric values
data	dataset, it will be extracted from x if it's an explainer NOTE: It is best when target variable is not present in the data
predict_function	predict function, it will be extracted from x if it's an explainer
label	name of the model. By default it's extracted from the 'class' attribute of the model.

**Value**

An object of the class `aspect_importance`. Contains dataframe that describes aspects' importance.

**Examples**

```
library("DALEX")

model_titanic_glm <- glm(survived == 1 ~
  class+gender+age+sibsp+parch+fare+embarked,
  data = titanic_imputed,
  family = "binomial")

explain_titanic_glm <- explain(model_titanic_glm,
  data = titanic_imputed[,-8],
  y = titanic_imputed$survived == 1,
  verbose = FALSE)

aspects <- list(wealth = c("class", "fare"),
  family = c("sibsp", "parch"),
  personal = c("gender", "age"),
  embarked = "embarked")

aspect_importance(explain_titanic_glm,
  new_observation = titanic_imputed[1,],
```

```

        aspects = aspects)

library("randomForest")
model_titanic_rf <- randomForest(survived ~ class + gender + age + sibsp +
                                parch + fare + embarked,
                                data = titanic_imputed)

explain_titanic_rf <- explain(model_titanic_rf,
                              data = titanic_imputed[,-8],
                              y = titanic_imputed$survived == 1,
                              verbose = FALSE)

aspect_importance(explain_titanic_rf,
                  new_observation = titanic_imputed[1,],
                  aspects = aspects)

```

---

aspect\_importance\_single

*Aspects importance for single aspects*

---

## Description

Calculates `aspect_importance` for single aspects (every aspect contains only one feature).

## Usage

```

aspect_importance_single(x, ...)

## S3 method for class 'explainer'
aspect_importance_single(
  x,
  new_observation,
  N = 100,
  sample_method = "default",
  n_var = 0,
  f = 2,
  ...
)

## Default S3 method:
aspect_importance_single(
  x,
  data,
  predict_function = predict,

```

```

new_observation,
N = 100,
label = class(x)[1],
sample_method = "default",
n_var = 0,
f = 2,
...
)

```

### Arguments

x	an explainer created with the <code>DALEX::explain()</code> function or a model to be explained.
...	other parameters
new_observation	selected observation with columns that corresponds to variables used in the model, should be without target variable
N	number of observations to be sampled (with replacement) from data
sample_method	sampling method in <a href="#">get_sample</a>
n_var	how many non-zero coefficients for lasso fitting, if zero than linear regression is used
f	frequency in <a href="#">get_sample</a>
data	dataset, it will be extracted from x if it's an explainer NOTE: Target variable shouldn't be present in the data
predict_function	predict function, it will be extracted from x if it's an explainer
label	name of the model. By default it's extracted from the 'class' attribute of the model.

### Value

An object of the class 'aspect\_importance'. Contains dataframe that describes aspects' importance.

### Examples

```

library("DALEX")

model_titanic_glm <- glm(survived == 1 ~ class + gender + age +
  sibsp + parch + fare + embarked,
  data = titanic_imputed,
  family = "binomial")

aspect_importance_single(model_titanic_glm, data = titanic_imputed[, -8],
  new_observation = titanic_imputed[1, -8])

```

---

champion\_challenger     *Compare machine learning models*

---

## Description

Determining if one model is better than the other one is a difficult task. Mostly because there is a lot of fields that have to be covered to make such a judgement. Overall performance, performance on the crucial subset, distribution of residuals, those are only few among many ideas related to that issue. Following function allow user to create a report based on various sections. Each says something different about relation between champion and challengers. DALEXtra package share 3 base sections which are [funnel\\_measure](#) [overall\\_comparison](#) and [training\\_test\\_comparison](#) but any object that has generic plot function can be included at report.

## Usage

```
champion_challenger(
  sections,
  dot_size = 4,
  output_dir_path = getwd(),
  output_name = "Report",
  model_performance_table = FALSE,
  title = "ChampionChallenger",
  author = Sys.info()[["user"]],
  ...
)
```

## Arguments

sections	- list of sections to be attached to report. Could be sections available with DALEXtra which are <a href="#">funnel_measure</a> <a href="#">training_test_comparison</a> , <a href="#">overall_comparison</a> or any other explanation that can work with plot function. Please provide name for not standard sections, that will be presented as section titles. Otherwise class of the object will be used.
dot_size	- dot_size argument passed to <a href="#">plot.funnel_measure</a> if <a href="#">funnel_measure</a> section present
output_dir_path	- path to directory where Report should be created. By default it is current working directory.
output_name	- name of the Report. By default it is "Report"
model_performance_table	- If TRUE and <a href="#">overall_comparison</a> section present, table of scores will be displayed.
title	- Title for report, by default it is "ChampionChallenger".
author	- Author of , report. By default it is current user name.
...	- other parameters passed to <code>rmarkdown::render</code> .

**Value**

rmarkdown report

**Examples**

```
library("mlr")
library("DALEXtra")
task <- mlr::makeRegrTask(
  id = "R",
  data = apartments,
  target = "m2.price"
)
learner_lm <- mlr::makeLearner(
  "regr.lm"
)
model_lm <- mlr::train(learner_lm, task)
explainer_lm <- explain_mlr(model_lm, apartmentsTest, apartmentsTest$m2.price, label = "LM")

learner_rf <- mlr::makeLearner(
  "regr.randomForest"
)
model_rf <- mlr::train(learner_rf, task)
explainer_rf <- explain_mlr(model_rf, apartmentsTest, apartmentsTest$m2.price, label = "RF")

learner_gbm <- mlr::makeLearner(
  "regr.gbm"
)
model_gbm <- mlr::train(learner_gbm, task)
explainer_gbm <- explain_mlr(model_gbm, apartmentsTest, apartmentsTest$m2.price, label = "GBM")

plot_data <- funnel_measure(explainer_lm, list(explainer_rf, explainer_gbm),
  nbins = 5, measure_function = DALEX::loss_root_mean_square)

champion_challenger(list(plot_data), dot_size = 3)
```

---

create\_env

*Create your conda virtual env with DALEX*

---

**Description**

Python objects may be loaded into R. However, it requires versions of the Python and libraries to match between both machines. This functions allow user to create conda virtual environment based on provided .yml file.



**Usage**

```
create_env(yml, condaenv)
```

**Arguments**

`yml` a path to the .yml file. If OS is Windows conda has to be added to the PATH first  
`condaenv` path to main conda folder. If OS is Unix You may want to specify it. When passed with windows, param will be omitted.

**Value**

Name of created virtual env.

**Author(s)**

Szymon Maksymiuk

**Examples**

```
## Not run:  
create_env(system.file("extdata", "testing_environment.yml", package = "DALEXtra"))  
  
## End(Not run)
```

---

explain\_h2o

*Create explainer from your h2o model*

---

**Description**

DALEX is designed to work with various black-box models like tree ensembles, linear models, neural networks etc. Unfortunately R packages that create such models are very inconsistent. Different tools use different interfaces to train, validate and use models. One of those tools, we would like to make more accessible is H2O.

