Package 'modelStudio'

February 21, 2023

Title Interactive Studio for Explanatory Model Analysis

Version 3.1.2

Description Automate the explanatory analysis of machine learning predictive models. Generate advanced interactive model explanations in the form of a serverless HTML site with only one line of code. This tool is model-agnostic, therefore compatible with most of the black-box predictive models and frameworks. The main function computes various (instance and model-level) explanations and produces a customisable dashboard, which consists of multiple panels for plots with their short descriptions. It is possible to easily save the dashboard and share it with others. modelStudio facilitates the process of Interactive Explanatory Model Analysis introduced in Baniecki et al. (2023) <doi:10.1007/s10618-023-00924-w>.

Depends R (>= 3.6)

License GPL-3

Encoding UTF-8

RoxygenNote 7.1.2

Imports DALEX (>= 2.2.1), ingredients (>= 2.2.0), iBreakDown (>= 2.0.1), r2d3, jsonlite, progress, digest

Suggests parallelMap, ranger, xgboost, knitr, rmarkdown, testthat, spelling

VignetteBuilder knitr

URL https://modelstudio.drwhy.ai, https://github.com/ModelOriented/modelStudio

BugReports https://github.com/ModelOriented/modelStudio/issues

Language en-US

LazyData true

NeedsCompilation no

Author Hubert Baniecki [aut, cre] (<https://orcid.org/0000-0001-6661-5364>), Przemyslaw Biecek [aut] (<https://orcid.org/0000-0001-8423-1823>), Piotr Piatyszek [ctb]

Maintainer Hubert Baniecki <hbaniecki@gmail.com>

Repository CRAN

Date/Publication 2023-02-20 23:20:02 UTC

R topics documented:

`happiness_train` *World Happiness Report*

Description

Datasets `happiness_train` and `happiness_test` are real data from the World Happiness Reports. Happiness is scored according to economic production, social support, etc. `happiness_train` accumulates the data from years 2015-2018, while `happiness_test` is the data from the year 2019, which imitates the out-of-time validation.

Usage

```
data(happiness_train); data(happiness_test)
```

Format

`happiness_train`: a data frame with 625 rows and 7 columns, `happiness_test`: a data frame with 156 rows and 7 columns

Details

Source: World Happiness Report at Kaggle.com

The following columns: GDP per Capita, Social Support, Life Expectancy, Freedom, Generosity, Corruption describe the extent to which these factors contribute in evaluating the happiness in each country. Variables:

- **score** - target variable, continuous value between 0 and 10 (regression)
- gdp_per_capita
- social_support
- healthy_life_expectancy
- freedom_life_choices

- generosity
- perceptions_of_corruption

`modelStudio` *Interactive Studio for Explanatory Model Analysis*

Description

This function computes various (instance and dataset level) model explanations and produces a customisable dashboard, which consists of multiple panels for plots with their short descriptions. Easily save the dashboard and share it with others. Tools for Explanatory Model Analysis unite with tools for Exploratory Data Analysis to give a broad overview of the model behavior.

The extensive documentation covers:

- Function parameters description - **perks and features**
- Framework and model compatibility - **R & Python examples**
- Theoretical introduction to the plots - Explanatory Model Analysis: Explore, Explain, and Examine Predictive Models

Displayed variable can be changed by clicking on the bars of plots or with the first dropdown list, and observation can be changed with the second dropdown list. The dashboard gathers useful, but not sensitive, information about how it is being used (e.g. computation length, package version, dashboard dimensions). This is for the development purposes only and can be blocked by setting `telemetry` to `FALSE`.

