Package ‘DALEXtra’

May 9, 2021

Title Extension for 'DALEX' Package

Version 2.1.1

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> ex-
   plainer 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

RoxygenNote 7.1.1

Imports reticulate, ggplot2

Suggests auditor, ingredients, gbm, ggrepel, h2o, iml, lime,
   localModel, mlr, mlr3, randomForest, recipes, rmarkdown, rpart,
   xgboost, testthat, tidymodels

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>),
   Przemyslaw Biecek [aut] (<https://orcid.org/0000-0001-8423-1823>),
   Anna Kozak [ctb],
   Hubert Baniecki [ctb]
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 inculded at report.
champion_challenger

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
            DAEXtra 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
donput_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("DAEXtra")
task <- mlr::makeRegrTask(
  id = "R",
  data = apartments,
  target = "m2.price"
)
learner_lm <- mlr::makeLearner(
create_env

"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 function allows the user to create a conda virtual environment based on a 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)

Usage

explain_h2o(
  model,
  data = NULL,
  y = NULL,
  weights = NULL,
  predict_function = NULL,
  predict_function_target_column = 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
predict_function_target_column Character or numeric containing either column name or column number in the model prediction object of the class that should be considered as positive (ie. the class that is associated with probability 1). If NULL, the second column of the output will be taken for binary classification. For a multiclass classification setting that parameter cause switch to binary classification mode with 1 vs others probabilities.
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

```r
# load packages and data
library(h2o)
library(DALEXtra)

# data <- DALEX::titanic_imputed

# init h2o
```
explain_keras

cluster <- try(h2o::h2o.init())
if (!inherits(cluster, "try-error")) {
    # 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)

---

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

```r
explain_keras(
    path,
    yml = NULL,
    condaenv = NULL,
    env = NULL,
    data = NULL,
    y = NULL,
    weights = NULL,
    predict_function = NULL,
    predict_function_target_column = 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 `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.
- `predict_function_target_column` Character or numeric containing either column name or column number in the model prediction object of the class that should be considered as positive (ie.
the class that is associated with probability 1). If NULL, the second column of
the output will be taken for binary classification. For a multiclass classification
setting that parameter cause switch to binary classification mode with 1 vs others
probabilities.

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.
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 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 available 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 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 dif-
ferent 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 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
library("DALEXtra")
## Not run:
# Explainer build (Keep in mind that 9th column is target)
test_data <-
read.csv(
  sep = ",")
# Keep in mind that when pickle is being built and loaded,
# not only Python version but libraries versions has to match as well
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

```r
y = NULL,
weights = NULL,
predict_function = NULL,
predict_function_target_column = 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 might 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

- **predict_function_target_column**
  - Character or numeric containing either column name or column number in the model prediction object of the class that should be considered as positive (i.e. the class that is associated with probability 1). If `NULL`, the second column of the output will be taken for binary classification. For a multiclass classification setting that parameter cause switch to binary classification mode with 1 vs others probabilities.

- **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.
If `TRUE` (default) then `WARNINGS`, `ERRORS` and `NOTES` are colorized. Will work only in the R console.

A named list (package, version, type) containing information about the model. If `NULL`, DALEX will seek for information on its own.

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

```r
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])
```

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**

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

**Arguments**

- **model** object - a fitted learner 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
- **predict_function_target_column** Character or numeric containing either column name or column number in the model prediction object of the class that should be considered as positive (ie. the class that is associated with probability 1). If NULL, the second column of the output will be taken for binary classification. For a multiclass classification setting that parameter cause switch to binary classification mode with 1 vs others probabilities.
- **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

```r
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

```r
explain_scikitlearn(path, yml = NULL,)
```
condaenv = NULL,
env = NULL,
data = NULL,
y = NULL,
weights = NULL,
predict_function = NULL,
predict_function_target_column = 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 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.
* predict_function_target_column: Character or numeric containing either column name or column number in the model prediction object of the class that should be considered as positive (ie. the class that is associated with probability 1). If NULL, the second column of the output will be taken for binary classification. For a multiclass classification setting that parameter cause switch to binary classification mode with 1 vs others probabilities.
* 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.

**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**