**Usage**

```
explain_h2o(  
  model,  
  data = NULL,  
  y = NULL,  
  weights = NULL,  
  predict_function = NULL,  
  residual_function = 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 y argument). NOTE: If target variable is present in the data, some of the functionalities may not work properly.
y	numeric vector with outputs / scores. If provided then it shall have the same size as data
weights	numeric vector with sampling weights. By default it's NULL. If provided then it shall have the same length as data
predict_function	function that takes two arguments: model and new data and returns numeric vector with predictions
residual_function	function that takes three arguments: model, data and response vector y. It should return a numeric vector with model residuals for given data. If not provided, response residuals ( $y - \hat{y}$ ) are calculated.
...	other parameters
label	character - the name of the model. By default it's extracted from the 'class' attribute of the model
verbose	if TRUE (default) then diagnostic messages will be printed
precalculate	if TRUE (default) then 'predicted_values' and 'residuals' are calculated when explainer is created.
colorize	if TRUE (default) then WARNINGS, ERRORS and NOTES are colorized. Will work only in the R console.
model_info	a named list (package, version, type) containing information about model. If NULL, DALEX will seek for information on its own.
type	type of a model, either classification or regression. If not specified then type will be extracted from model_info.

### Value

explainer object ([explain](#)) ready to work with DALEX

### Examples

```
# # load packages and data
```

```
library(h2o)
library(DALEXtra)

# data <- DALEX::titanic_imputed

# init h2o
h2o.init()

# stop h2o progress printing
h2o.no_progress()

# split the data
# h2o_split <- h2o.splitFrame(as.h2o(data))
# train <- h2o_split[[1]]
# test <- as.data.frame(h2o_split[[2]])
# h2o automl takes target as factor
# train$survived <- as.factor(train$survived)

# fit a model
# automl <- h2o.automl(y = "survived",
#                     training_frame = train,
#                     max_runtime_secs = 30)

# create an explainer for the model
# explainer <- explain_h2o(automl,
#                          data = test,
#                          y = test$survived,
#                          label = "h2o")

titanic_test <- read.csv(system.file("extdata", "titanic_test.csv", package = "DALEXtra"))
titanic_train <- read.csv(system.file("extdata", "titanic_train.csv", package = "DALEXtra"))
titanic_h2o <- h2o::as.h2o(titanic_train)
titanic_h2o["survived"] <- h2o::as.factor(titanic_h2o["survived"])
titanic_test_h2o <- h2o::as.h2o(titanic_test)
model <- h2o::h2o.gbm(
  training_frame = titanic_h2o,
  y = "survived",
  distribution = "bernoulli",
  ntrees = 500,
  max_depth = 4,
  min_rows = 12,
  learn_rate = 0.001
)
explain_h2o(model, titanic_test[,1:17], titanic_test[,18])

h2o.shutdown(prompt = FALSE)
```

## Description

Keras models may be loaded into R environment like any other Python object. This function helps to inspect performance of Python model and compare it with other models, using R tools like DALEX. This function creates an object that is easily accessible R version of Keras model exported from Python via pickle file.

## Usage

```
explain_keras(
  path,
  yml = NULL,
  condaenv = NULL,
  env = NULL,
  data = NULL,
  y = NULL,
  weights = NULL,
  predict_function = NULL,
  residual_function = NULL,
  ...,
  label = NULL,
  verbose = TRUE,
  precalculate = TRUE,
  colorize = TRUE,
  model_info = NULL,
  type = NULL
)
```

## Arguments

path	a path to the pickle file. Can be used without other arguments if you are sure that active Python version match pickle version.
yml	a path to the yml file. Conda virtual env will be recreated from this file. If OS is Windows conda has to be added to the PATH first
condaenv	If yml param is provided, a path to the main conda folder. If yml is null, a name of existing conda environment.
env	A path to python virtual environment.
data	test data set that will be passed to <a href="#">explain</a> .
y	vector that will be passed to <a href="#">explain</a> .
weights	numeric vector with sampling weights. By default it's NULL. If provided then it shall have the same length as data
predict_function	predict function that will be passed into <a href="#">explain</a> . If NULL, default will be used.
residual_function	residual function that will be passed into <a href="#">explain</a> . If NULL, default will be used.
...	other parameters

label	label that will be passed into <code>explain</code> . If NULL, default will be used.
verbose	bool that will be passed into <code>explain</code> . If NULL, default will be used.
precalculate	if TRUE (default) then 'predicted_values' and 'residuals' are calculated when explainer is created.
colorize	if TRUE (default) then WARNINGS, ERRORS and NOTES are colorized. Will work only in the R console.
model_info	a named list (package, version, type) containg information about model. If NULL, DALEX will seek for information on it's own.
type	type of a model, either classification or regression. If not specified then type will be extracted from model_info.

### Value

An object of the class 'explainer'.

**Example of Python code avialble at documentation** [explain\\_scikitlearn](#)

### Errors use case

Here is shortened version of solution for specific errors

#### There already exists environment with a name specified by given .yaml file

If you provide .yaml file that in its header contatins name exact to name of environment that already exists, existing will be set active without changing it.

You have two ways of solving that issue. Both connected with anaconda prompt. First is removing conda env with command:

```
conda env remove --name myenv
```

And execute function once again. Second is updating env via:

```
conda env create -f environment.yaml
```

#### Conda cannot find specified packages at channels you have provided.

That error may be casued by a lot of things. One of those is that specified version is too old to be avialble from offcial conda repo. Edit Your .yaml file and add link to proper repository at channels section.

Issue may be also connected with the platform. If model was created on the platform with dif-ferent OS yo may need to remove specific version from .yaml file.

```
-numpy=1.16.4=py36h19fb1c0_0
```

```
-numpy-base=1.16.4=py36hc3f5095_0
```

In the example above You have to remove =py36h19fb1c0\_0 and =py36hc3f5095\_0

If some packages are not availbe for anaconda at all, use pip statement

If .yaml file seems not to work, virtual env can be created manually using anaconda prompt.

```
conda create -n name_of_env python=3.4
```

```
conda install -n name_of_env name_of_package=0.20
```

### Author(s)

Szymon Maksymiuk

## Examples

```

library("DALEXtra")
## Not run:
# Explainer build (Keep in mind that 9th column is target)
test_data <-
  read.csv(
    "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv",
    sep = ",")
# Keep in mind that when pickle is being built and loaded,
# not only Python version but libraries versions has to match aswell
explainer <- explain_keras(system.file("extdata", "keras.pkl", package = "DALEXtra"),
  conda = "myenv",
  data = test_data[,1:8], y = test_data[,9])
plot(model_performance(explainer))

# Predictions with newdata
predict(explainer, test_data[1:10,1:8])

## End(Not run)

```

---

explain\_mlr

*Create explainer from your mlr model*

---

## Description

DALEX is designed to work with various black-box models like tree ensembles, linear models, neural networks etc. Unfortunately R packages that create such models are very inconsistent. Different tools use different interfaces to train, validate and use models. One of those tools, which is one of the most popular one is mlr package. We would like to present dedicated explain function for it.

## Usage