Usage

```
modelStudio(explainer, ...)

## S3 method for class 'explainer'
modelStudio(
  explainer,
  new_observation = NULL,
  new_observation_y = NULL,
  new_observation_n = 3,
  facet_dim = c(2, 2),
  time = 500,
  max_features = 10,
  max_features_fi = NULL,
  N = 300,
  N_fi = N * 10,
  N_sv = N * 3,
  B = 10,
  B_fi = B,
  eda = TRUE,
  open_plots = c("fi"),
```

```
  show_info = TRUE,
  parallel = FALSE,
  options = ms_options(),
  viewer = "external",
  widget_id = NULL,
  license = NULL,
  telemetry = TRUE,
  max_vars = NULL,
  verbose = NULL,
  ...
)
```

Arguments

explainer	An explainer created with DALEX::explain().
...	Other parameters.
new_observation	New observations with columns that correspond to variables used in the model.
new_observation_y	True label for new_observation (optional).
new_observation_n	Number of observations to be taken from the explainer$data if new_observation = NULL. See **vignette**
facet_dim	Dimensions of the grid. Default is c(2,2).
time	Time in ms. Set the animation length. Default is 500.
max_features	Maximum number of features to be included in BD, SV, and FI plots. Default is 10.
max_features_fi	Maximum number of features to be included in FI plot. Default is max_features.
N	Number of observations used for the calculation of PD and AD. Default is 300. See **vignette**
N_fi	Number of observations used for the calculation of FI. Default is 10*N.
N_sv	Number of observations used for the calculation of SV. Default is 3*N.
B	Number of permutation rounds used for calculation of SV. Default is 10. See **vignette**
B_fi	Number of permutation rounds used for calculation of FI. Default is B.
eda	Compute EDA plots and Residuals vs Feature plot, which adds the data to the dashboard. Default is TRUE.
open_plots	A vector listing plots to be initially opened (and on which positions). Default is c("fi").
show_info	Verbose a progress on the console. Default is TRUE.
parallel	Speed up the computation using parallelMap::parallelMap(). See **vignette**. This might interfere with showing progress using show_info.
options	Customize modelStudio. See ms_options and **vignette**.

viewer	Default is external to display in an external RStudio window. Use browser to display in an external browser or internal to use the RStudio internal viewer pane for output.
widget_id	Use an explicit element ID for the widget (rather than an automatically generated one). Useful e.g. when using modelStudio with Shiny. See **vignette**.
license	Path to the file containing the license (con parameter passed to readLines()). It can be used e.g. to include the license for explainer$data as a comment in the source of .html output file.
telemetry	The dashboard gathers useful, but not sensitive, information about how it is being used (e.g. computation length, package version, dashboard dimensions). This is for the development purposes only and can be blocked by setting telemetry to FALSE.
max_vars	An alias for max_features. If provided, it will override the value.
verbose	An alias for show_info. If provided, it will override the value.

Value

An object of the r2d3, htmlwidget, modelStudio class.

References

- The input object is implemented in **DALEX**
- Feature Importance, Ceteris Paribus, Partial Dependence and Accumulated Dependence explanations are implemented in **ingredients**
- Break Down and Shapley Values explanations are implemented in **iBreakDown**

See Also

Vignettes: **modelStudio - R & Python examples** and **modelStudio - perks and features**

