Package ‘modelplotr’

October 13, 2020

Type Package

Title Plots to Evaluate the Business Performance of Predictive Models

Version 1.1.0

URL https://github.com/jurrr/modelplotr

BugReports https://github.com/jurrr/modelplotr/issues

Description Plots to assess the quality of predictive models from a business perspective.
Using these plots, it can be shown how implementation of the model will impact business targets like response on a campaign or return on investment. Different scopes can be selected: compare models, compare datasets or compare target class values and various plot customization and highlighting options are available.
targets like response on a campaign. Different scopes can be selected: compare models, compare datasets or compare target class values and various plot customization and highlighting options are available.

Depends R (>= 3.1.0)

License GPL-3

Encoding UTF-8

LazyData true

Imports ggplot2 (>= 2.2.1), gridExtra (>= 2.3.0), magrittr (>= 1.5.0), dplyr (>= 0.7.7), RColorBrewer (>= 1.1.2), ggfittext (>= 0.6.0), scales (>= 1.0.0), rlang (>= 0.3.1)

RoxygenNote 7.1.1

Suggests mlr (>= 2.12.1), caret (>= 6.0), randomForest (>= 4.6.14), nnet(>= 7.3-12), e1071, h2o, keras, knitr, rmarkdown, testthat, xgboost, stringr, kableExtra, lattice, ranger, glmnet

VignetteBuilder knitr

NeedsCompilation no

Author Jurriaan Nagelkerke [aut, cre],
Pieter Marcus [aut]

Maintainer Jurriaan Nagelkerke <jurriaan.nagelkerke@gmail.com>

Repository CRAN

Date/Publication 2020-10-13 04:20:05 UTC

R topics documented:

`aggregate_over_ntiles` *Build a dataframe with aggregated evaluation measures*

Description

Build a dataframe with aggregated actuals and predictions. Records in this dataframe represent the unique combinations of models [m], datasets [d], targetvalues [t] and ntiles [n]. The size of this dataframe therefore is (m*d*t*n) rows and 23 columns.

In most cases, you do not need to use function since the `plotting_scope` function will call this function automatically.

Usage

```
aggregate_over_ntiles(prepared_input)
```

Arguments

`prepared_input`	Dataframe resulting from function `prepare_scores_and_ntiles` or a data frame that meets requirements as specified in the section below: **When you build input for aggregate_over_ntiles() yourself** .

Value

Dataframe object is returned, containing:

column	type	definition
model_label	String	Name of the model object
dataset_label	Factor	Datasets to include in the plot as factor levels
target_class	String or Integer	Target classes to include in the plot
ntile	Integer	Ntile groups based on model probability for target class
neg	Integer	Number of cases not belonging to target class in dataset in ntile
pos	Integer	Number of cases belonging to target class in dataset in ntile
tot	Integer	Total number of cases in dataset in ntile
pct	Decimal	Percentage of cases in dataset in ntile that belongs to target class (pos/tot)
negtot	Integer	Total number of cases not belonging to target class in dataset
postot	Integer	Total number of cases belonging to target class in dataset
tottot	Integer	Total number of cases in dataset
pcttot	Decimal	Percentage of cases in dataset that belongs to target class (postot / tottot)
cumneg	Integer	Cumulative number of cases not belonging to target class in dataset from ntile 1 up until nt
cumpos	Integer	Cumulative number of cases belonging to target class in dataset from ntile 1 up until ntile
cumtot	Integer	Cumulative number of cases in dataset from ntile 1 up until ntile
cumpct	Integer	Cumulative percentage of cases belonging to target class in dataset from ntile 1 up until nti
gain	Decimal	Gains value for dataset for ntile (pos/postot)
cumgain	Decimal	Cumulative gains value for dataset for ntile (cumpos/postot)
gain_ref	Decimal	Lower reference for gains value for dataset for ntile (ntile/#ntiles)
gain_opt	Decimal	Upper reference for gains value for dataset for ntile
lift	Decimal	Lift value for dataset for ntile (pct/pcttot)
cumlift	Decimal	Cumulative lift value for dataset for ntile ((cumpos/cumtot)/pcttot)
cumlift_ref	Decimal	Reference value for Cumulative lift value (constant: 1)

When you build input for aggregate_over_ntiles() yourself

To make plots with modelplotr, is not required to use the function prepare_scores_and_ntiles to generate the required input data. You can create your own dataframe containing actuals and probabilities and ntiles (1st ntile = (1/#ntiles) percent with highest model probability, last ntile = (1/#ntiles) percent with lowest probability according to model) , In that case, make sure the input dataframe contains the folowing columns & formats:

column	type	definition
model_label	Factor	Name of the model object
dataset_label	Factor	Datasets to include in the plot as factor levels
y_true	Factor	Target with actual values
prob_[tv1]	Decimal	Probability according to model for target value 1
prob_[tv2]	Decimal	Probability according to model for target value 2
...	...	...
prob_[tvn]	Decimal	Probability according to model for target value n
ntl_[tv1]	Integer	Ntile based on probability according to model for target value 1
ntl_[tv2]	Integerl	Ntile based on probability according to model for target value 2
...	...	...
ntl_[tvn]	Integer	Ntile based on probability according to model for target value n

See `build_input_yourself` for an example to build the required input yourself.

See Also

`modelplotr` for generic info on the package `moddelplotr`

`vignette('modelplotr')`

`prepare_scores_and_ntiles` for details on the function `prepare_scores_and_ntiles` that generates the required input.

`plotting_scope` for details on the function `plotting_scope` that filters the output of `aggregate_over_ntiles` to prepare it for the required evaluation.

`build_input_yourself` for an example to build the required input yourself.

`https://github.com/modelplot/modelplotr` for details on the package

`https://modelplot.github.io/` for our blog on the value of the model plots

Examples

```
## Not run:
# load example data (Bank clients with/without a term deposit - see ?bank_td for details)
data("bank_td")

# prepare data for training model for binomial target has_td and train models
train_index =  sample(seq(1, nrow(bank_td)),size = 0.5*nrow(bank_td) ,replace = FALSE)
train = bank_td[train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
test = bank_td[-train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]

#train models using mlr...
trainTask <- mlr::makeClassifTask(data = train, target = "has_td")
testTask <- mlr::makeClassifTask(data = test, target = "has_td")
mlr::configureMlr() # this line is needed when using mlr without loading it (mlr::)
task = mlr::makeClassifTask(data = train, target = "has_td")
lrn = mlr::makeLearner("classif.randomForest", predict.type = "prob")
rf = mlr::train(lrn, task)
lrn = mlr::makeLearner("classif.multinom", predict.type = "prob")
mnl = mlr::train(lrn, task)
#... or train models using caret...
# setting caret cross validation, here tuned for speed (not accuracy!)
fitControl <- caret::trainControl(method = "cv",number = 2,classProbs=TRUE)
# random forest using ranger package, here tuned for speed (not accuracy!)
rf = caret::train(has_td ~.,data = train, method = "ranger",trControl = fitControl,
                tuneGrid = expand.grid(.mtry = 2,.splitrule = "gini",.min.node.size=10))
# mnl model using glmnet package
mnl = caret::train(has_td ~.,data = train, method = "glmnet",trControl = fitControl)
#... or train models using h2o...
h2o::h2o.init()
h2o::h2o.no_progress()
h2o_train = h2o::as.h2o(train)
h2o_test = h2o::as.h2o(test)
gbm <- h2o::h2o.gbm(y = "has_td",
                          x = setdiff(colnames(train), "has_td"),
                          training_frame = h2o_train,
                          nfolds = 5)
#... or train models using keras.
x_train <- as.matrix(train[,-1]); y=train[,1]; y_train <- keras::to_categorical(as.numeric(y)-1);
`%>%` <- magrittr::`%>%`
nn <- keras::keras_model_sequential() %>%
keras::layer_dense(units = 16,kernel_initializer = "uniform",activation = 'relu',
                   input_shape = NCOL(x_train))%>%
  keras::layer_dense(units = 16,kernel_initializer = "uniform", activation='relu') %>%
  keras::layer_dense(units = length(levels(train[,1])),activation='softmax')
nn %>% keras::compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=c('accuracy'))
nn %>% keras::fit(x_train,y_train,epochs = 20,batch_size = 1028,verbose=0)

# preparation steps
scores_and_ntiles <- prepare_scores_and_ntiles(datasets=list("train","test"),
                      dataset_labels = list("train data","test data"),
                      models = list("rf","mnl", "gbm","nn"),
                      model_labels = list("random forest","multinomial logit",
                                "gradient boosting machine","artificial neural network"),
                      target_column="has_td")
aggregated <- aggregate_over_ntiles(prepared_input=scores_and_ntiles)
head(aggregated)
plot_input <- plotting_scope(prepared_input = aggregated)
head(plot_input)

## End(Not run)
```

bank_td — *Bank clients that have/have not subscribed a term deposit.*

Description

A dataset containing some customer characteristics for clients of a bank that have/have not subscribed a term deposit.