```

explain_mlr(
  model,
  data = NULL,
  y = NULL,
  weights = NULL,
  predict_function = NULL,
  residual_function = 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 y argument). NOTE: If target variable is present in the data, some of the functionalities may not work properly.
y	numeric vector with outputs / scores. If provided then it shall have the same size as data.
weights	numeric vector with sampling weights. By default it's NULL. If provided then it shall have the same length as data
predict_function	function that takes two arguments: model and new data and returns numeric vector with predictions
residual_function	function that takes three arguments: model, data and response vector y. It should return a numeric vector with model residuals for given data. If not provided, response residuals ( $y - \hat{y}$ ) are calculated.
...	other parameters
label	character - the name of the model. By default it's extracted from the 'class' attribute of the model
verbose	if TRUE (default) then diagnostic messages will be printed
precalculate	if TRUE (default) then 'predicted_values' and 'residuals' are calculated when explainer is created.
colorize	if TRUE (default) then WARNINGS, ERRORS and NOTES are colorized. Will work only in the R console.
model_info	a named list (package, version, type) containing information about model. If NULL, DALEX will seek for information on its own.
type	type of a model, either classification or regression. If not specified then type will be extracted from model_info.

**Value**

explainer object ([explain](#)) ready to work with DALEX

**Examples**

```
library("DALEXtra")
titanic_test <- read.csv(system.file("extdata", "titanic_test.csv", package = "DALEXtra"))
titanic_train <- read.csv(system.file("extdata", "titanic_train.csv", package = "DALEXtra"))
library("mlr")
task <- mlr::makeClassifTask(
  id = "R",
  data = titanic_train,
  target = "survived"
)
learner <- mlr::makeLearner(
```

```

"classif.gbm",
par.vals = list(
  distribution = "bernoulli",
  n.trees = 500,
  interaction.depth = 4,
  n.minobsinnode = 12,
  shrinkage = 0.001,
  bag.fraction = 0.5,
  train.fraction = 1
),
predict.type = "prob"
)
gbm <- mlr::train(learner, task)
explain_mlr(gbm, titanic_test[,1:17], titanic_test[,18])

```

---

explain\_mlr3

*Create explainer from your mlr model*

---

## Description

DALEX is designed to work with various black-box models like tree ensembles, linear models, neural networks etc. Unfortunately R packages that create such models are very inconsistent. Different tools use different interfaces to train, validate and use models. One of those tools, which is one of the most popular one is mlr3 package. We would like to present dedicated explain function for it.

## Usage

```

explain_mlr3(
  model,
  data = NULL,
  y = NULL,
  weights = NULL,
  predict_function = NULL,
  residual_function = NULL,
  ...,
  label = NULL,
  verbose = TRUE,
  precalculate = TRUE,
  colorize = TRUE,
  model_info = NULL,
  type = NULL
)

```

## Arguments

`model` object - a fitted learned created with mlr3.



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 y argument). NOTE: If target variable is present in the data, some of the functionalities may not work properly.
y	numeric vector with outputs / scores. If provided then it shall have the same size as data
weights	numeric vector with sampling weights. By default it's NULL. If provided then it shall have the same length as data
predict_function	function that takes two arguments: model and new data and returns numeric vector with predictions
residual_function	function that takes three arguments: model, data and response vector y. It should return a numeric vector with model residuals for given data. If not provided, response residuals ( $y - \hat{y}$ ) are calculated.
...	other parameters
label	character - the name of the model. By default it's extracted from the 'class' attribute of the model
verbose	if TRUE (default) then diagnostic messages will be printed.
precalculate	if TRUE (default) then 'predicted_values' and 'residuals' are calculated when explainer is created.
colorize	if TRUE (default) then WARNINGS, ERRORS and NOTES are colorized. Will work only in the R console.
model_info	a named list (package, version, type) containing information about model. If NULL, DALEX will seek for information on its own.
type	type of a model, either classification or regression. If not specified then type will be extracted from model_info.

## Value

explainer object ([explain](#)) ready to work with DALEX

## Examples

```
library("DALEXtra")
library(mlr3)
titanic_imputed$survived <- as.factor(titanic_imputed$survived)
task_classif <- TaskClassif$new(id = "1", backend = titanic_imputed, target = "survived")
learner_classif <- lrn("classif.rpart", predict_type = "prob")
learner_classif$train(task_classif)
explain_mlr3(learner_classif, data = titanic_imputed,
            y = as.numeric(as.character(titanic_imputed$survived)))

task_regr <- TaskRegr$new(id = "2", backend = apartments, target = "m2.price")
learner_regr <- lrn("regr.rpart")
learner_regr$train(task_regr)
```

```
explain_mlr3(learner_regr, data = apartments, apartments$m2.price)
```

---

explain\_scikitlearn    *Wrapper for Python Scikit-Learn Models*

---

## Description

scikit-learn models may be loaded into R environment like any other Python object. This function helps to inspect performance of Python model and compare it with other models, using R tools like DALEX. This function creates an object that is easily accessible R version of scikit-learn model exported from Python via pickle file.

## Usage

```
explain_scikitlearn(
  path,
  yml = NULL,
  condaenv = NULL,
  env = NULL,
  data = NULL,
  y = NULL,
  weights = NULL,
  predict_function = NULL,
  residual_function = NULL,
  ...,
  label = NULL,
  verbose = TRUE,
  precalculate = TRUE,
  colorize = TRUE,
  model_info = NULL,
  type = NULL
)
```

## Arguments

path	a path to the pickle file. Can be used without other arguments if you are sure that active Python version match pickle version.
yml	a path to the yml file. Conda virtual env will be recreated from this file. If OS is Windows conda has to be added to the PATH first
condaenv	If yml param is provided, a path to the main conda folder. If yml is null, a name of existing conda environment.
env	A path to python virtual environment.
data	test data set that will be passed to <a href="#">explain</a> .
y	vector that will be passed to <a href="#">explain</a> .

weights	numeric vector with sampling weights. By default it's NULL. If provided then it shall have the same length as data
predict_function	predict function that will be passed into <code>explain</code> . If NULL, default will be used.
residual_function	residual function that will be passed into <code>explain</code> . If NULL, default will be used.
...	other parameters
label	label that will be passed into <code>explain</code> . If NULL, default will be used.
verbose	bool that will be passed into <code>explain</code> . If NULL, default will be used.
precalculate	if TRUE (default) then 'predicted_values' and 'residuals' are calculated when explainer is created.
colorize	if TRUE (default) then WARNINGS, ERRORS and NOTES are colorized. Will work only in the R console.
model_info	a named list (package, version, type) containg information about model. If NULL, DALEX will seek for information on it's own.
type	type of a model, either classification or regression. If not specified then type will be extracted from model_info.