```python
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 available from official conda repo. Edit Your .yml file and add link to proper reposititory 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 available 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

```r
## 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_tidymodels  Create explainer from your tidymodels workflow.

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 tidymodels package. We would like to present dedicated explain function for it.
explain_tidymodels

Usage

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

Arguments

- **model**: object - a fitted workflow created with mlr3.
- **data**: data.frame or matrix - data that was used for fitting. 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. Tibble will be converted into data.frame
- **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
- **predict_function_target_column**: Character or numeric containing either column name or column number in the model prediction object of the class that should be considered as positive (ie. the class that is associated with probability 1). If NULL, the second column of the output will be taken for binary classification. For a multiclass classification setting that parameter cause switch to binary classification mode with 1 vs others probabilities.
- **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
explain_xgboost

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("tidymodels")
library("recipes")
data <- titanic_imputed
data$survived <- as.factor(data$survived)
rec <- recipe(survived ~ ., data = data) %>%
  step_normalize(fare)
model <- decision_tree(tree_depth = 25) %>%
  set_engine("rpart") %>%
  set_mode("classification")

wflow <- workflow() %>%
  add_recipe(rec) %>%
  add_model(model)

model_fitted <- wflow %>%
  fit(data = data)

explain_tidymodels(model_fitted, data = titanic_imputed, y = titanic_imputed$survived)

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

```r
explain_xgboost(
  model,
  data = NULL,
  y = NULL,
  weights = NULL,
  predict_function = NULL,
  predict_function_target_column = NULL,
  residual_function = 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. For classif task has to be numeric in range \([0, \text{nclasses})\).
- `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.
- `predict_function_target_column`: Character or numeric containing either column name or column number in the model prediction object of the class that should be considered as positive (i.e., the class that is associated with probability 1). If `NULL`, the second column of the output will be taken for binary classification. For a multiclass classification setting that parameter cause switch to binary classification mode with 1 vs others probabilities.
- `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
**explain_xgboost**

- **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**

```r
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  

Calculate 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 built and creates categories according to quantiles of columns in partition data. nbins parameter determinates number of quanatiles. For each category difference in provided measure is being calculated. Positive value of that difference 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

```r
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

- **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. If NULL, RMSE will be used for regression, one minus auc for classification and crossentropy for multiclass classification.
- **nbins** - Number of quanatiles (partition points) for numeric columns. In case when more than one quanatile have the same value, there will be less partition points.
- **partition_data** - 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.
- **cutoff** - Threshold for categorical data. Entries less frequent than specified value will be merged into one category.
funnel_measure

- 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 inforamation about models used in analyses

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)

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, ...)
overall_comparison

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

## S3 method for class 'workflow'
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
- `tidymodels` models created with `tidymodels` 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 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
plot.funnel_measure

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 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.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.
plot.funnel_measure

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

```r
## 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

```r
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)
predict_surrogate

Instance Level Surrogate Models

Description

Interface to different implementations of the LIME method. Find information how the LIME method works here: https://ema.drwhy.ai/LIME.html.

Usage

predict_surrogate(explainer, new_observation, ..., type = "localModel")

predict_surrogate_local_model(explainer,
     new_observation,
     size = 1000,
     seed = 1313,
     ...
)

predict_model.dalex_explainer(x, newdata, ...)

model_type.dalex_explainer(x, ...)

predict_surrogate_lime(explainer,
     new_observation,
     n_features = 4,
## S3 method for class 'predict_surrogate_lime'

```r
plot(x, ...)
```

`predict_surrogate_iml(explainer, new_observation, k = 4, ...)`

### Arguments

- **explainer**: a model to be explained, preprocessed by the `explain` function
- **new_observation**: a new observation for which predictions need to be explained
- **...**: other parameters that will be passed to
- **type**: which implementation of the LIME method should be used. Either `localModel` (default), `lime` or `iml`.
- **size**: will be passed to the localModel implementation, by default 1000
- **seed**: seed for random number generator, by default 1313
- **x**: an object to be plotted
- **newdata**: alias for `new_observation`
- **n_features**: will be passed to the lime implementation, by default 4
- **n_permutations**: will be passed to the lime implementation, by default 1000
- **labels**: will be passed to the lime implementation, by default first value in the y vector
- **k**: will be passed to the iml implementation, by default 4

### Value

Depending on the **type** there are different classess of the resulting object.