Usage

```
bank_td
```

Format

A data frame with 2000 rows and 6 variables:

has_td has the client subscribed a term deposit? Values: "term deposit", "no". This variable is used as the binary target variable in examples for the modelplotr package.

td_type what type of term deposit did the client subscribe? Values: "no.td", "td.type.A", "td.type.B", "td.type.C". This variable is used as the multinomial target variable in examples for the modelplotr package.

duration last contact duration, in seconds (numeric)

campaign number of contacts performed during this campaign and for this client

pdays number of days that passed by after the client was last contacted from a previous campaign

previous number of contacts performed before this campaign and for this client (numeric)

euribor3m euribor 3 month rate

Source

This dataset is a subset of the dataset made available by the University of California, Irvine. The complete dataset is available here: `https://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank-additional.zip`

---

build_input_yourself *Example: build required input from a custom model*

Description

It's very easy to apply modelplotr to predictive models that are developed in caret, mlr, h2o or keras. However, also for models that are developed differently, even those built outside of R, it only takes a bit more work to use modelplotr on top of these models. In this section we introduce the required format and an example.

When you build input for plotting_scope() yourself

To make plots with modelplotr, is not required to use the function prepare_scores_and_ntiles to generate the required input data. You can create your own dataframe containing actuals and probabilities and ntiles (1st ntile = (1/#ntiles) percent with highest model probability, last ntile = (1/#ntiles) percent with lowest probability according to model) , In that case, make sure the input dataframe contains the folowing columns & formats:

column	**type**	**definition**
model_label	Factor	Name of the model object
dataset_label	Factor	Datasets to include in the plot as factor levels
y_true	Factor	Target with actual values
prob_[tv1]	Decimal	Probability according to model for target value 1
prob_[tv2]	Decimal	Probability according to model for target value 2
...	...	...
prob_[tvn]	Decimal	Probability according to model for target value n
ntl_[tv1]	Integer	Ntile based on probability according to model for target value 1
ntl_[tv2]	Integerl	Ntile based on probability according to model for target value 2
...	...	...
ntl_[tvn]	Integer	Ntile based on probability according to model for target value n

Examples

```
# load example data (Bank clients with/without a term deposit - see ?bank_td for details)
data("bank_td")
library(dplyr)
# prepare data for training model for binomial target has_td and train models
```

```
train_index =  sample(seq(1, nrow(bank_td)),size = 0.5*nrow(bank_td) ,replace = FALSE)
train = bank_td[train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
test = bank_td[-train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]

#train logistic regression model with stats package
glm.model <- glm(has_td ~.,family=binomial(link='logit'),data=train)
#score model
prob_no.term.deposit <- stats::predict(glm.model,newdata=train,type='response')
prob_term.deposit <- 1-prob_no.term.deposit
#set number of ntiles
ntiles = 10
# determine cutoffs
cutoffs = c(stats::quantile(prob_term.deposit,probs = seq(0,1,1/ntiles),na.rm = TRUE))
#calculate ntile values
ntl_term.deposit <- (ntiles+1)-as.numeric(cut(prob_term.deposit,breaks=cutoffs,include.lowest=TRUE))
ntl_no.term.deposit <- (ntiles+1)-ntl_term.deposit
# create scored data frame
scores_and_ntiles <- train %>%
    select(has_td) %>%
    mutate(model_label=factor('logistic regression'),
           dataset_label=factor('train data'),
           y_true=factor(has_td),
           prob_term.deposit = prob_term.deposit,
           prob_no.term.deposit = prob_no.term.deposit,
           ntl_term.deposit = ntl_term.deposit,
           ntl_no.term.deposit = ntl_no.term.deposit) %>%
    select(-has_td)

# add test data
#score model on test data
prob_no.term.deposit <- stats::predict(glm.model,newdata=test,type='response')
prob_term.deposit <- 1-prob_no.term.deposit
#set number of ntiles
ntiles = 10
# determine cutoffs
cutoffs = c(stats::quantile(prob_term.deposit,probs = seq(0,1,1/ntiles),na.rm = TRUE))
#calculate ntile values
ntl_term.deposit <- (ntiles+1)-as.numeric(cut(prob_term.deposit,breaks=cutoffs,include.lowest=TRUE))
ntl_no.term.deposit <- (ntiles+1)-ntl_term.deposit
scores_and_ntiles <- scores_and_ntiles %>%
  rbind(
   test %>%
    select(has_td) %>%
    mutate(model_label=factor('logistic regression'),
           dataset_label=factor('test data'),
           y_true=factor(has_td),
           prob_term.deposit = prob_term.deposit,
           prob_no.term.deposit = prob_no.term.deposit,
           ntl_term.deposit = ntl_term.deposit,
           ntl_no.term.deposit = ntl_no.term.deposit) %>%
    select(-has_td)
    )
```

```
plot_input <- plotting_scope(prepared_input = scores_and_ntiles,scope='compare_datasets')
plot_cumgains()
```

`customize_plot_text` *Customize textual elements of the plots*

Description

Function to overrule the default textual elements in the plots, like title, subtitle, axis labels and annotation texts when the highlighting parameter `highlight_ntile` is specified.

Usage

```
customize_plot_text(plot_input = plot_input)
```

Arguments

`plot_input`	Dataframe. Dataframe needs to be created with `plotting_scope` or else meet required input format.

Value

List with default values for all textual elements of the plots.

How to customize textual elements of plots

All textual parts of the plots can be customized, for instance to translate textual elements to another language or to change the annotation text that is added with the `highlight_ntile` parameter. Once you have created the `plot_input` dataframe using `plotting_Scope`, you can run this `customize_plot_text()` function. It returns a list, containing all textual elements of the plots, including annotation texts. For instance, run

```
my_plot_text <-customize_plot_text(plot_input = plot_input)
```

The list contains plot-specific elements (e.g. `...$cumgains$...`)).
Now, you can change the textual elements by overriding the element(s) you want to customize. For instance, if you want to change the textual elements of the gains plot to Dutch:

```
my_plot_text$gains$plottitle <-'Cumulatieve Gains grafiek'
my_plot_text$gains$x_axis_label <-'Deciel'
my_plot_text$gains$y_axis_label <-'cumulatieve gains'
my_plot_text$cumgains$optimal_gains_label <-'maximale gains'
my_plot_text$cumgains$minimal_gains_label <-'minimale gains'
plot_cumgains(custom_plot_text = my_plot_text)
```

To change the annotation text, use the placeholders starting with '&' to dynamically include:

palaceholder	placeholder value
`&NTL`	ntile specified with parameter `highlight_ntile`.
`&PCTNTL`	Total percentage of dataset selected up until specified ntile.
`&MDL`	Selected model label(s).
`&DS`	Selected dataset label(s).
`&YVAL`	Selected target class (Y-value).
`&VALUE`	The plot specific value at specified ntile. Eg. Cumulative gains, Rumulative lift, Response, Cumulative respon

For instance, to translate the gains plot annotation text to Dutch:
`my_plot_text$cumlift$annotationtext <-"Door &PCTNTL met de hoogste modelkans volgens model &MDL in &DS te selecteren is deze selectie van &YVAL observaties &CUMLIFT keer beter dan een random selectie."`
`plot_cumlift(highlight_ntile=3,custom_plot_text=my_plot_text)`

See Also

`modelplotr` for generic info on the package `moddelplotr`

`vignette('modelplotr')`

`https://github.com/modelplot/modelplotr` for details on the package

`https://modelplot.github.io/` for our blog on the value of the model plots

Examples

```
# load example data (Bank clients with/without a term deposit - see ?bank_td for details)
data("bank_td")

# prepare data for training model for binomial target has_td and train models
train_index =  sample(seq(1, nrow(bank_td)),size = 0.5*nrow(bank_td) ,replace = FALSE)
train = bank_td[train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
test = bank_td[-train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]

#train models using caret... (or use mlr or H2o or keras ... see ?prepare_scores_and_ntiles)
# setting caret cross validation, here tuned for speed (not accuracy!)
fitControl <- caret::trainControl(method = "cv",number = 2,classProbs=TRUE)
# random forest using ranger package, here tuned for speed (not accuracy!)
rf = caret::train(has_td ~.,data = train, method = "ranger",trControl = fitControl,
                tuneGrid = expand.grid(.mtry = 2,.splitrule = "gini",.min.node.size=10))
# mnl model using glmnet package
mnl = caret::train(has_td ~.,data = train, method = "glmnet",trControl = fitControl)

# load modelplotr
library(modelplotr)

# transform datasets and model objects to input for modelplotr
scores_and_ntiles <- prepare_scores_and_ntiles(datasets=list("train","test"),
                         dataset_labels = list("train data","test data"),
                         models = list("rf","mnl"),
                         model_labels = list("random forest","multinomial logit"),
                         target_column="has_td",
                              ntiles=100)

# set scope for analysis (default: no comparison)
plot_input <- plotting_scope(prepared_input = scores_and_ntiles)

# customize all textual elements of plots
mytexts <- customize_plot_text(plot_input = plot_input)
mytexts$cumresponse$plottitle <- 'Expected conversion rate for Campaign XYZ'
mytexts$cumresponse$plotsubtitle <- 'proposed selection: best 15 percentiles according to our model'
mytexts$cumresponse$y_axis_label <- '% Conversion'
mytexts$cumresponse$x_axis_label <- 'percentiles (percentile = 1% of customers)'
mytexts$cumresponse$annotationtext <-
  "Selecting up until the &NTL percentile with model &MDL has an expected conversion rate of &VALUE"
plot_cumresponse(data=plot_input,custom_plot_text = mytexts,highlight_ntile = 15)
```

`modelplotr` *modelplotr: Plots to Evaluate the Business Performance of Predictive Models.*

Description

Plots to evaluate the business performance of predictive models in R. A number of widely used plots to assess the quality of a predictive model from a business perspective can easily be created. Using these plots, it can be shown how implementation of the model will impact business targets like response on a campaign or return on investment. It's very easy to apply modelplotr to predictive models that are developed in caret, mlr, h2o or keras. For other models, even those built outside of R, an instruction is included. The modelplotr package provides three categories of important functions: datapreparation, plot parameterization and plotting.