### Value

An object of the class 'explainer'. It has additional field `param_set` when user can check parameters of scikitlearn model

### Example of Python code

```
from pandas import DataFrame, read_csv
import pandas as pd
import pickle
import sklearn.ensemble
model = sklearn.ensemble.GradientBoostingClassifier()
model = model.fit(titanic_train_X, titanic_train_Y)
pickle.dump(model, open("gbm.pkl", "wb"), protocol = 2)
```

In order to export environment into .yaml, activating virtual env via `activate name_of_the_env` and execution of the following shell command is necessary  
`conda env export > environment.yaml`

### Errors use case

Here is shortened version of solution for specific errors

### There already exists environment with a name specified by given .yaml file

If you provide .yaml file that in its header contains name exact to name of environment that already exists, existing will be set active without changing it.

You have two ways of solving that issue. Both connected with anaconda prompt. First is removing conda env with command:

```
conda env remove --name myenv
```

And execute function once again. Second is updating env via:

```
conda env create -f environment.yml
```

### **Conda cannot find specified packages at channels you have provided.**

That error may be caused by a lot of things. One of those is that specified version is too old to be available from official conda repo. Edit Your .yml file and add link to proper repository at channels section.

Issue may be also connected with the platform. If model was created on the platform with different OS you may need to remove specific version from .yml file.

```
-numpy=1.16.4=py36h19fb1c0_0
```

```
-numpy-base=1.16.4=py36hc3f5095_0
```

In the example above You have to remove =py36h19fb1c0\_0 and =py36hc3f5095\_0

If some packages are not available for anaconda at all, use pip statement

If .yml file seems not to work, virtual env can be created manually using anaconda prompt.

```
conda create -n name_of_env python=3.4
```

```
conda install -n name_of_env name_of_package=0.20
```

### **Author(s)**

Szymon Maksymiuk

### **Examples**

```
## Not run:
# Explainer build (Keep in mind that 18th column is target)
titanic_test <- read.csv(system.file("extdata", "titanic_test.csv", package = "DALEXtra"))
# Keep in mind that when pickle is being built and loaded,
# not only Python version but libraries versions has to match aswell
explainer <- explain_scikitlearn(system.file("extdata", "scikitlearn.pkl", package = "DALEXtra"),
  yml = system.file("extdata", "testing_environment.yml", package = "DALEXtra"),
  data = titanic_test[,1:17], y = titanic_test$survived)
plot(model_performance(explainer))

# Predictions with newdata
predict(explainer, titanic_test[1:10,1:17])

## End(Not run)
```

---

`explain_xgboost`*Create explainer from your xgboost model*

---

## Description

DALEX is designed to work with various black-box models like tree ensembles, linear models, neural networks etc. Unfortunately R packages that create such models are very inconsistent. Different tools use different interfaces to train, validate and use models. One of those tools, we would like to make more accessible is xgboost.

## Usage

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

## Arguments

<code>model</code>	object - a model to be explained
<code>data</code>	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 <code>y</code> argument). NOTE: If target variable is present in the data, some of the functionalities may not work properly.
<code>y</code>	numeric vector with outputs / scores. If provided then it shall have the same size as data. For classif task has to be numeric in range [0, nclasses)
<code>weights</code>	numeric vector with sampling weights. By default it's NULL. If provided then it shall have the same length as data
<code>predict_function</code>	function that takes two arguments: model and new data and returns numeric vector with predictions

residual_function	function that takes three arguments: model, data and response vector y. It should return a numeric vector with model residuals for given data. If not provided, response residuals ( $y - \hat{y}$ ) are calculated.
...	other parameters
label	character - the name of the model. By default it's extracted from the 'class' attribute of the model
verbose	if TRUE (default) then diagnostic messages will be printed
precalculate	if TRUE (default) then 'predicted_values' and 'residuals' are calculated when explainer is created.
colorize	if TRUE (default) then WARNINGS, ERRORS and NOTES are colorized. Will work only in the R console.
model_info	a named list (package, version, type) containing information about model. If NULL, DALEX will seek for information on its own.
type	type of a model, either classification or regression. If not specified then type will be extracted from model_info.
encode_function	function(data, ...) that if executed with data parameters returns encoded dataframe that was used to fit model. Xgboost does not handle factors on its own so such function is needed to acquire better explanations.
true_labels	vector of y before encoding.

## Value

explainer object ([explain](#)) ready to work with DALEX

## Examples

```
library("xgboost")
library("DALEXtra")
library("mlr")
# 8th column is target that has to be omitted in X data
data <- as.matrix(createDummyFeatures(titanic_imputed[,-8]))
model <- xgboost(data, titanic_imputed$survived, nrounds = 10,
  params = list(objective = "binary:logistic"),
  prediction = TRUE)
# explainer with encode function
explainer_1 <- explain_xgboost(model, data = titanic_imputed[,-8],
  titanic_imputed$survived,
  encode_function = function(data) {
  as.matrix(createDummyFeatures(data))
})
plot(predict_parts(explainer_1, titanic_imputed[1,-8]))

# explainer without encode function
explainer_2 <- explain_xgboost(model, data = data, titanic_imputed$survived)
plot(predict_parts(explainer_2, data[1,], drop = FALSE))
```

---

funnel_measure	<i>Caluculate difference in performance in models across different categories</i>
----------------	---

---

### Description

Function `funnel_measure` allows users to compare two models based on their explainers. It partitions dataset on which models were built and creates categories according to quantiles of columns in partition data. `nbins` parameter determinates number of qunatiles. For each category difference in provided measure is being calculated. Positive value of that differnece means that Champion model has better performance in specified category, while negative value means that one of the Challengers was better. Function allows to compare multiple Challengers at once.

### Usage

```
funnel_measure(  
  champion,  
  challengers,  
  measure_function = NULL,  
  nbins = 5,  
  partition_data = champion$data,  
  cutoff = 0.01,  
  cutoff_name = "Other",  
  factor_conversion_threshold = 7,  
  show_info = TRUE,  
  categories = NULL  
)
```

### Arguments

<code>champion</code>	- explainer of champion model.
<code>challengers</code>	- explainer of challenger model or list of explainers.
<code>measure_function</code>	- measure function that calculates performance of model based on true observation and prediction. Order of parameters is important and should be (y, y_hat). The measure calculated by the function should have the property that lower score value indicates better model. If NULL, RMSE will be used for regression, one minus auc for classification and crossentropy for multiclass classification.
<code>nbins</code>	- Number of qunatiles (partition points) for numeric columns. In case when more than one qunatile have the same value, there will be less partition points.
<code>partition_data</code>	- Data by which test dataset will be partitioned for computation. Can be either data.frame or character vector. When second is passed, it has to indicate names of columns that will be extracted fromm test data. By default full test data. If data.frame, number of rows has to be equal to number of rows in test data.
<code>cutoff</code>	- Threshold for categorical data. Entries less frequent than specified value will be merged into one category.

cutoff_name	- Name for new category that arised after merging entries less frequent than cutoff
factor_conversion_threshold	- Numeric columns with lower number of unique values than value of this parameter will be treated as factors
show_info	- Logical value indicating if progress bar should be shown.
categories	- a named list of variable names that will be plotted in a different colour. By deafault it is partitioned on Explanatory, External and Target.

