Package 'DALEXtra'

November 18, 2019

Title Extension for 'DALEX' Package

Version 0.2.0

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, 'java' 'h2o' library and 'mljar' API. 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 (>= 0.4.9)

License GPL

LazyData true

RoxygenNote 7.0.0

Imports reticulate, ggplot2, glmnet, ggdendro, gridExtra

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

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],
Przemyslaw Biecek [aut] (<https://orcid.org/0000-0001-8423-1823>),
Katarzyna Pekala [aut],
Anna Kozak [ctb]

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

Repository CRAN

Date/Publication 2019-11-18 16:00:02 UTC

R topics documented:

`aspect_importance` *Calculates the feature groups importance (called aspects importance) for a selected observation*

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 DALEX::explain() 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 containting grouping of features into aspects
N	number of observations to be sampled (with replacement) from data
sample_method	sampling method in get_sample
n_var	maximum number of non-zero coefficients after lasso fitting, if zero than linear regression is used

f	frequency in get_sample
show_cor	show if all features in aspect are pairwise positivly 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")
library("ingredients")

titanic_imputed$country <- NULL

model_titanic_glm <- glm(survived == "yes" ~
                         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 == "yes",
                               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 == "yes",
                              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 `DALEX::explain()` 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 get_sample
n_var	how many non-zero coefficients for lasso fitting, if zero than linear regression is used
f	frequency in in get_sample
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 == "yes" ~ class + gender + age +
                         sibsp + parch + fare + embarked,
                         data = titanic_imputed,
                         family = "binomial")

aspect_importance_single(model_titanic_glm, data = titanic_imputed[,-9],
                         new_observation = titanic_imputed[1,-9])
```

---

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 judgemnt. 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 inculded 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 funnel_measure training_test_comparison, overall_comparison or any other explanation that can work with plot function. Please provide name for not standard sections, that will be presented as section titles. Oterwise class of the object will be used.
dot_size	- dot_size argument passed to plot.funnel_measure if funnel_measure 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 overall_comparison 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 rmarkdown::render.

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 requiers 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

```
if(DALEXtra:::is_conda()) {
  create_env(system.file("extdata", "testing_environment.yml", package = "DALEXtra"))
} else {
  "conda is required"
}
```

`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
)
```

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 my 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. This will happenn also if 'verbose' is TRUE
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.

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"))
h2o::h2o.init()
h2o::h2o.no_progress()
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::h2o.shutdown(prompt = FALSE)
```

`explain_keras` *Wrapper for Python Keras Models*

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
)
```

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 `explain`.
`y`	vector that will be passed to `explain`.
`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 `explain`. If NULL, default will be used.

residual_function	residual function that will be passed into explain. If NULL, default will be used.
...	other parameters
label	label that will be passed into explain. If NULL, default will be used.
verbose	bool that will be passed into explain. If NULL, default will be used.
precalculate	if TRUE (default) then 'predicted_values' and 'residuals' are calculated when explainer is created. This will happenn also if 'verbose' is TRUE.
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.

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 .yml file
If you provide .yml 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.yml`

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 avaialble from offcial 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 yo 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 availbe for anaconda at all, use pip statement

If .yml file seems not to work, virtual env can be created manually using anaconda promt.
`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")

   # 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])
```

---

explain_mljar *Create explainer from your mljar model*

Description

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

Usage

```
explain_mljar(
  model,
  project_title,
  data = NULL,
  y = NULL,
  weights = NULL,
  predict_function = NULL,
  residual_function = NULL,
  ...,
  label = NULL,
  verbose = TRUE,
```

```
  precalculate = TRUE,
  colorize = TRUE,
  model_info = NULL
)
```

Arguments

model	object - a mljar model to be explained
project_title	character - a name of project_title in which model was built. Without it predictions are unreachable.
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 my 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. This will happenn also if 'verbose' is TRUE
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.

Value

explainer object (explain) ready to work with DALEX

Examples

```
## Not run:
library("DALEXtra")
library(mljar)
titanic_test <- read.csv(system.file("extdata", "titanic_test.csv", package = "DALEXtra"))
```

```
model <- mljar_fit(titanic_test[,1:17], titanic_test[,18],
                   proj_title="Project title", exp_title="experiment title",
                   algorithms = c("logreg"), metric = "logloss")
# It Works
explainer <- explain_mljar(model, project_title = "Project title",
                           data = titanic_test[,1:17], y = titanic_test[,18])
# But it works aswell
explainer <- explain_mljar(model, project_title = "Project title",
                           verbose = FALSE, precalculate = FALSE)

## 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
)
```

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 my 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. This will happenn also if 'verbose' is TRUE
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.

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
)
```

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 my 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. This will happenn also if 'verbose' is TRUE

`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.