### References


---

**print.funnel_measure**  
Print funnel_measure object

---

**Description**

Print funnel_measure object
print.overall_comparison

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
print.training_test_comparison

Usage

```r
## 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

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

Arguments

- `x`: an object of class funnel_measure
- `...`: other parameters

Examples

```r
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")
```
training_test_comparison

```r
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

```r
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

```r
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)
```

---

**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 informations from many models have to be extracted with various techniques.

**Usage**

```r
## 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, ...)

## S3 method for class 'workflow'
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
• mlr3 see more in explain_mlr3
• xgboost see more in explain_xgboost
• tidymodels see more in explain_tidymodels

Value

An numeric vector of predictions
champion_challenger, 2
create_env, 4
explain, 6, 8, 9, 12, 14, 15, 19, 21
explain_h2o, 5, 38
explain_keras, 7, 38
explain_mlr, 10, 38
explain_mlr3, 12, 38
explain_scikitlearn, 9, 14, 34, 38
explain_tidymodels, 17, 38
explain_xgboost, 19, 38
funnel_measure, 2, 3, 22, 27
geom_point, 27
model_info.GraphLearner
    (model_info.WrappedModel), 24
model_info.H2OBinomialModel
    (model_info.WrappedModel), 24
model_info.H2OMultinomialModel
    (model_info.WrappedModel), 24
model_info.H2ORegressionModel
    (model_info.WrappedModel), 24
model_info.keras
    (model_info.WrappedModel), 24
model_info.LearnerClassif
    (model_info.WrappedModel), 24
model_info.LearnerRegr
    (model_info.WrappedModel), 24
model_info.scikitlearn_model
    (model_info.WrappedModel), 24
model_info.workflow
    (model_info.WrappedModel), 24
model_info.xgb.Booster
    (model_info.WrappedModel), 24
model_performance, 25, 26
model_type.dalex_explainer
    (predict_surrogate), 30
overall_comparison, 2, 3, 25, 28
plot.funnel_measure, 3, 26
plot.overall_comparison, 28
plot.predict_surrogate_lime
    (predict_surrogate), 30
plot.training_test_comparison, 29
plot.radar, 28
predict_model.dalex_explainer
    (predict_surrogate), 30
predict_parts (predict_surrogate), 30
predict_parts_break_down
    (predict_surrogate), 30
predict_parts_ibreak_down
    (predict_surrogate), 30
predict_parts_shap (predict_surrogate), 30
predict_surrogate, 30
predict_surrogate_ilm
    (predict_surrogate), 30
predict_surrogate_lime
    (predict_surrogate), 30
predict_surrogate_local_model
    (predict_surrogate), 30
print.funnel_measure, 31
print.overall_comparison, 32
print.scikitlearn_set, 33
print.training_test_comparison, 34
training_test_comparison, 2, 3, 29, 35
yhat.GraphLearner (yhat.WrappedModel), 36
yhat.H2OBinomialModel
    (yhat.WrappedModel), 36
yhat.H2OMultinomialModel
    (yhat.WrappedModel), 36
yhat.H2ORegressionModel
    (yhat.WrappedModel), 36
yhat.keras (yhat.WrappedModel), 36
yhat.LearnerClassif(yhat.WrappedModel), 36
yhat.LearnerRegr(yhat.WrappedModel), 36
yhat.scikitlearn_model(yhat.WrappedModel), 36
yhat.workflow(yhat.WrappedModel), 36
yhat.WrappedModel, 36
yhat.xgb.Booster(yhat.WrappedModel), 36