### Value

An object of the class `funnel_measure`

It is a named list containing following fields:

- `data` `data.frame` that consists of columns:
  - Variable Variable according to which partitions were made
  - Measure Difference in measures. Positive value indicates that champion was better, while negative that challenger.
  - Label String that defines subset of Variable values (partition rule).
  - Challenger Label of challenger explainer that was used in Measure
  - Category a category of the variable passed to function
- `models_info` `data.frame` containig inforamtion about models used in analysys

### Examples

```
library("mlr")
library("DALEXtra")
task <- mlr::makeRegrTask(
  id = "R",
  data = apartments,
  target = "m2.price"
)
learner_lm <- mlr::makeLearner(
  "regr.lm"
)
model_lm <- mlr::train(learner_lm, task)
explainer_lm <- explain_mlr(model_lm, apartmentsTest, apartmentsTest$m2.price, label = "LM")

learner_rf <- mlr::makeLearner(
  "regr.randomForest"
)
model_rf <- mlr::train(learner_rf, task)
explainer_rf <- explain_mlr(model_rf, apartmentsTest, apartmentsTest$m2.price, label = "RF")

learner_gbm <- mlr::makeLearner(
  "regr.gbm"
)
model_gbm <- mlr::train(learner_gbm, task)
```



```
explainer_gbm <- explain_mlr(model_gbm, apartmentsTest, apartmentsTest$m2.price, label = "GBM")

plot_data <- funnel_measure(explainer_lm, list(explainer_rf, explainer_gbm),
                           nbins = 5, measure_function = DALEX::loss_root_mean_square)
plot(plot_data)
```

---

get\_sample

*Function for getting binary matrix*

---

## Description

Function creates binary matrix, to be used in aspect\_importance method. It starts with a zero matrix. Then it replaces some zeros with ones. It either randomly replaces one or two zeros per row. Or replace random number of zeros per row - average number of replaced zeros can be controlled by parameter f. Function doesn't allow the returned matrix to have rows with only zeros.

## Usage

```
get_sample(n, p, sample_method = c("default", "binom"), f = 2)
```

## Arguments

n	number of rows
p	number of columns
sample_method	sampling method
f	frequency for binomial sampling

## Value

a binary matrix

## Examples

```
## Not run:
get_sample(100,6,"binom",3)

## End(Not run)
```

---

group_variables	<i>Groups numeric features into aspects</i>
-----------------	---

---

**Description**

Divides correlated features into groups, called aspects. Division is based on correlation cutoff level.

**Usage**

```
group_variables(
  x,
  p = 0.5,
  clust_method = "complete",
  draw_tree = FALSE,
  draw_abline = TRUE
)
```

**Arguments**

x	dataframe with only numeric columns
p	correlation value for cut-off level
clust_method	the agglomeration method to be used, see <a href="#">hclust</a> methods
draw_tree	if TRUE, function plots tree that illustrates grouping
draw_abline	if TRUE, function plots vertical line at cut-off level

**Value**

list of aspects

**Examples**

```
library("DALEX")
dragons_data <- dragons[,c(2,3,4,7,8)]
group_variables(dragons_data, p = 0.7, clust_method = "complete")
```

---

model_info.WrappedModel
-------------------------

*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**

```

## S3 method for class 'WrappedModel'
model_info(model, is_multiclass = FALSE, ...)

## S3 method for class 'H2ORegressionModel'
model_info(model, is_multiclass = FALSE, ...)

## S3 method for class 'H2OBinomialModel'
model_info(model, is_multiclass = FALSE, ...)

## S3 method for class 'H2OMultinomialModel'
model_info(model, is_multiclass = FALSE, ...)

## S3 method for class 'scikitlearn_model'
model_info(model, is_multiclass = FALSE, ...)

## S3 method for class 'keras'
model_info(model, is_multiclass = FALSE, ...)

## S3 method for class 'LearnerRegr'
model_info(model, is_multiclass = FALSE, ...)

## S3 method for class 'LearnerClassif'
model_info(model, is_multiclass = FALSE, ...)

## S3 method for class 'GraphLearner'
model_info(model, is_multiclass = FALSE, ...)

## S3 method for class 'xgb.Booster'
model_info(model, is_multiclass = FALSE, ...)

```

**Arguments**

model	- model object
is_multiclass	- if TRUE and task is classification, then multitask classification is set. Else is omitted. If model_info was executed withing explain function. DALEX will recognize subtype on it's own. @param is_multiclass
...	- another arguments

Currently supported packages are:

- mlr models created with mlr package
- h2o models created with h2o package
- scikit-learn models created with scikit-learn python library and accessed via reticulate
- keras models created with keras python library and accessed via reticulate
- mlr3 models created with mlr3 package
- xgboost models created with xgboost package

**Value**

A named list of class `model_info`

---

<code>overall_comparison</code>	<i>Compare champion with challengers globally</i>
---------------------------------	---

---

**Description**

The function creates objects that present global model performance using various measures. Those data can be easily plotted with `plot` function. It uses `auditor` package to create `model_performance` of all passed explainers. Keep in mind that type of task has to be specified.

**Usage**

```
overall_comparison(champion, challengers, type)
```

**Arguments**

`champion` - explainer of champion model.  
`challengers` - explainer of challenger model or list of explainers.  
`type` - type of the task. Either classification or regression

**Value**

An object of the class `overall_comparison`

It is a named list containing following fields:

- radar list of `model_performance` objects and other parameters that will be passed to generic `plot` function
- `accordance` data.frame object of champion responses and challenger's corresponding to them. Used to plot accordance.
- `models_info` data.frame containing information about models used in analysis.

**Examples**

```
library("DALEXtra")
library("mlr")
task <- mlr::makeRegrTask(
  id = "R",
  data = apartments,
  target = "m2.price"
)
learner_lm <- mlr::makeLearner(
  "regr.lm"
)
model_lm <- mlr::train(learner_lm, task)
```

```

explainer_lm <- explain_mlr(model_lm, apartmentsTest, apartmentsTest$m2.price, label = "LM")

learner_rf <- mlr::makeLearner(
  "regr.randomForest"
)
model_rf <- mlr::train(learner_rf, task)
explainer_rf <- explain_mlr(model_rf, apartmentsTest, apartmentsTest$m2.price, label = "RF")

learner_gbm <- mlr::makeLearner(
  "regr.gbm"
)
model_gbm <- mlr::train(learner_gbm, task)
explainer_gbm <- explain_mlr(model_gbm, apartmentsTest, apartmentsTest$m2.price, label = "gbm")

data <- overall_comparison(explainer_lm, list(explainer_gbm, explainer_rf), type = "regression")
plot(data)

```

---

plot.aspect\_importance

*Function for plotting aspect\_importance results*

---

## Description

This function plots the results of aspect\_importance.