Datapreparation functions

The datapreparation functions are:

`prepare_scores_and_ntiles` Function that builds a dataframe that contains actuals and predictions on the target variable for each dataset in `datasets` and each model in `models`. As inputs, it takes dataframes to score and model objects created with **caret**, **mlr**, **h2o** or **keras**. Specifically for keras models, built with keras_model_sequential() or with the keras functional API, there is the `prepare_scores_and_ntiles_keras` function. To use modelplotr on top of models created otherwise, even models built outside r, see `aggregate_over_ntiles`

`plotting_scope` Function that creates a dataframe in the required format for all modelplotr plots, relevant to the selected scope of evaluation. Each record in this dataframe represents a unique combination of datasets, models, target classes and ntiles. As an input, plotting_scope can handle both a dataframe created with `aggregate_over_ntiles` as well as a dataframe created with `prepare_scores_and_ntiles` (or with `prepare_scores_and_ntiles_keras` or created otherwise, with similar layout).

`aggregate_over_ntiles` Function that aggregates the output of `prepare_scores_and_ntiles` to create a dataframe with aggregated actuals and predictions. Each record in this dataframe represents a unique combination of datasets, models, target classes and ntiles. In most cases, you do not need to use function since the `plotting_scope` function will call this function automatically.

Parameterization functions

Most parameterization functions are internal functions. However, one is available for customization:

`customize_plot_text` Function that returns a list that contains all textual elements for all plots that modelplotr can create. By changing the elements in this list - simply by overwriting values - and then including this list with the `custom_plot_text` parameter in plot functions, plot texts can easily be customized to meet your (language) preferences

Plotting functions

The plotting functions are:

`plot_cumgains` Generates the cumulative gains plot. This plot, often referred to as the gains chart, helps answering the question: ***When we apply the model and select the best X ntiles, what percentage of the actual target class observations can we expect to target?***

`plot_cumlift` Generates the cumulative lift plot, often referred to as lift plot or index plot, helps you answer the question: ***When we apply the model and select the best X ntiles, how many times better is that than using no model at all?***

`plot_response` Generates the response plot. It plots the percentage of target class observations per ntile. It can be used to answer the following business question: ***When we apply the model and select ntile X, what is the expected percentage of target class observations in that ntile?***

`plot_cumresponse` Generates the cumulative response plot. It plots the cumulative percentage of target class observations up until that ntile. It helps answering the question: ***When we apply the model and select up until ntile X, what is the expected percentage of target class observations in the selection?***

`plot_multiplot` Generates a canvas with all four evaluation plots - cumulative gains, cumulative lift, response and cumulative response - combined on one canvas

`plot_costsrevs` It plots the cumulative costs and revenues up until that ntile when the model is used for campaign selection. It can be used to answer the following business question: ***When we apply the model and select up until ntile X, what are the expected costs and revenues of the campaign?***

`plot_profit` Generates the Profit plot. It plots the cumulative profit up until that ntile when the model is used for campaign selection. It can be used to answer the following business question: ***When we apply the model and select up until ntile X, what is the expected profit of the campaign?***

`plot_roi` Generates the Return on Investment plot. It plots the cumulative revenues as a percentage of investments up until that ntile when the model is used for campaign selection. It can be used to answer the following business question: ***When we apply the model and select up until ntile X, what is the expected investment of the campaign?***

Author(s)

Jurriaan Nagelkerke <jurriaan.nagelkerke@gmail.com> [aut, cre]

Pieter Marcus <pieter.marcus@persgroep.net> [aut]

See Also

vignette('modelplotr')

https://github.com/modelplot/modelplotr for details on the package

https://modelplot.github.io/ for our blog posts on using modelplotr

Examples

```
## Not run:
# load example data (Bank clients with/without a term deposit - see ?bank_td for details)
data("bank_td")

# prepare data for training model for binomial target has_td and train models
train_index =  sample(seq(1, nrow(bank_td)),size = 0.5*nrow(bank_td) ,replace = FALSE)
train = bank_td[train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
test = bank_td[-train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]

#train models using caret... (or use mlr or H2o or keras ... see ?prepare_scores_and_ntiles)
# setting caret cross validation, here tuned for speed (not accuracy!)
fitControl <- caret::trainControl(method = "cv",number = 2,classProbs=TRUE)
# random forest using ranger package, here tuned for speed (not accuracy!)
rf = caret::train(has_td ~.,data = train, method = "ranger",trControl = fitControl,
                tuneGrid = expand.grid(.mtry = 2,.splitrule = "gini",.min.node.size=10))
# mnl model using glmnet package
mnl = caret::train(has_td ~.,data = train, method = "glmnet",trControl = fitControl)

# load modelplotr
library(modelplotr)

# transform datasets and model objects to input for modelplotr
scores_and_ntiles <- prepare_scores_and_ntiles(datasets=list("train","test"),
                         dataset_labels = list("train data","test data"),
                         models = list("rf","mnl"),
                         model_labels = list("random forest","multinomial logit"),
                         target_column="has_td",
                         ntiles=100)

# set scope for analysis (default: no comparison)
plot_input <- plotting_scope(prepared_input = scores_and_ntiles)
head(plot_input)

# ALL PLOTS, with defaults
plot_cumgains(data=plot_input)
plot_cumlift(data=plot_input)
plot_response(data=plot_input)
plot_cumresponse(data=plot_input)
plot_multiplot(data=plot_input)
# financial plots - these need some financial parameters
plot_costsrevs(data=plot_input,fixed_costs=1000,variable_costs_per_unit=10,profit_per_unit=50)
plot_profit(data=plot_input,fixed_costs=1000,variable_costs_per_unit=10,profit_per_unit=50)
plot_roi(data=plot_input,fixed_costs=1000,variable_costs_per_unit=10,profit_per_unit=50)
```

```
# CHANGING THE SCOPE OF ANALYSIS
# changing the scope - compare models:
plot_input <- plotting_scope(prepared_input = scores_and_ntiles,scope="compare_models")
plot_cumgains(data=plot_input)
# changing the scope - compare datasets:
plot_input <- plotting_scope(prepared_input = scores_and_ntiles,scope="compare_datasets")
plot_roi(data = plot_input,fixed_costs=1000,variable_costs_per_unit=10,profit_per_unit=50)
# changing the scope - compare target classes:
plot_input <- plotting_scope(prepared_input = scores_and_ntiles,scope="compare_targetclasses")
plot_response(data=plot_input)
# HIGHLIGHTING OPTIONS
plot_input <- plotting_scope(prepared_input = scores_and_ntiles,
                         scope = 'compare_datasets',select_model_label = 'random forest')
plot_cumgains(data=plot_input,highlight_ntile=20)
plot_cumlift(data=plot_input,highlight_ntile=20,highlight_how = 'plot')
plot_response(data=plot_input,highlight_ntile=20,highlight_how = 'text')
plot_cumresponse(data=plot_input,highlight_ntile=20,highlight_how = 'plot_text')
plot_costsrevs(data=plot_input,fixed_costs = 1000,variable_costs_per_unit = 10,
               profit_per_unit = 50,highlight_ntile='max_roi')
plot_profit(data=plot_input,fixed_costs = 1500,variable_costs_per_unit = 10,profit_per_unit = 50)
plot_roi(data=plot_input,fixed_costs = 1500,variable_costs_per_unit = 10,profit_per_unit = 50)

# OTHER PLOT CUSTOMIZATIONS
# customize line colors
plot_input <- plotting_scope(prepared_input = scores_and_ntiles,scope = 'compare_models')
plot_cumgains(data=plot_input,custom_line_colors = c('pink','navyblue'))
# customize all textual elements of plots
plot_input <- plotting_scope(prepared_input = scores_and_ntiles)
mytexts <- customize_plot_text(plot_input = plot_input)
mytexts$cumresponse$plottitle <- 'Expected conversion rate for Campaign XYZ'
mytexts$cumresponse$plotsubtitle <- 'proposed selection: best 15 percentiles according to our model'
mytexts$cumresponse$y_axis_label <- '% Conversion'
mytexts$cumresponse$x_axis_label <- 'percentiles (percentile = 1% of customers)'
mytexts$cumresponse$annotationtext <-
"Selecting up until the &NTL percentile with model &MDL has an expected conversion rate of &VALUE"
plot_cumresponse(data=plot_input,custom_plot_text = mytexts,highlight_ntile = 15)

## End(Not run)
```