Examples

```
library("DALEX")
library("modelStudio")

#:# ex1 classification on 'titanic' data

# fit a model
model_titanic <- glm(survived ~., data = titanic_imputed, family = "binomial")

# create an explainer for the model
explainer_titanic <- explain(model_titanic,
                             data = titanic_imputed,
                             y = titanic_imputed$survived,
                             label = "Titanic GLM")

# pick observations
new_observations <- titanic_imputed[1:2,]
rownames(new_observations) <- c("Lucas","James")
```

```
# make a studio for the model
modelStudio(explainer_titanic,
            new_observations,
            N = 200,  B = 5) # faster example



#:# ex2 regression on 'apartments' data
if (requireNamespace("ranger", quietly=TRUE)) {
  library("ranger")
  model_apartments <- ranger(m2.price ~. ,data = apartments)

  explainer_apartments <- explain(model_apartments,
                                  data = apartments,
                                  y = apartments$m2.price)

  new_apartments <- apartments[1:2,]
  rownames(new_apartments) <- c("ap1","ap2")

  # change dashboard dimensions and animation length
  modelStudio(explainer_apartments,
              new_apartments,
              facet_dim = c(2, 3),
              time = 800)

  # add information about true labels
  modelStudio(explainer_apartments,
              new_apartments,
              new_observation_y = new_apartments$m2.price)

  # don't compute EDA plots
  modelStudio(explainer_apartments,
              eda = FALSE)
}

#:# ex3 xgboost model on 'HR' dataset
if (requireNamespace("xgboost", quietly=TRUE)) {
  library("xgboost")
  HR_matrix <- model.matrix(status == "fired" ~ . -1, HR)

  # fit a model
  xgb_matrix <- xgb.DMatrix(HR_matrix, label = HR$status == "fired")
  params <- list(max_depth = 3, objective = "binary:logistic", eval_metric = "auc")
  model_HR <- xgb.train(params, xgb_matrix, nrounds = 300)

  # create an explainer for the model
  explainer_HR <- explain(model_HR,
                          data = HR_matrix,
                          y = HR$status == "fired",
                          type = "classification",
                          label = "xgboost")
```

```
  # pick observations
  new_observation <- HR_matrix[1:2, , drop=FALSE]
  rownames(new_observation) <- c("id1", "id2")

  # make a studio for the model
  modelStudio(explainer_HR,
              new_observation)
}
```

`ms_merge_observations` *Merge the observations of modelStudio objects*

Description

This function merges local explanations from multiple `modelStudio` objects into one.

Usage

```
ms_merge_observations(...)
```

Arguments

`...`	`modelStudio` objects created with `modelStudio()`.

Value

An object of the `r2d3`, `htmlwidget`, `modelStudio` class.

References

- The input object is implemented in **DALEX**
- Feature Importance, Ceteris Paribus, Partial Dependence and Accumulated Dependence explanations are implemented in **ingredients**
- Break Down and Shapley Values explanations are implemented in **iBreakDown**

See Also

Vignettes: **modelStudio - R & Python examples** and **modelStudio - perks and features**

Examples

```
library("DALEX")
library("modelStudio")

# fit a model
model_happiness <- glm(score ~., data = happiness_train)

# create an explainer for the model
explainer_happiness <- explain(model_happiness,
                               data = happiness_test,
                               y = happiness_test$score)

# make studios for the model
ms1 <- modelStudio(explainer_happiness,
                   N = 200,  B = 5)

ms2 <- modelStudio(explainer_happiness,
                   new_observation = head(happiness_test, 3),
                   N = 200,  B = 5)

# merge
ms <- ms_merge_observations(ms1, ms2)
ms
```

---

ms_options *Modify default options and pass them to modelStudio*

Description

This function returns default options for modelStudio. It is possible to modify values of this list and pass it to the options parameter in the main function. **WARNING: Editing default options may cause unintended behavior.**

Usage

```
ms_options(...)
```

Arguments

...	Options to change in the form option_name = value.

Value

list of options for modelStudio.

Options

Main options::

scale_plot `TRUE` Makes every plot the same height, ignores `bar_width`.
show_boxplot `TRUE` Display boxplots in Feature Importance and Shapley Values plots.
show_subtitle `TRUE` Should the subtitle be displayed?
subtitle `label` parameter from `explainer`.
ms_title Title of the dashboard.
ms_subtitle Subtitle of the dashboard (makes space between the title and line).
ms_margin_* Dashboard margins. Change `margin_top` for more `ms_subtitle` space.
margin_* Plot margins. Change `margin_left` for longer/shorter axis labels.
w `420` in px. Inner plot width.
h `280` in px. Inner plot height.
bar_width `16` in px. Default width of bars for all plots, ignored when `scale_plot = TRUE`.
line_size `2` in px. Default width of lines for all plots.
point_size `3` in px. Default point radius for all plots.
[bar,line,point _color] `[#46bac2,#46bac2,#371ea3]`
positive_color `#8bdcbe` for Break Down and Shapley Values bars.
negative_color `#f05a71` for Break Down and Shapley Values bars.
default_color `#371ea3` for Break Down bar and highlighted line.