## Usage

```

## S3 method for class 'aspect_importance'
plot(
  x,
  ...,
  bar_width = 10,
  aspects_on_axis = TRUE,
  add_importance = FALSE,
  digits_to_round = 2,
  text_size = 3
)

```

## Arguments

x	object of aspect_importance class
...	other parameters
bar_width	bar width
aspects_on_axis	if TRUE, labels on axis Y show aspect names, otherwise they show features names
add_importance	if TRUE, plot is annotated with values of aspects importance

`digits_to_round` integer indicating the number of decimal places used for rounding values of aspects importance shown on the plot

`text_size` size of labels annotating values of aspects importance, if applicable

**Value**

a ggplot2 object

**Examples**

```
library("DALEX")

model_titanic_glm <- glm(survived == 1 ~
  class+gender+age+sibsp+parch+fare+embarked,
  data = titanic_imputed,
  family = "binomial")

explain_titanic_glm <- explain(model_titanic_glm,
  data = titanic_imputed[,-8],
  y = titanic_imputed$survived == 1,
  verbose = FALSE)

aspects <- list(wealth = c("class", "fare"),
  family = c("sibsp", "parch"),
  personal = c("gender", "age"),
  embarked = "embarked")

plot(aspect_importance(explain_titanic_glm,
  new_observation = titanic_imputed[1,],
  aspects = aspects))
```

---

`plot.funnel_measure` *Funnel plot for difference in measures*

---

**Description**

Function `plot.funnel_measure` creates funnel plot of differences in measures for two models across variable areas. It uses data created with `'funnel_measure'` function.

**Usage**

```
## S3 method for class 'funnel_measure'
plot(x, ..., dot_size = 0.5)
```

**Arguments**

- x - funnel\_measure object created with `funnel_measure` function.
- ... - other parameters
- dot\_size - size of the dot on plots. Passed to `geom_point`.

**Value**

ggplot object

**Examples**

```
library("mlr")
library("DALEXtra")
task <- mlr::makeRegrTask(
  id = "R",
  data = apartments,
  target = "m2.price"
)
learner_lm <- mlr::makeLearner(
  "regr.lm"
)
model_lm <- mlr::train(learner_lm, task)
explainer_lm <- explain_mlr(model_lm, apartmentsTest, apartmentsTest$m2.price, label = "LM")

learner_rf <- mlr::makeLearner(
  "regr.randomForest"
)
model_rf <- mlr::train(learner_rf, task)
explainer_rf <- explain_mlr(model_rf, apartmentsTest, apartmentsTest$m2.price, label = "RF")

learner_gbm <- mlr::makeLearner(
  "regr.gbm"
)
model_gbm <- mlr::train(learner_gbm, task)
explainer_gbm <- explain_mlr(model_gbm, apartmentsTest, apartmentsTest$m2.price, label = "GBM")

plot_data <- funnel_measure(explainer_lm, list(explainer_rf, explainer_gbm),
  nbins = 5, measure_function = DALEX::loss_root_mean_square)
plot(plot_data)
```

---

plot.overall\_comparison

*Plot function for overall\_comparison*

---

**Description**

The function plots data created with `overall_comparison`. For radar plot it uses auditor's `plot_radar`. Keep in mind that the function creates two plots returned as list.

**Usage**

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

**Arguments**

```
x          - data created with overall_comparison
...        - other parameters
```

**Value**

A named list of ggplot objects.

It consists of:

- `radar_plot` plot created with `plot_radar`
- `accordance_plot` accordance plot of responses. OX axis stand for champion response, while OY for one of challengers responses. Colour indicates on challenger.

**Examples**

```
library("DALEXtra")
library("mlr")
task <- mlr::makeRegrTask(
  id = "R",
  data = apartments,
  target = "m2.price"
)
learner_lm <- mlr::makeLearner(
  "regr.lm"
)
model_lm <- mlr::train(learner_lm, task)
explainer_lm <- explain_mlr(model_lm, apartmentsTest, apartmentsTest$m2.price, label = "LM")

learner_rf <- mlr::makeLearner(
  "regr.randomForest"
)
model_rf <- mlr::train(learner_rf, task)
explainer_rf <- explain_mlr(model_rf, apartmentsTest, apartmentsTest$m2.price, label = "RF")

learner_gbm <- mlr::makeLearner(
  "regr.gbm"
)
model_gbm <- mlr::train(learner_gbm, task)
explainer_gbm <- explain_mlr(model_gbm, apartmentsTest, apartmentsTest$m2.price, label = "GBM")
```



```
data <- overall_comparison(explainer_lm, list(explainer_gbm, explainer_rf), type = "regression")
plot(data)
```

---

plot.training\_test\_comparison

*Plot and compare performance of model between training and test set*

---

## Description

Function `plot.training_test_comparison` plots dependency between model performance on test and training dataset based on `training_test_comparison` object. Green line indicates  $y = x$  line.

## Usage

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

## Arguments

`x` - object created with `training_test_comparison` function.  
`...` - other parameters

## Value

ggplot object

## Examples

```
library("mlr")
library("DALEXtra")
task <- mlr::makeRegrTask(
  id = "R",
  data = apartments,
  target = "m2.price"
)
learner_lm <- mlr::makeLearner(
  "regr.lm"
)
model_lm <- mlr::train(learner_lm, task)
explainer_lm <- explain_mlr(model_lm, apartmentsTest, apartmentsTest$m2.price, label = "LM")

learner_rf <- mlr::makeLearner(
  "regr.randomForest"
)
model_rf <- mlr::train(learner_rf, task)
explainer_rf <- explain_mlr(model_rf, apartmentsTest, apartmentsTest$m2.price, label = "RF")
```

```

learner_gbm <- mlr::makeLearner(
  "regr.gbm"
)
model_gbm <- mlr::train(learner_gbm, task)
explainer_gbm <- explain_mlr(model_gbm, apartmentsTest, apartmentsTest$m2.price, label = "GBM")

data <- training_test_comparison(explainer_lm, list(explainer_gbm, explainer_rf),
                                training_data = apartments,
                                training_y = apartments$m2.price)

plot(data)

```

---

`plot_aspects_importance_grouping`

*Function plots tree with aspect importance values*

---

### **Description**

This function plots tree that shows order of feature grouping and aspect importance values of every newly created aspect.

### **Usage**

```

plot_aspects_importance_grouping(
  x,
  data,
  predict_function = predict,
  new_observation,
  N = 100,
  clust_method = "complete",
  absolute_value = FALSE,
  cumulative_max = FALSE,
  show_labels = TRUE,
  axis_lab_size = 10,
  text_size = 3
)

```

### **Arguments**

<code>x</code>	a model to be explained
<code>data</code>	dataset, should be without target variable
<code>predict_function</code>	predict function
<code>new_observation</code>	selected observation with columns that corresponds to variables used in the model, should be without target variable

N	number of observations to be sampled (with replacement) from data
clust_method	the agglomeration method to be used, see <a href="#">hclust</a> methods
absolute_value	if TRUE, aspect importance values will be drawn as absolute values
cumulative_max	if TRUE, aspect importance shown on tree will be max value of children and node aspect importance values
show_labels	if TRUE, plot will have annotated axis Y
axis_lab_size	size of labels on axis Y, if applicable
text_size	size of labels annotating values of aspects importance

**Value**

ggplot

**Examples**

```
library(DALEX)
apartments_num <- apartments[,unlist(lapply(apartments, is.numeric))]
apartments_num_lm_model <- lm(m2.price ~ ., data = apartments_num)
apartments_num_new_observation <- apartments_num[2,-1]
apartments_num_mod <- apartments_num[,-1]
plot_aspects_importance_grouping(x = apartments_num_lm_model,
data = apartments_num_mod, new_observation = apartments_num_new_observation)
```

---

plot\_group\_variables *Plots tree with correlation values*

---

**Description**

Plots tree that illustrates the results of group\_variables function.