`plotting_scope` *Build dataframe with formatted input for all plots.*

Description

Build a dataframe in the required format for all modelplotr plots, relevant to the selected scope of evaluation. Each record in this dataframe represents a unique combination of datasets, models, target classes and ntiles. As an input, plotting_scope can handle both a dataframe created with `aggregate_over_ntiles` as well as a dataframe created with `prepare_scores_and_ntiles` (or created otherwise with similar layout). There are four perspectives:

"no_comparison" (default) In this perspective, you're interested in the performance of one model on one dataset for one target class. Therefore, only one line is plotted in the plots. The parameters `select_model_label`, `select_dataset_label` and `select_targetclass` determine which group is plotted. When not specified, the first alphabetic model, the first alphabetic dataset and the smallest (when `select_smallest_targetclass=TRUE`) or first alphabetic target value are selected

"compare_models" In this perspective, you're interested in how well different models perform in comparison to each other on the same dataset and for the same target value. This results in a comparison between models available in ntiles_aggregate$model_label for a selected dataset (default: first alphabetic dataset) and for a selected target value (default: smallest (when `select_smallest_targetclass=TRUE`) or first alphabetic target value).

"compare_datasets" In this perspective, you're interested in how well a model performs in different datasets for a specific model on the same target value. This results in a comparison between datasets available in ntiles_aggregate$dataset_label for a selected model (default: first alphabetic model) and for a selected target value (default: smallest (when `select_smallest_targetclass=TRUE`) or first alphabetic target value).

"compare_targetclasses" In this perspective, you're interested in how well a model performs for different target values on a specific dataset.This resuls in a comparison between target classes available in ntiles_aggregate$target_class for a selected model (default: first alphabetic model) and for a selected dataset (default: first alphabetic dataset).

Usage

```
plotting_scope(
  prepared_input,
  scope = "no_comparison",
  select_model_label = NA,
  select_dataset_label = NA,
  select_targetclass = NA,
  select_smallest_targetclass = TRUE
)
```

Arguments

`prepared_input`	Dataframe. Dataframe created with `prepare_scores_and_ntiles` or dataframe created with `aggregate_over_ntiles` or a dataframe that is created otherwise with similar layout as the output of these functions (see ?prepare_scores_and_ntiles and ?aggregate_over_ntiles for layout details).
`scope`	String. Evaluation type of interest. Possible values: "compare_models","compare_datasets", "compare_targetclasses","no_comparison". Default is NA, equivalent to "no_comparison".
`select_model_label`	String. Selected model when scope is "compare_datasets" or "compare_targetclasses" or "no_comparison". Needs to be identical to model descriptions as specified in model_labels (or models when model_labels is not specified). When scope is "compare_models", select_model_label can be used to take a subset of available models.

`select_dataset_label`
: String. Selected dataset when scope is compare_models or compare_targetclasses or no_comparison. Needs to be identical to dataset descriptions as specified in dataset_labels (or datasets when dataset_labels is not specified). When scope is "compare_datasets", select_dataset_label can be used to take a subset of available datasets.

`select_targetclass`
: String. Selected target value when scope is compare_models or compare_datasets or no_comparison. Default is smallest value when select_smallest_targetclass=TRUE, otherwise first alphabetical value. When scope is "compare_targetclasses", select_targetclass can be used to take a subset of available target classes.

`select_smallest_targetclass`
: Boolean. Select the target value with the smallest number of cases in dataset as group of interest. Default is True, hence the target value with the least observations is selected.

Value

Dataframe `plot_input` is a subset of `ntiles_aggregate`.

When you build input for plotting_scope() yourself

To make plots with modelplotr, is not required to use the function prepare_scores_and_ntiles to generate the required input data. You can create your own dataframe containing actuals and probabilities and ntiles (1st ntile = (1/#ntiles) percent with highest model probability, last ntile = (1/#ntiles) percent with lowest probability according to model) , In that case, make sure the input dataframe contains the folowing columns & formats:

column	**type**	**definition**
model_label	Factor	Name of the model object
dataset_label	Factor	Datasets to include in the plot as factor levels
y_true	Factor	Target with actual values
prob_[tv1]	Decimal	Probability according to model for target value 1
prob_[tv2]	Decimal	Probability according to model for target value 2
...	...	...
prob_[tvn]	Decimal	Probability according to model for target value n
ntl_[tv1]	Integer	Ntile based on probability according to model for target value 1
ntl_[tv2]	Integerl	Ntile based on probability according to model for target value 2
...	...	...
ntl_[tvn]	Integer	Ntile based on probability according to model for target value n

See build_input_yourself for an example to build the required input yourself.

See Also

`modelplotr` for generic info on the package `moddelplotr`

`vignette('modelplotr')`

`aggregate_over_ntiles` for details on the function `aggregate_over_ntiles` that generates the required input.

prepare_scores_and_ntiles for details on the function prepare_scores_and_ntiles that generates the required input.

build_input_yourself for an example to build the required input yourself. filters the output of aggregate_over_ntiles to prepare it for the required evaluation.

https://github.com/modelplot/modelplotr for details on the package

https://modelplot.github.io/ for our blog on the value of the model plots

Examples

```
## Not run:
# load example data (Bank clients with/without a term deposit - see ?bank_td for details)
data("bank_td")

# prepare data for training model for binomial target has_td and train models
train_index =  sample(seq(1, nrow(bank_td)),size = 0.5*nrow(bank_td) ,replace = FALSE)
train = bank_td[train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
test = bank_td[-train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]

#train models using mlr...
trainTask <- mlr::makeClassifTask(data = train, target = "has_td")
testTask <- mlr::makeClassifTask(data = test, target = "has_td")
mlr::configureMlr() # this line is needed when using mlr without loading it (mlr::)
task = mlr::makeClassifTask(data = train, target = "has_td")
lrn = mlr::makeLearner("classif.randomForest", predict.type = "prob")
rf = mlr::train(lrn, task)
lrn = mlr::makeLearner("classif.multinom", predict.type = "prob")
mnl = mlr::train(lrn, task)
#... or train models using caret...
# setting caret cross validation, here tuned for speed (not accuracy!)
fitControl <- caret::trainControl(method = "cv",number = 2,classProbs=TRUE)
# random forest using ranger package, here tuned for speed (not accuracy!)
rf = caret::train(has_td ~.,data = train, method = "ranger",trControl = fitControl,
                tuneGrid = expand.grid(.mtry = 2,.splitrule = "gini",.min.node.size=10))
# mnl model using glmnet package
mnl = caret::train(has_td ~.,data = train, method = "glmnet",trControl = fitControl)
#... or train models using h2o...
h2o::h2o.init()
h2o::h2o.no_progress()
h2o_train = h2o::as.h2o(train)
h2o_test = h2o::as.h2o(test)
gbm <- h2o::h2o.gbm(y = "has_td",
                          x = setdiff(colnames(train), "has_td"),
                          training_frame = h2o_train,
                          nfolds = 5)
#... or train models using keras.
x_train <- as.matrix(train[,-1]); y=train[,1]; y_train <- keras::to_categorical(as.numeric(y)-1)
`%>%` <- magrittr::`%>%`
nn <- keras::keras_model_sequential() %>%
keras::layer_dense(units = 16,kernel_initializer = "uniform",activation = 'relu',
                   input_shape = NCOL(x_train))%>%
  keras::layer_dense(units=16,kernel_initializer="uniform",activation='relu') %>%
  keras::layer_dense(units=length(levels(train[,1])),activation='softmax')
nn %>% keras::compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=c('accuracy'))
nn %>% keras::fit(x_train,y_train,epochs = 20,batch_size = 1028,verbose=0)

# preparation steps
scores_and_ntiles <- prepare_scores_and_ntiles(datasets=list("train","test"),
                      dataset_labels = list("train data","test data"),
                      models = list("rf","mnl", "gbm","nn"),
                      model_labels = list("random forest","multinomial logit",
                                "gradient boosting machine","artificial neural network"),
                      target_column="has_td")
plot_input <- plotting_scope(prepared_input = scores_and_ntiles)
plot_cumgains(data = plot_input)
plot_cumlift(data = plot_input)
plot_response(data = plot_input)
plot_cumresponse(data = plot_input)
plot_multiplot(data = plot_input)
plot_costsrevs(data=plot_input,fixed_costs=1000,variable_costs_per_unit=10,profit_per_unit=50)
plot_profit(data=plot_input,fixed_costs=1000,variable_costs_per_unit=10,profit_per_unit=50)
plot_roi(data=plot_input,fixed_costs=1000,variable_costs_per_unit=10,profit_per_unit=50)

## End(Not run)
```

plot_costsrevs *Costs & Revenues plot*

Description

Generates the Costs & Revenues plot. It plots the cumulative costs and revenues up until that ntile when the model is used for campaign selection. It can be used to answer the following business question: ***When we apply the model and select up until ntile X, what are the expected costs and revenues of the campaign?*** Extra parameters needed for this plot are: fixed_costs, variable_costs_per_unit and profit_per_unit.