Plot-specific options:: `**` is a two letter code unique to each plot, might be one of `[bd,sv,cp,fi,pd,ad,rv,fd,tv,at]`.

****_title** Plot-specific title. Default varies.
****_subtitle** Plot-specific subtitle. Default is `subtitle`.
****_axis_title** Plot-specific axis title. Default varies.
****_bar_width** Plot-specific width of bars. Default is `bar_width`, ignored when `scale_plot = TRUE`.
****_line_size** Plot-specific width of lines. Default is `line_size`.
****_point_size** Plot-specific point radius. Default is `point_size`.
****_*_color** Plot-specific `[bar,line,point]` color. Default is `[bar,line,point]_color`.

References

- The input object is implemented in **DALEX**
- Feature Importance, Ceteris Paribus, Partial Dependence and Accumulated Dependence explanations are implemented in **ingredients**
- Break Down and Shapley Values explanations are implemented in **iBreakDown**

See Also

Vignettes: **modelStudio - R & Python examples** and **modelStudio - perks and features**

Examples

```
library("DALEX")
library("modelStudio")

# fit a model
model_apartments <- glm(m2.price ~. , data = apartments)

# create an explainer for the model
explainer_apartments <- explain(model_apartments,
                                data = apartments,
                                y = apartments$m2.price)

# pick observations
new_observation <- apartments[1:2,]
rownames(new_observation) <- c("ap1","ap2")

# modify default options
new_options <- ms_options(
  show_subtitle = TRUE,
  bd_subtitle = "Hello World",
  line_size = 5,
  point_size = 9,
  line_color = "pink",
  point_color = "purple",
  bd_positive_color = "yellow",
  bd_negative_color = "orange"
)

# make a studio for the model
modelStudio(explainer_apartments,
            new_observation,
            options = new_options,
            N = 200,  B = 5) # faster example
```

`ms_update_observations` *Update the observations of a modelStudio object*

Description

This function calculates local explanations on new observations and adds them to the `modelStudio` object.

Usage

```
ms_update_observations(
  object,
  explainer,
```

```
  new_observation = NULL,
  new_observation_y = NULL,
  max_features = 10,
  B = 10,
  show_info = TRUE,
  parallel = FALSE,
  widget_id = NULL,
  overwrite = FALSE,
  ...
)
```

Arguments

`object`	A `modelStudio` created with `modelStudio()`.
`explainer`	An `explainer` created with `DALEX::explain()`.
`new_observation`	New observations with columns that correspond to variables used in the model.
`new_observation_y`	True label for `new_observation` (optional).
`max_features`	Maximum number of features to be included in BD and SV plots. Default is `10`.
`B`	Number of permutation rounds used for calculation of SV and FI. Default is `10`. See **vignette**
`show_info`	Verbose a progress on the console. Default is `TRUE`.
`parallel`	Speed up the computation using `parallelMap::parallelMap()`. See **vignette**. This might interfere with showing progress using `show_info`.
`widget_id`	Use an explicit element ID for the widget (rather than an automatically generated one). Useful e.g. when using `modelStudio` with Shiny. See **vignette**.
`overwrite`	Overwrite existing observations and their explanations. Default is `FALSE` which means add new observations to the existing ones.
`...`	Other parameters.

Value

An object of the `r2d3`, `htmlwidget`, `modelStudio` class.