**Usage**

```
plot_group_variables(
  x,
  p,
  show_labels = TRUE,
  draw_abline = TRUE,
  axis_lab_size = 10,
  text_size = 3
)
```

**Arguments**

x	hclust object
p	correlation value for cutoff level
show_labels	if TRUE, plot will have annotated axis Y
draw_abline	if TRUE, cutoff line will be drawn
axis_lab_size	size of labels on axis Y, if applicable
text_size	size of labels annotating values of correlations

**Value**

tree plot

**Examples**

```
library("DALEX")
dragons_data <- dragons[,c(2,3,4,7,8)]
group_variables(dragons_data, p = 0.7, clust_method = "complete",
               draw_tree = TRUE)
```

---

print.funnel\_measure *Print funnel\_measure object*

---

**Description**

Print funnel\_measure object

**Usage**

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

**Arguments**

x	an object of class funnel_measure
...	other parameters

**Examples**

```
library("DALEXtra")
library("mlr")
task <- mlr::makeRegrTask(
  id = "R",
  data = apartments,
  target = "m2.price")
```

```

)
learner_lm <- mlr::makeLearner(
  "regr.lm"
)
model_lm <- mlr::train(learner_lm, task)
explainer_lm <- explain_mlr(model_lm, apartmentsTest, apartmentsTest$m2.price, label = "LM")

learner_rf <- mlr::makeLearner(
  "regr.randomForest"
)
model_rf <- mlr::train(learner_rf, task)
explainer_rf <- explain_mlr(model_rf, apartmentsTest, apartmentsTest$m2.price, label = "RF")

learner_gbm <- mlr::makeLearner(
  "regr.gbm"
)
model_gbm <- mlr::train(learner_gbm, task)
explainer_gbm <- explain_mlr(model_gbm, apartmentsTest, apartmentsTest$m2.price, label = "GBM")

plot_data <- funnel_measure(explainer_lm, list(explainer_rf, explainer_gbm),
                           nbins = 5, measure_function = DALEX::loss_root_mean_square)
print(plot_data)

```

---

```
print.overall_comparison
```

*Print overall\_comparison object*

---

## Description

Print overall\_comparison object

## Usage

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

## Arguments

x	an object of class overall_comparison
...	other parameters

## Examples

```

library("DALEXtra")
library("mlr")
task <- mlr::makeRegrTask(
  id = "R",

```

```

    data = apartments,
    target = "m2.price"
  )
  learner_lm <- mlr::makeLearner(
    "regr.lm"
  )
  model_lm <- mlr::train(learner_lm, task)
  explainer_lm <- explain_mlr(model_lm, apartmentsTest, apartmentsTest$m2.price, label = "LM")

  learner_rf <- mlr::makeLearner(
    "regr.randomForest"
  )
  model_rf <- mlr::train(learner_rf, task)
  explainer_rf <- explain_mlr(model_rf, apartmentsTest, apartmentsTest$m2.price, label = "RF")

  learner_gbm <- mlr::makeLearner(
    "regr.gbm"
  )
  model_gbm <- mlr::train(learner_gbm, task)
  explainer_gbm <- explain_mlr(model_gbm, apartmentsTest, apartmentsTest$m2.price, label = "gbm")

  data <- overall_comparison(explainer_lm, list(explainer_gbm, explainer_rf), type = "regression")
  print(data)

```

---

`print.scikitlearn_set` *Prints scikitlearn\_set class*

---

## Description

Prints `scikitlearn_set` class

## Usage

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

## Arguments

<code>x</code>	a list from explainer created with <a href="#">explain_scikitlearn</a>
<code>...</code>	other arguments

---

```
print.training_test_comparison
      Print funnel_measure object
```

---

### Description

Print funnel\_measure object

### Usage

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

### Arguments

x                    an object of class funnel\_measure  
...                   other parameters

### Examples

```
library("mlr")
library("DALEXtra")
task <- mlr::makeRegrTask(
  id = "R",
  data = apartments,
  target = "m2.price"
)
learner_lm <- mlr::makeLearner(
  "regr.lm"
)
model_lm <- mlr::train(learner_lm, task)
explainer_lm <- explain_mlr(model_lm, apartmentsTest, apartmentsTest$m2.price, label = "LM")

learner_rf <- mlr::makeLearner(
  "regr.randomForest"
)
model_rf <- mlr::train(learner_rf, task)
explainer_rf <- explain_mlr(model_rf, apartmentsTest, apartmentsTest$m2.price, label = "RF")

learner_gbm <- mlr::makeLearner(
  "regr.gbm"
)
model_gbm <- mlr::train(learner_gbm, task)
explainer_gbm <- explain_mlr(model_gbm, apartmentsTest, apartmentsTest$m2.price, label = "GBM")

data <- training_test_comparison(explainer_lm, list(explainer_gbm, explainer_rf),
  training_data = apartments,
  training_y = apartments$m2.price)

print(data)
```

---

 training\_test\_comparison

*Compare performance of model between training and test set*


---

## Description

Function `training_test_comparison` calculates performance of the provided model based on specified measure function. Response of the model is calculated based on test data, extracted from the explainer and training data, provided by the user. Output can be easily shown with `print` or `plot` function.

## Usage

```
training_test_comparison(
  champion,
  challengers,
  training_data,
  training_y,
  measure_function = NULL
)
```

## Arguments

`champion` - explainer of champion model.

`challengers` - explainer of challenger model or list of explainers.

`training_data` - data without target column that will be passed to predict function and then to measure function. Keep in mind that they have to differ from data passed to an explainer.

`training_y` - target column for `training_data`

`measure_function` - measure function that calculates performance of model based on true observation and prediction. Order of parameters is important and should be `(y, y_hat)`. By default it is RMSE.

## Value

An object of the class `training_test_comparison`.

It is a named list containing:

- `data` `data.frame` with following columns
  - `measure_test` performance on test set
  - `measure_train` performance on training set
  - `label` label of explainer
  - `type` flag that indicates if explainer was passed as champion or as challenger.
- `models_info` `data.frame` containing information about models used in analysis



**Examples**

```

library("mlr")
library("DALEXtra")
task <- mlr::makeRegrTask(
  id = "R",
  data = apartments,
  target = "m2.price"
)
learner_lm <- mlr::makeLearner(
  "regr.lm"
)
model_lm <- mlr::train(learner_lm, task)
explainer_lm <- explain_mlr(model_lm, apartmentsTest, apartmentsTest$m2.price, label = "LM")

learner_rf <- mlr::makeLearner(
  "regr.randomForest"
)
model_rf <- mlr::train(learner_rf, task)
explainer_rf <- explain_mlr(model_rf, apartmentsTest, apartmentsTest$m2.price, label = "RF")

learner_gbm <- mlr::makeLearner(
  "regr.gbm"
)
model_gbm <- mlr::train(learner_gbm, task)
explainer_gbm <- explain_mlr(model_gbm, apartmentsTest, apartmentsTest$m2.price, label = "GBM")

data <- training_test_comparison(explainer_lm, list(explainer_gbm, explainer_rf),
                               training_data = apartments,
                               training_y = apartments$m2.price)

plot(data)

```

---

triplot

*Three plots that sum up automatic aspect importance grouping*


---

**Description**

This function shows:

- plot for aspect\_importance with single aspect
- tree that shows aspect\_importance for every newly expanded aspect
- clustering tree.