Usage

```
plot_costsrevs(
  data = plot_input,
  highlight_ntile = "max_profit",
  highlight_how = "plot_text",
  save_fig = FALSE,
  save_fig_filename = NA,
  custom_line_colors = NA,
  custom_plot_text = NULL,
  fixed_costs,
  variable_costs_per_unit,
  profit_per_unit
)
```

Arguments

`data`	Dataframe. Dataframe needs to be created with `plotting_scope` or else meet required input format.
`highlight_ntile`	Integer or string ("max_roi" or "max_profit"). Specifying the ntile at which the plot is annotated and/or performances are highlighted. Default value is `max_profit`, highlighting the ntile where difference between returns and costs (hence: profits) is greatest.
`highlight_how`	String. How to annotate the plot. Possible values: "plot_text","plot", "text". Default is "plot_text", both highlighting the ntile and value on the plot as well as in text below the plot. "plot" only highligths the plot, but does not add text below the plot explaining the plot at chosen ntile. "text" adds text below the plot explaining the plot at chosen ntile but does not highlight the plot.
`save_fig`	Logical. Save plot to file? Default = FALSE. When set to TRUE, saved plot is optimized for 36x24cm.
`save_fig_filename`	String. Filename of saved plot. Default the plot is saved as tempdir()/plotname.png.
`custom_line_colors`	Vector of Strings. Specifying colors for the lines in the plot. When not specified, colors from the RColorBrewer palet "Set1" are used.
`custom_plot_text`	List. List with customized textual elements for plot. Create a list with defaults by using `customize_plot_text` and override default values to customize.
`fixed_costs`	Numeric. Specifying the fixed costs related to a selection based on the model. These costs are constant and do not vary with selection size (ntiles).
`variable_costs_per_unit`	Numeric. Specifying the variable costs per selected unit for a selection based on the model. These costs vary with selection size (ntiles).
`profit_per_unit`	Numeric. Specifying the profit per unit in case the selected unit converts / responds positively.

Value

gtable, containing 6 grobs.

See Also

`modelplotr` for generic info on the package `moddelplotr`

`vignette('modelplotr')`

`plotting_scope` for details on the function `plotting_scope` that transforms a dataframe created with `prepare_scores_and_ntiles` or `aggregate_over_ntiles` to a dataframe in the required format for all modelplotr plots.

aggregate_over_ntiles for details on the function aggregate_over_ntiles that aggregates the output of prepare_scores_and_ntiles to create a dataframe with aggregated actuals and predictions. In most cases, you do not need to use it since the plotting_scope function will call this function automatically.

https://github.com/modelplot/modelplotr for details on the package

https://modelplot.github.io/ for our blog on the value of the model plots

Examples

```
# load example data (Bank clients with/without a term deposit - see ?bank_td for details)
data("bank_td")
# prepare data for training model for binomial target has_td and train models
train_index =  sample(seq(1, nrow(bank_td)),size = 0.5*nrow(bank_td) ,replace = FALSE)
train = bank_td[train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
test = bank_td[-train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
#train models using caret... (or use mlr or H2o or keras ... see ?prepare_scores_and_ntiles)
# setting caret cross validation, here tuned for speed (not accuracy!)
fitControl <- caret::trainControl(method = "cv",number = 2,classProbs=TRUE)
# random forest using ranger package, here tuned for speed (not accuracy!)
rf = caret::train(has_td ~.,data = train, method = "ranger",trControl = fitControl,
                tuneGrid = expand.grid(.mtry = 2,.splitrule = "gini",.min.node.size=10))
# mnl model using glmnet package
mnl = caret::train(has_td ~.,data = train, method = "glmnet",trControl = fitControl)
# load modelplotr
library(modelplotr)
# transform datasets and model objects to input for modelplotr
scores_and_ntiles <- prepare_scores_and_ntiles(datasets=list("train","test"),
                         dataset_labels = list("train data","test data"),
                         models = list("rf","mnl"),
                         model_labels = list("random forest","multinomial logit"),
                         target_column="has_td",
                         ntiles=100)
# set scope for analysis (default: no comparison)
plot_input <- plotting_scope(prepared_input = scores_and_ntiles,scope='compare_models')
plot_costsrevs(data=plot_input,fixed_costs=1000,variable_costs_per_unit= 10,profit_per_unit=50)
plot_costsrevs(data=plot_input,fixed_costs=1000,variable_costs_per_unit= 10,profit_per_unit=50,
               highlight_ntile=20)
plot_costsrevs(data=plot_input,fixed_costs=1000,variable_costs_per_unit= 10,profit_per_unit=50,
               highlight_ntile='max_roi')
plot_costsrevs(data=plot_input,fixed_costs=1000,variable_costs_per_unit= 10,profit_per_unit=50,
               highlight_ntile='max_profit')
```

---

plot_cumgains *Cumulative gains plot*

Description

Generates the cumulative gains plot. This plot, often referred to as the gains chart, helps answering the question: ***When we apply the model and select the best X ntiles, what percentage of the actual target class observations can we expect to target?***

Usage

```
plot_cumgains(
  data = plot_input,
  highlight_ntile = NA,
  highlight_how = "plot_text",
  save_fig = FALSE,
  save_fig_filename = NA,
  custom_line_colors = NA,
  custom_plot_text = NULL
)
```

Arguments

`data`	Dataframe. Dataframe needs to be created with `plotting_scope` or else meet required input format.
`highlight_ntile`	Integer. Specifying the ntile at which the plot is annotated and/or performances are highlighted.
`highlight_how`	String. How to annotate the plot. Possible values: "plot_text","plot", "text". Default is "plot_text", both highlighting the ntile and value on the plot as well as in text below the plot. "plot" only highligths the plot, but does not add text below the plot explaining the plot at chosen ntile. "text" adds text below the plot explaining the plot at chosen ntile but does not highlight the plot.
`save_fig`	Logical. Save plot to file? Default = FALSE. When set to TRUE, saved plots are optimized for 18x12cm.
`save_fig_filename`	String. Filename of saved plot. Default the plot is saved as tempdir()/plotname.png.
`custom_line_colors`	Vector of Strings. Specifying colors for the lines in the plot. When not specified, colors from the RColorBrewer palet "Set1" are used.
`custom_plot_text`	List. List with customized textual elements for plot. Create a list with defaults by using `customize_plot_text` and override default values to customize.

Value

ggplot object. Cumulative gains plot.

See Also

`modelplotr` for generic info on the package `moddelplotr`

`vignette('modelplotr')`

`plotting_scope` for details on the function `plotting_scope` that transforms a dataframe created with `prepare_scores_and_ntiles` or `aggregate_over_ntiles` to a dataframe in the required format for all modelplotr plots.

aggregate_over_ntiles for details on the function aggregate_over_ntiles that aggregates the output of prepare_scores_and_ntiles to create a dataframe with aggregated actuals and predictions. In most cases, you do not need to use it since the plotting_scope function will call this function automatically.

https://github.com/modelplot/modelplotr for details on the package

https://modelplot.github.io/ for our blog on the value of the model plots

Examples

```
# load example data (Bank clients with/without a term deposit - see ?bank_td for details)
data("bank_td")
# prepare data for training model for binomial target has_td and train models
train_index =  sample(seq(1, nrow(bank_td)),size = 0.5*nrow(bank_td) ,replace = FALSE)
train = bank_td[train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
test = bank_td[-train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
#train models using caret... (or use mlr or H2o or keras ... see ?prepare_scores_and_ntiles)
# setting caret cross validation, here tuned for speed (not accuracy!)
fitControl <- caret::trainControl(method = "cv",number = 2,classProbs=TRUE)
# random forest using ranger package, here tuned for speed (not accuracy!)
rf = caret::train(has_td ~.,data = train, method = "ranger",trControl = fitControl,
                tuneGrid = expand.grid(.mtry = 2,.splitrule = "gini",.min.node.size=10))
# mnl model using glmnet package
mnl = caret::train(has_td ~.,data = train, method = "glmnet",trControl = fitControl)
# load modelplotr
library(modelplotr)
# transform datasets and model objects to input for modelplotr
scores_and_ntiles <- prepare_scores_and_ntiles(datasets=list("train","test"),
                         dataset_labels = list("train data","test data"),
                         models = list("rf","mnl"),
                         model_labels = list("random forest","multinomial logit"),
                         target_column="has_td",
                         ntiles=100)
plot_input <- plotting_scope(prepared_input = scores_and_ntiles,scope="compare_models")
plot_cumgains(data=plot_input)
plot_cumgains(data=plot_input,custom_line_colors=c("orange","purple"))
plot_cumgains(data=plot_input,highlight_ntile=20)
```

---

plot_cumlift *Cumulative Lift plot*

Description

Generates the cumulative lift plot, often referred to as lift plot or index plot, helps you answer the question: When we apply the model and select the best X ntiles, how many times better is that than using no model at all?