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
)
```

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 explain.
y	vector that will be passed to explain.
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 explain. If NULL, default will be used.
residual_function	residual function that will be passed into explain. If NULL, default will be used.
...	other parameters
label	label that will be passed into explain. If NULL, default will be used.
verbose	bool that will be passed into explain. If NULL, default will be used.
precalculate	if TRUE (default) then 'predicted_values' and 'residuals' are calculated when explainer is created. This will happenn also if 'verbose' is TRUE.
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.

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 .yml, activating virtual env via `activate name_of_the_env` and execution of the following shell command is necessary
`conda env export > environment.yml`

Errors use case

Here is shortened version of solution for specific errors

There already exists environment with a name specified by given .yml file
If you provide .yml 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.yml`

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 avaialble from offcial 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 yo 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 availbe for anaconda at all, use pip statement

If .yml file seems not to work, virtual env can be created manually using anaconda promt.
`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")
if(DALEXtra:::is_conda()) {
   # 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])

} else {
  print('Conda is required.')
}
```

---

funnel_measure *Caluculate difference in performance in models across different categories*

Description

Function `funnel_measure` allows users to compare two models based on their explainers. It partitions dataset on which models were builded and creates categories according to quantiles of columns in `parition 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 = DALEX::loss_root_mean_square,
  nbins = 5,
  partition_data = champion$data,
  cutoff = 0.01,
  cutoff_name = "Other",
  factor_conversion_threshold = 7,
```

```
  show_info = TRUE,
  categories = NULL
)
```

Arguments

`champion`	- explainer of champion model.
`challengers`	- explainer of challenger model or list of explainers.
`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). The measure calculated by the function should have the property that lower score value indicates better model. By default it is RMSE.
`nbins`	- 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.
`partition_data`	- Data by which test dataset will be paritioned 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.
`cutoff`	- 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 controled 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 *Groups numeric features into aspects*

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 hclust 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`

Exract 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 inforamtion are stored

Usage

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

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

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

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

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

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

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

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

Arguments

`model`	- model object
`...`	- 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` pyhton library and accesed via `reticulate`
- `keras` models created with `keras` pyhton library and accesed via `reticulate`
- `mljar` models created with `mljar` API and accesed via `mljar` R package
- `mlr3` models created with `mlr3` package

Value

A named list of class `model_info`

`overall_comparison` *Compare champion with challengers globally*

Description

The function creates objects that present global model perfromance using various measures. Those date can be easily ploted 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 containig inforamtion about models used in analysys.

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, oherwise 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")

titanic_imputed$country <- NULL

model_titanic_glm <- glm(survived == "yes" ~
                         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 == "yes",
                               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 dependecy between model performance on test and trainig 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(DALEXtra)
titanic_train <- read.csv(system.file("extdata", "titanic_train.csv", package = "DALEXtra"))
titanic_test <- read.csv(system.file("extdata", "titanic_test.csv", package = "DALEXtra"))
```

```
h2o::h2o.init()
h2o::h2o.no_progress()
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
)
explainer_h2o <- explain_h2o(model, titanic_test[,1:17], titanic_test[,18])

explainer_scikit <- 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)

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)
explainer_mlr <- explain_mlr(gbm, titanic_test[,1:17], titanic_test[,18])

data <- training_test_comparison(explainer_scikit, list(explainer_h2o, explainer_mlr),
                                 training_data = titanic_train[,-18],
                                 training_y = titanic_train[,18])
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

`x`	a model to be explained
`data`	dataset, should be without target variable
`predict_function`	predict function
`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
`clust_method`	the agglomeration method to be used, see `hclust` 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

`x`	a list from explainer created with `explain_scikitlearn`
`...`	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(DALEXtra)
titanic_train <- read.csv(system.file("extdata", "titanic_train.csv", package = "DALEXtra"))
titanic_test <- read.csv(system.file("extdata", "titanic_test.csv", package = "DALEXtra"))
h2o::h2o.init()
h2o::h2o.no_progress()
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
)
explainer_h2o <- explain_h2o(model, titanic_test[,1:17], titanic_test[,18])

explainer_scikit <- 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)

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)
explainer_mlr <- explain_mlr(gbm, titanic_test[,1:17], titanic_test[,18])
```

```
data <- training_test_comparison(explainer_scikit, list(explainer_h2o, explainer_mlr),
                                 training_data = titanic_train[,-18],
                                 training_y = titanic_train[,18])
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 caluclated 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 = DALEX::loss_root_mean_square
)
```

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 containig:

- `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 containig inforamtion about models used in analysys

Examples

```
library(DALEXtra)
titanic_train <- read.csv(system.file("extdata", "titanic_train.csv", package = "DALEXtra"))
titanic_test <- read.csv(system.file("extdata", "titanic_test.csv", package = "DALEXtra"))
h2o::h2o.init()
h2o::h2o.no_progress()
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
)
explainer_h2o <- explain_h2o(model, titanic_test[,1:17], titanic_test[,18])

explainer_scikit <- 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)

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)
explainer_mlr <- explain_mlr(gbm, titanic_test[,1:17], titanic_test[,18])

data <- training_test_comparison(explainer_scikit, list(explainer_h2o, explainer_mlr),
                                 training_data = titanic_train[,-18],
                                 training_y = titanic_train[,18])
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 DALEX::explain() 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 hclust 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 abbreviated 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 functios are default predict functions. Each function returns a single numeric score for each new observation. Those functions are very important since informations 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 'scikitlearn_model'
yhat(X.model, newdata, ...)

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

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

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

## S3 method for class 'LearnerClassif'
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:

- `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`
- `mljar` see more in `explain_mljar`
- `mlr3` see more in `explain_mlr3`

Value

An numeric vector of predictions

Index