**Usage**

```

triplot(x, ...)

## S3 method for class 'explainer'
triplot(

```

```

x,
new_observation,
N = 500,
clust_method = "complete",
absolute_value = FALSE,
cumulative_max = FALSE,
add_importance_labels = TRUE,
show_axis_y_duplicated_labels = FALSE,
axis_lab_size = 10,
text_size = 3,
...
)

## Default S3 method:
triplot(
  x,
  data,
  predict_function = predict,
  new_observation,
  N = 500,
  clust_method = "complete",
  absolute_value = FALSE,
  cumulative_max = FALSE,
  add_importance_labels = TRUE,
  show_axis_y_duplicated_labels = FALSE,
  abbrev_labels = 0,
  axis_lab_size = 10,
  text_size = 3,
  ...
)

```

### Arguments

x	an explainer created with the <code>DALEX::explain()</code> function or a model to be explained.
...	other parameters
new_observation	selected observation with columns that corresponds to variables used in the model, should be without target variable
N	number of rows to be sampled from data
clust_method	the agglomeration method to be used, see <a href="#">hclust</a> methods
absolute_value	if TRUE, aspect importance values will be drawn as absolute values
cumulative_max	if TRUE, aspect importance shown on tree will be max value of children and node aspect importance values
add_importance_labels	if TRUE, first plot is annotated with values of aspects importance

show\_axis\_y\_duplicated\_labels  
                                   if TRUE, every plot will have annotated axis Y  
 axis\_lab\_size   size of labels on axis  
 text\_size       size of labels annotating values of aspects importance and correlations  
 data            dataset, it will be extracted from x if it's an explainer NOTE: Target variable  
                                   shouldn't be present in the data  
 predict\_function  
                                   predict function, it will be extracted from x if it's an explainer  
 abbrev\_labels   if greater than 0, labels for axis Y in single aspect importance plot will be ab-  
                                   breviated according to this parameter

### Examples

```

library(DALEX)
apartments_num <- apartments[,unlist(lapply(apartments, is.numeric))]
apartments_num_lm_model <- lm(m2.price ~ ., data = apartments_num)
apartments_num_new_observation <- apartments_num[30,-1]
apartments_num_mod <- apartments_num[, -1]
triplot(x = apartments_num_lm_model,
        data = apartments_num_mod,
        new_observation = apartments_num_new_observation,
        add_importance_labels = FALSE)
  
```

---

yhat.WrappedModel           *Wrapper over the predict function*

---

### Description

These functions are default predict functions. Each function returns a single numeric score for each new observation. Those functions are very important since information from many models have to be extracted with various techniques.

### Usage

```

## S3 method for class 'WrappedModel'
yhat(X.model, newdata, ...)

## S3 method for class 'H2ORegressionModel'
yhat(X.model, newdata, ...)

## S3 method for class 'H2OBinomialModel'
yhat(X.model, newdata, ...)

## S3 method for class 'H2OMultinomialModel'
yhat(X.model, newdata, ...)
  
```

```
## S3 method for class 'scikitlearn_model'  
yhat(X.model, newdata, ...)  
  
## S3 method for class 'keras'  
yhat(X.model, newdata, ...)  
  
## S3 method for class 'LearnerRegr'  
yhat(X.model, newdata, ...)  
  
## S3 method for class 'LearnerClassif'  
yhat(X.model, newdata, ...)  
  
## S3 method for class 'GraphLearner'  
yhat(X.model, newdata, ...)  
  
## S3 method for class 'xgb.Booster'  
yhat(X.model, newdata, ...)
```

### Arguments

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

### Details

Currently supported packages are:

- mlr see more in [explain\\_mlr](#)
- h2o see more in [explain\\_h2o](#)
- scikit-learn see more in [explain\\_scikitlearn](#)
- keras see more in [explain\\_keras](#)
- mlr3 see more in [explain\\_mlr3](#)
- xgboost see more in [explain\\_xgboost](#)

### Value

An numeric vector of predictions

# Index

aspect\_importance, 3  
aspect\_importance\_single, 5

champion\_challenger, 7  
create\_env, 8

explain, 10, 12, 13, 15, 17–19, 22  
explain\_h2o, 9, 44  
explain\_keras, 11, 44  
explain\_mlr, 14, 44  
explain\_mlr3, 16, 44  
explain\_scikitlearn, 13, 18, 38, 44  
explain\_xgboost, 21, 44

funnel\_measure, 7, 23, 31

geom\_point, 31  
get\_sample, 4, 6, 25  
group\_variables, 26

hclust, 26, 35, 42

lime (aspect\_importance), 3

model\_info.GraphLearner  
(model\_info.WrappedModel), 26  
model\_info.H20BinomialModel  
(model\_info.WrappedModel), 26  
model\_info.H20MultinomialModel  
(model\_info.WrappedModel), 26  
model\_info.H20RegressionModel  
(model\_info.WrappedModel), 26  
model\_info.keras  
(model\_info.WrappedModel), 26  
model\_info.LearnerClassif  
(model\_info.WrappedModel), 26  
model\_info.LearnerRegr  
(model\_info.WrappedModel), 26  
model\_info.scikitlearn\_model  
(model\_info.WrappedModel), 26  
model\_info.WrappedModel, 26

model\_info.xgb.Booster  
(model\_info.WrappedModel), 26  
model\_performance, 28

overall\_comparison, 7, 28, 32

plot.aspect\_importance, 29  
plot.funnel\_measure, 7, 30  
plot.overall\_comparison, 31  
plot.training\_test\_comparison, 33  
plot\_aspects\_importance\_grouping, 34  
plot\_group\_variables, 35  
plot\_radar, 32  
print.funnel\_measure, 36  
print.overall\_comparison, 37  
print.scikitlearn\_set, 38  
print.training\_test\_comparison, 39

training\_test\_comparison, 7, 33, 40  
triplet, 41

yhat.GraphLearner (yhat.WrappedModel),  
43  
yhat.H20BinomialModel  
(yhat.WrappedModel), 43  
yhat.H20MultinomialModel  
(yhat.WrappedModel), 43  
yhat.H20RegressionModel  
(yhat.WrappedModel), 43  
yhat.keras (yhat.WrappedModel), 43  
yhat.LearnerClassif  
(yhat.WrappedModel), 43  
yhat.LearnerRegr (yhat.WrappedModel), 43  
yhat.scikitlearn\_model  
(yhat.WrappedModel), 43  
yhat.WrappedModel, 43  
yhat.xgb.Booster (yhat.WrappedModel), 43