Usage

```
plot_cumlift(
  data = plot_input,
  highlight_ntile = NA,
  highlight_how = "plot_text",
  save_fig = FALSE,
  save_fig_filename = NA,
  custom_line_colors = NA,
  custom_plot_text = NULL
)
```

Arguments

`data`	Dataframe. Dataframe needs to be created with `plotting_scope` or else meet required input format.
`highlight_ntile`	Integer. Specifying the ntile at which the plot is annotated and/or performances are highlighted.
`highlight_how`	String. How to annotate the plot. Possible values: "plot_text","plot", "text". Default is "plot_text", both highlighting the ntile and value on the plot as well as in text below the plot. "plot" only highligths the plot, but does not add text below the plot explaining the plot at chosen ntile. "text" adds text below the plot explaining the plot at chosen ntile but does not highlight the plot.
`save_fig`	Logical. Save plot to file? Default = FALSE. When set to TRUE, saved plots are optimized for 18x12cm.
`save_fig_filename`	String. Filename of saved plot. Default the plot is saved as tempdir()/plotname.png.
`custom_line_colors`	Vector of Strings. Specifying colors for the lines in the plot. When not specified, colors from the RColorBrewer palet "Set1" are used.
`custom_plot_text`	List. List with customized textual elements for plot. Create a list with defaults by using `customize_plot_text` and override default values to customize.

Value

ggplot object. Lift plot.

See Also

`modelplotr` for generic info on the package `moddelplotr`

`vignette('modelplotr')`

`plotting_scope` for details on the function `plotting_scope` that transforms a dataframe created with `prepare_scores_and_ntiles` or `aggregate_over_ntiles` to a dataframe in the required format for all modelplotr plots.

aggregate_over_ntiles for details on the function aggregate_over_ntiles that aggregates the output of prepare_scores_and_ntiles to create a dataframe with aggregated actuals and predictions. In most cases, you do not need to use it since the plotting_scope function will call this function automatically.

https://github.com/modelplot/modelplotr for details on the package

https://modelplot.github.io/ for our blog on the value of the model plots

Examples

```
# load example data (Bank clients with/without a term deposit - see ?bank_td for details)
data("bank_td")
# prepare data for training model for binomial target has_td and train models
train_index =  sample(seq(1, nrow(bank_td)),size = 0.5*nrow(bank_td) ,replace = FALSE)
train = bank_td[train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
test = bank_td[-train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
#train models using caret... (or use mlr or H2o or keras ... see ?prepare_scores_and_ntiles)
# setting caret cross validation, here tuned for speed (not accuracy!)
fitControl <- caret::trainControl(method = "cv",number = 2,classProbs=TRUE)
# random forest using ranger package, here tuned for speed (not accuracy!)
rf = caret::train(has_td ~.,data = train, method = "ranger",trControl = fitControl,
                 tuneGrid = expand.grid(.mtry = 2,.splitrule = "gini",.min.node.size=10))
# mnl model using glmnet package
mnl = caret::train(has_td ~.,data = train, method = "glmnet",trControl = fitControl)
# load modelplotr
library(modelplotr)
# transform datasets and model objects to input for modelplotr
scores_and_ntiles <- prepare_scores_and_ntiles(datasets=list("train","test"),
                         dataset_labels = list("train data","test data"),
                         models = list("rf","mnl"),
                         model_labels = list("random forest","multinomial logit"),
                         target_column="has_td",
                         ntiles=100)
plot_input <- plotting_scope(prepared_input = scores_and_ntiles,scope="compare_datasets")
plot_cumlift(data=plot_input)
plot_cumlift(data=plot_input,custom_line_colors=c("orange","purple"))
plot_cumlift(data=plot_input,highlight_ntile=2)
```

---

plot_cumresponse *Cumulative Respose plot*

Description

Generates the cumulative response plot. It plots the cumulative percentage of target class observations up until that ntile. It helps answering the question: When we apply the model and select up until ntile X, what is the expected percentage of target class observations in the selection?

Usage

```
plot_cumresponse(
  data = plot_input,
  highlight_ntile = NA,
  highlight_how = "plot_text",
  save_fig = FALSE,
  save_fig_filename = NA,
  custom_line_colors = NA,
  custom_plot_text = NULL
)
```

Arguments

`data`	Dataframe. Dataframe needs to be created with `plotting_scope` or else meet required input format.
`highlight_ntile`	Integer. Specifying the ntile at which the plot is annotated and/or performances are highlighted.
`highlight_how`	String. How to annotate the plot. Possible values: "plot_text","plot", "text". Default is "plot_text", both highlighting the ntile and value on the plot as well as in text below the plot. "plot" only highligths the plot, but does not add text below the plot explaining the plot at chosen ntile. "text" adds text below the plot explaining the plot at chosen ntile but does not highlight the plot.
`save_fig`	Logical. Save plot to file? Default = FALSE. When set to TRUE, saved plots are optimized for 18x12cm.
`save_fig_filename`	String. Filename of saved plot. Default the plot is saved as tempdir()/plotname.png.
`custom_line_colors`	Vector of Strings. Specifying colors for the lines in the plot. When not specified, colors from the RColorBrewer palet "Set1" are used.
`custom_plot_text`	List. List with customized textual elements for plot. Create a list with defaults by using `customize_plot_text` and override default values to customize.

Value

ggplot object. Cumulative Response plot.

See Also

`modelplotr` for generic info on the package `moddelplotr`

`vignette('modelplotr')`

`plotting_scope` for details on the function `plotting_scope` that transforms a dataframe created with `prepare_scores_and_ntiles` or `aggregate_over_ntiles` to a dataframe in the required format for all modelplotr plots.

aggregate_over_ntiles for details on the function aggregate_over_ntiles that aggregates the output of prepare_scores_and_ntiles to create a dataframe with aggregated actuals and predictions. In most cases, you do not need to use it since the plotting_scope function will call this function automatically.

https://github.com/modelplot/modelplotr for details on the package

https://modelplot.github.io/ for our blog on the value of the model plots

Examples

```
# load example data (Bank clients with/without a term deposit - see ?bank_td for details)
data("bank_td")
# prepare data for training model for binomial target has_td and train models
train_index =  sample(seq(1, nrow(bank_td)),size = 0.5*nrow(bank_td) ,replace = FALSE)
train = bank_td[train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
test = bank_td[-train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
#train models using caret... (or use mlr or H2o or keras ... see ?prepare_scores_and_ntiles)
# setting caret cross validation, here tuned for speed (not accuracy!)
fitControl <- caret::trainControl(method = "cv",number = 2,classProbs=TRUE)
# random forest using ranger package, here tuned for speed (not accuracy!)
rf = caret::train(has_td ~.,data = train, method = "ranger",trControl = fitControl,
                tuneGrid = expand.grid(.mtry = 2,.splitrule = "gini",.min.node.size=10))
# mnl model using glmnet package
mnl = caret::train(has_td ~.,data = train, method = "glmnet",trControl = fitControl)
# load modelplotr
library(modelplotr)
# transform datasets and model objects to input for modelplotr
scores_and_ntiles <- prepare_scores_and_ntiles(datasets=list("train","test"),
                         dataset_labels = list("train data","test data"),
                         models = list("rf","mnl"),
                         model_labels = list("random forest","multinomial logit"),
                         target_column="has_td",
                         ntiles=20)
plot_input <- plotting_scope(prepared_input = scores_and_ntiles)
plot_cumresponse(data=plot_input)
plot_cumresponse(data=plot_input,custom_line_colors="pink")
plot_cumresponse(data=plot_input,highlight_ntile=5)
```

---

plot_multiplot *Create plot with all four evaluation plots*

Description

Generates a layout containing a number graphical elements, including title, subtitle and the four model evaluation plots: cumulative gains plot, lift plot, response plot and cumulative response plot.

Usage

```
plot_multiplot(
  data = plot_input,
  save_fig = FALSE,
  save_fig_filename = NA,
  custom_line_colors = NA,
  highlight_ntile = NA,
  custom_plot_text = NULL
)
```

Arguments

`data`	Dataframe. Dataframe needs to be created with `plotting_scope` or else meet required input format.
`save_fig`	Logical. Save plot to file? Default = FALSE. When set to TRUE, saved plot is optimized for 36x24cm.
`save_fig_filename`	String. Filename of saved plot. Default the plot is saved as tempdir()/plotname.png.
`custom_line_colors`	Vector of Strings. Specifying colors for the lines in the plot. When not specified, colors from the RColorBrewer palet "Set1" are used.
`highlight_ntile`	Integer. Specifying the ntile at which the plot is annotated and/or performances are highlighted.
`custom_plot_text`	List. List with customized textual elements for plot. Create a list with defaults by using `customize_plot_text` and override default values to customize.

Value

gtable, containing 6 grobs.