References

- The input object is implemented in **DALEX**
- Feature Importance, Ceteris Paribus, Partial Dependence and Accumulated Dependence explanations are implemented in **ingredients**
- Break Down and Shapley Values explanations are implemented in **iBreakDown**

See Also

Vignettes: **modelStudio - R & Python examples** and **modelStudio - perks and features**

Examples

```
library("DALEX")
library("modelStudio")

# fit a model
model_titanic <- glm(survived ~., data = titanic_imputed, family = "binomial")

# create an explainer for the model
explainer_titanic <- explain(model_titanic,
                             data = titanic_imputed,
                             y = titanic_imputed$survived)

# make a studio for the model
ms <- modelStudio(explainer_titanic,
                  N = 200,  B = 5) # faster example



# add new observations
ms <- ms_update_observations(ms,
                             explainer_titanic,
                             new_observation = titanic_imputed[100:101,],
                             new_observation_y = titanic_imputed$survived[100:101])
ms



# overwrite the observations with new ones
ms <- ms_update_observations(ms,
                             explainer_titanic,
                             new_observation = titanic_imputed[100:101,],
                             overwrite = TRUE)
ms
```

`ms_update_options` *Update the options of a modelStudio object*

Description

This function updates the options of a `modelStudio` object. **WARNING: Editing default options may cause unintended behavior.**

Usage

```
ms_update_options(object, ...)
```

Arguments

object	A modelStudio created with modelStudio().
...	Options to change in the form option_name = value, e.g. time = 0, facet_dim = c(1,2).

Value

An object of the r2d3, htmlwidget, modelStudio class.

Options

Main options::

scale_plot TRUE Makes every plot the same height, ignores bar_width.

show_boxplot TRUE Display boxplots in Feature Importance and Shapley Values plots.

show_subtitle TRUE Should the subtitle be displayed?

subtitle label parameter from explainer.

ms_title Title of the dashboard.

ms_subtitle Subtitle of the dashboard (makes space between the title and line).

ms_margin_* Dashboard margins. Change margin_top for more ms_subtitle space.

margin_* Plot margins. Change margin_left for longer/shorter axis labels.

w 420 in px. Inner plot width.

h 280 in px. Inner plot height.

bar_width 16 in px. Default width of bars for all plots, ignored when scale_plot = TRUE.

line_size 2 in px. Default width of lines for all plots.

point_size 3 in px. Default point radius for all plots.

[bar,line,point _color] [#46bac2,#46bac2,#371ea3]

positive_color #8bdcbe for Break Down and Shapley Values bars.

negative_color #f05a71 for Break Down and Shapley Values bars.

default_color #371ea3 for Break Down bar and highlighted line.

Plot-specific options:: ** is a two letter code unique to each plot, might be one of [bd,sv,cp,fi,pd,ad,rv,fd,tv,at].

****_title** Plot-specific title. Default varies.

****_subtitle** Plot-specific subtitle. Default is subtitle.

****_axis_title** Plot-specific axis title. Default varies.

****_bar_width** Plot-specific width of bars. Default is bar_width, ignored when scale_plot = TRUE.

****_line_size** Plot-specific width of lines. Default is line_size.

****_point_size** Plot-specific point radius. Default is point_size.

****_*_color** Plot-specific [bar,line,point] color. Default is [bar,line,point]_color.

References

- The input object is implemented in **DALEX**
- Feature Importance, Ceteris Paribus, Partial Dependence and Accumulated Dependence explanations are implemented in **ingredients**
- Break Down and Shapley Values explanations are implemented in **iBreakDown**

See Also

Vignettes: **modelStudio - R & Python examples** and **modelStudio - perks and features**

Examples

```
library("DALEX")
library("modelStudio")

# fit a model
model_titanic <- glm(survived ~., data = titanic_imputed, family = "binomial")

# create an explainer for the model
explainer_titanic <- explain(model_titanic,
                             data = titanic_imputed,
                             y = titanic_imputed$survived)

# make a studio for the model
ms <- modelStudio(explainer_titanic,
                  N = 200,  B = 5) # faster example

# update the options
new_ms <- ms_update_options(ms,
                            time = 0,
                            facet_dim = c(1,2),
                            margin_left = 150)
new_ms
```

Index