See Also

`modelplotr` for generic info on the package `moddelplotr`

`vignette('modelplotr')`

`plotting_scope` for details on the function `plotting_scope` that transforms a dataframe created with `prepare_scores_and_ntiles` or `aggregate_over_ntiles` to a dataframe in the required format for all modelplotr plots.

`aggregate_over_ntiles` for details on the function `aggregate_over_ntiles` that aggregates the output of `prepare_scores_and_ntiles` to create a dataframe with aggregated actuals and predictions. In most cases, you do not need to use it since the `plotting_scope` function will call this function automatically.

`https://github.com/modelplot/modelplotr` for details on the package

`https://modelplot.github.io/` for our blog on the value of the model plots

Examples

```
# load example data (Bank clients with/without a term deposit - see ?bank_td for details)
data("bank_td")
# prepare data for training model for binomial target has_td and train models
train_index =  sample(seq(1, nrow(bank_td)),size = 0.5*nrow(bank_td) ,replace = FALSE)
train = bank_td[train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
test = bank_td[-train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
#train models using caret... (or use mlr or H2o or keras ... see ?prepare_scores_and_ntiles)
# setting caret cross validation, here tuned for speed (not accuracy!)
fitControl <- caret::trainControl(method = "cv",number = 2,classProbs=TRUE)
# random forest using ranger package, here tuned for speed (not accuracy!)
rf = caret::train(has_td ~.,data = train, method = "ranger",trControl = fitControl,
                tuneGrid = expand.grid(.mtry = 2,.splitrule = "gini",.min.node.size=10))
# mnl model using glmnet package
mnl = caret::train(has_td ~.,data = train, method = "glmnet",trControl = fitControl)
# load modelplotr
library(modelplotr)
# transform datasets and model objects to input for modelplotr
scores_and_ntiles <- prepare_scores_and_ntiles(datasets=list("train","test"),
                         dataset_labels = list("train data","test data"),
                         models = list("rf","mnl"),
                         model_labels = list("random forest","multinomial logit"),
                         target_column="has_td",
                         ntiles=10)
plot_input <- plotting_scope(prepared_input = scores_and_ntiles)
plot_multiplot(data=plot_input)
plot_multiplot(data=plot_input,highlight_ntile = 2)
```

`plot_profit` *Profit plot*

Description

Generates the Profit plot. It plots the cumulative profit up until that ntile when the model is used for campaign selection. It can be used to answer the following business question: ***When we apply the model and select up until ntile X, what is the expected profit of the campaign?*** Extra parameters needed for this plot are: fixed_costs, variable_costs_per_unit and profit_per_unit.

Usage

```
plot_profit(
  data = plot_input,
  highlight_ntile = "max_profit",
  highlight_how = "plot_text",
  save_fig = FALSE,
  save_fig_filename = NA,
  custom_line_colors = NA,
  custom_plot_text = NULL,
  fixed_costs,
  variable_costs_per_unit,
  profit_per_unit
)
```

Arguments

`data`	Dataframe. Dataframe needs to be created with `plotting_scope` or else meet required input format.
`highlight_ntile`	Integer or string ("max_roi" or "max_profit"). Specifying the ntile at which the plot is annotated and/or performances are highlighted. Default value is `max_profit`, highlighting the ntile where profit is highest.
`highlight_how`	String. How to annotate the plot. Possible values: "plot_text","plot", "text". Default is "plot_text", both highlighting the ntile and value on the plot as well as in text below the plot. "plot" only highligths the plot, but does not add text below the plot explaining the plot at chosen ntile. "text" adds text below the plot explaining the plot at chosen ntile but does not highlight the plot.
`save_fig`	Logical. Save plot to file? Default = FALSE. When set to TRUE, saved plot is optimized for 36x24cm.
`save_fig_filename`	String. Filename of saved plot. Default the plot is saved as tempdir()/plotname.png.
`custom_line_colors`	Vector of Strings. Specifying colors for the lines in the plot. When not specified, colors from the RColorBrewer palet "Set1" are used.
`custom_plot_text`	List. List with customized textual elements for plot. Create a list with defaults by using `customize_plot_text` and override default values to customize.
`fixed_costs`	Numeric. Specifying the fixed costs related to a selection based on the model. These costs are constant and do not vary with selection size (ntiles).
`variable_costs_per_unit`	Numeric. Specifying the variable costs per selected unit for a selection based on the model. These costs vary with selection size (ntiles).
`profit_per_unit`	Numeric. Specifying the profit per unit in case the selected unit converts / responds positively.

Value

gtable, containing 6 grobs.

See Also

`modelplotr` for generic info on the package `moddelplotr`

`vignette('modelplotr')`

`plotting_scope` for details on the function `plotting_scope` that transforms a dataframe created with `prepare_scores_and_ntiles` or `aggregate_over_ntiles` to a dataframe in the required format for all modelplotr plots.

aggregate_over_ntiles for details on the function aggregate_over_ntiles that aggregates the output of prepare_scores_and_ntiles to create a dataframe with aggregated actuals and predictions. In most cases, you do not need to use it since the plotting_scope function will call this function automatically.

https://github.com/modelplot/modelplotr for details on the package

https://modelplot.github.io/ for our blog on the value of the model plots

Examples

```
# load example data (Bank clients with/without a term deposit - see ?bank_td for details)
data("bank_td")
# prepare data for training model for binomial target has_td and train models
train_index =  sample(seq(1, nrow(bank_td)),size = 0.5*nrow(bank_td) ,replace = FALSE)
train = bank_td[train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
test = bank_td[-train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
#train models using caret... (or use mlr or H2o or keras ... see ?prepare_scores_and_ntiles)
# setting caret cross validation, here tuned for speed (not accuracy!)
fitControl <- caret::trainControl(method = "cv",number = 2,classProbs=TRUE)
# random forest using ranger package, here tuned for speed (not accuracy!)
rf = caret::train(has_td ~.,data = train, method = "ranger",trControl = fitControl,
                tuneGrid = expand.grid(.mtry = 2,.splitrule = "gini",.min.node.size=10))
# mnl model using glmnet package
mnl = caret::train(has_td ~.,data = train, method = "glmnet",trControl = fitControl)
# load modelplotr
library(modelplotr)
# transform datasets and model objects to input for modelplotr
scores_and_ntiles <- prepare_scores_and_ntiles(datasets=list("train","test"),
                         dataset_labels = list("train data","test data"),
                         models = list("rf","mnl"),
                         model_labels = list("random forest","multinomial logit"),
                         target_column="has_td",
                         ntiles=100)
# set scope for analysis (default: no comparison)
plot_input <- plotting_scope(prepared_input = scores_and_ntiles,scope='compare_models')
plot_profit(data=plot_input,fixed_costs=1000,variable_costs_per_unit= 10,profit_per_unit=50)
plot_profit(data=plot_input,fixed_costs=1000,variable_costs_per_unit= 10,profit_per_unit=50,
            highlight_ntile=20)
plot_profit(data=plot_input,fixed_costs=1000,variable_costs_per_unit= 10,profit_per_unit=50,
            highlight_ntile='max_roi')
```

---

plot_response *Response plot*

Description

Generates the response plot. It plots the percentage of target class observations per ntile. It can be used to answer the following business question: When we apply the model and select ntile X, what is the expected percentage of target class observations in that ntile?

Usage

```
plot_response(
  data = plot_input,
  highlight_ntile = NA,
  highlight_how = "plot_text",
  save_fig = FALSE,
  save_fig_filename = NA,
  custom_line_colors = NA,
  custom_plot_text = NULL
)
```

Arguments

`data`	Dataframe. Dataframe needs to be created with `plotting_scope` or else meet required input format.
`highlight_ntile`	Integer. Specifying the ntile at which the plot is annotated and/or performances are highlighted.
`highlight_how`	String. How to annotate the plot. Possible values: "plot_text","plot", "text". Default is "plot_text", both highlighting the ntile and value on the plot as well as in text below the plot. "plot" only highligths the plot, but does not add text below the plot explaining the plot at chosen ntile. "text" adds text below the plot explaining the plot at chosen ntile but does not highlight the plot.
`save_fig`	Logical. Save plot to file? Default = FALSE. When set to TRUE, saved plots are optimized for 18x12cm.
`save_fig_filename`	String. Filename of saved plot. Default the plot is saved as tempdir()/plotname.png.
`custom_line_colors`	Vector of Strings. Specifying colors for the lines in the plot. When not specified, colors from the RColorBrewer palet "Set1" are used.
`custom_plot_text`	List. List with customized textual elements for plot. Create a list with defaults by using `customize_plot_text` and override default values to customize.

Value

ggplot object. Response plot.

See Also

`modelplotr` for generic info on the package `moddelplotr`

`vignette('modelplotr')`

`plotting_scope` for details on the function `plotting_scope` that transforms a dataframe created with `prepare_scores_and_ntiles` or `aggregate_over_ntiles` to a dataframe in the required format for all modelplotr plots.

aggregate_over_ntiles for details on the function aggregate_over_ntiles that aggregates the output of prepare_scores_and_ntiles to create a dataframe with aggregated actuals and predictions. In most cases, you do not need to use it since the plotting_scope function will call this function automatically.

https://github.com/modelplot/modelplotr for details on the package

https://modelplot.github.io/ for our blog on the value of the model plots

Examples

```
# load example data (Bank clients with/without a term deposit - see ?bank_td for details)
data("bank_td")
# prepare data for training model for binomial target has_td and train models
train_index =  sample(seq(1, nrow(bank_td)),size = 0.5*nrow(bank_td) ,replace = FALSE)
train = bank_td[train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
test = bank_td[-train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
#train models using caret... (or use mlr or H2o or keras ... see ?prepare_scores_and_ntiles)
# setting caret cross validation, here tuned for speed (not accuracy!)
fitControl <- caret::trainControl(method = "cv",number = 2,classProbs=TRUE)
# random forest using ranger package, here tuned for speed (not accuracy!)
rf = caret::train(has_td ~.,data = train, method = "ranger",trControl = fitControl,
                tuneGrid = expand.grid(.mtry = 2,.splitrule = "gini",.min.node.size=10))
# mnl model using glmnet package
mnl = caret::train(has_td ~.,data = train, method = "glmnet",trControl = fitControl)
# load modelplotr
library(modelplotr)
# transform datasets and model objects to input for modelplotr
scores_and_ntiles <- prepare_scores_and_ntiles(datasets=list("train","test"),
                         dataset_labels = list("train data","test data"),
                         models = list("rf","mnl"),
                         model_labels = list("random forest","multinomial logit"),
                         target_column="has_td",
                         ntiles=100)
plot_input <- plotting_scope(prepared_input = scores_and_ntiles)
plot_response(data=plot_input)
plot_response(data=plot_input,custom_line_colors=RColorBrewer::brewer.pal(3,"Dark2"))
plot_response(data=plot_input,highlight_ntile=2)
```

---

plot_roi *ROI plot*

Description

Generates the Return on Investment plot. It plots the cumulative revenues as a percentage of investments up until that ntile when the model is used for campaign selection. It can be used to answer the following business question: ***When we apply the model and select up until ntile X, what is the expected return on investment of the campaign?*** Extra parameters needed for this plot are: fixed_costs, variable_costs_per_unit and profit_per_unit.

Usage

```
plot_roi(
  data = plot_input,
  highlight_ntile = "max_roi",
  highlight_how = "plot_text",
  save_fig = FALSE,
  save_fig_filename = NA,
  custom_line_colors = NA,
  custom_plot_text = NULL,
  fixed_costs,
  variable_costs_per_unit,
  profit_per_unit
)
```

Arguments

`data`	Dataframe. Dataframe needs to be created with `plotting_scope` or else meet required input format.
`highlight_ntile`	Integer or string ("max_roi" or "max_profit"). Specifying the ntile at which the plot is annotated and/or performances are highlighted. Default value is `max_roi`, highlighting the ntile where roi is highest.
`highlight_how`	String. How to annotate the plot. Possible values: "plot_text","plot", "text". Default is "plot_text", both highlighting the ntile and value on the plot as well as in text below the plot. "plot" only highligths the plot, but does not add text below the plot explaining the plot at chosen ntile. "text" adds text below the plot explaining the plot at chosen ntile but does not highlight the plot.
`save_fig`	Logical. Save plot to file? Default = FALSE. When set to TRUE, saved plot is optimized for 36x24cm.
`save_fig_filename`	String. Filename of saved plot. Default the plot is saved as tempdir()/plotname.png.
`custom_line_colors`	Vector of Strings. Specifying colors for the lines in the plot. When not specified, colors from the RColorBrewer palet "Set1" are used.
`custom_plot_text`	List. List with customized textual elements for plot. Create a list with defaults by using `customize_plot_text` and override default values to customize.
`fixed_costs`	Numeric. Specifying the fixed costs related to a selection based on the model. These costs are constant and do not vary with selection size (ntiles).
`variable_costs_per_unit`	Numeric. Specifying the variable costs per selected unit for a selection based on the model. These costs vary with selection size (ntiles).
`profit_per_unit`	Numeric. Specifying the profit per unit in case the selected unit converts / responds positively.

Value

gtable, containing 6 grobs. # load example data (Bank clients with/without a term deposit - see ?bank_td for details)

See Also

`modelplotr` for generic info on the package `moddelplotr`

`vignette('modelplotr')`

`plotting_scope` for details on the function `plotting_scope` that transforms a dataframe created with `prepare_scores_and_ntiles` or `aggregate_over_ntiles` to a dataframe in the required format for all modelplotr plots.

`aggregate_over_ntiles` for details on the function `aggregate_over_ntiles` that aggregates the output of `prepare_scores_and_ntiles` to create a dataframe with aggregated actuals and predictions. In most cases, you do not need to use it since the `plotting_scope` function will call this function automatically.

`https://github.com/modelplot/modelplotr` for details on the package

`https://modelplot.github.io/` for our blog on the value of the model plots

Examples

```
# load example data (Bank clients with/without a term deposit - see ?bank_td for details)
data("bank_td")
# prepare data for training model for binomial target has_td and train models
train_index =  sample(seq(1, nrow(bank_td)),size = 0.5*nrow(bank_td) ,replace = FALSE)
train = bank_td[train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
test = bank_td[-train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
#train models using caret... (or use mlr or H2o or keras ... see ?prepare_scores_and_ntiles)
# setting caret cross validation, here tuned for speed (not accuracy!)
fitControl <- caret::trainControl(method = "cv",number = 2,classProbs=TRUE)
# random forest using ranger package, here tuned for speed (not accuracy!)
rf = caret::train(has_td ~.,data = train, method = "ranger",trControl = fitControl,
                tuneGrid = expand.grid(.mtry = 2,.splitrule = "gini",.min.node.size=10))
# mnl model using glmnet package
mnl = caret::train(has_td ~.,data = train, method = "glmnet",trControl = fitControl)
# load modelplotr
library(modelplotr)
# transform datasets and model objects to input for modelplotr
scores_and_ntiles <- prepare_scores_and_ntiles(datasets=list("train","test"),
                         dataset_labels = list("train data","test data"),
                         models = list("rf","mnl"),
                         model_labels = list("random forest","multinomial logit"),
                         target_column="has_td",
                         ntiles=100)
# set scope for analysis (default: no comparison)
plot_input <- plotting_scope(prepared_input = scores_and_ntiles)
plot_roi(data=plot_input,fixed_costs=1000,variable_costs_per_unit= 10,profit_per_unit=50)
plot_roi(data=plot_input,fixed_costs=1000,variable_costs_per_unit= 10,profit_per_unit=50,
         highlight_ntile=20)
plot_roi(data=plot_input,fixed_costs=1000,variable_costs_per_unit= 10,profit_per_unit=50,
         highlight_ntile="max_profit")
```

prepare_scores_and_ntiles *Build a dataframe containing Actuals, Probabilities and Ntiles*

Description

Build dataframe object that contains actuals and predictions on the target variable for each dataset in `datasets` and each model in `models`

Usage

```
prepare_scores_and_ntiles(
  datasets,
  dataset_labels,
  models,
  model_labels,
  target_column,
  ntiles = 10
)
```

Arguments

`datasets`	List of Strings. A list of the names of the dataframe objects to include in model evaluation. All dataframes need to contain a target variable and feature variables.
`dataset_labels`	List of Strings. A list of labels for the datasets, shown in plots. When dataset_labels is not specified, the names from `datasets` are used.
`models`	List of Strings. List of the names of the model objects, containing parameters to apply models to datasets. To use this function, model objects need to be generated by the mlr package or the caret package or the h20 package or the keras package. Modelplotr automatically detects whether the model is built using mlr or caret or h2o or keras.
`model_labels`	List of Strings. Labels for the models, shown in plots. When model_labels is not specified, the names from `moddels` are used.
`target_column`	String. Name of the target variable in datasets. Target can be either binary or multinomial. Continuous targets are not supported.
`ntiles`	Integer. Number of ntiles. The ntile parameter represents the specified number of equally sized buckets the observations in each dataset are grouped into. By default, observations are grouped in 10 equally sized buckets, often referred to as deciles.

Value

Dataframe. A dataframe is built, based on the `datasets` and `models` specified. It contains the dataset name, actuals on the `target_column` , the predicted probabilities for each target class (eg. unique target value) and attribution to ntiles in the dataset for each target class.

When you build scores_and_ntiles yourself

To make plots with modelplotr, is not required to use this function to generate input for function `plotting_scope` You can create your own dataframe containing actuals and predictions and ntiles, See `build_input_yourself` for an example to build the required input for `plotting_scope` or `aggregate_over_ntiles` yourself, within r or even outside of r.

See Also

`modelplotr` for generic info on the package `moddelplotr`

`vignette('modelplotr')`

`plotting_scope` for details on the function `plotting_scope` that transforms a dataframe created with `prepare_scores_and_ntiles` or `aggregate_over_ntiles` to a dataframe in the required format for all modelplotr plots.

`aggregate_over_ntiles` for details on the function `aggregate_over_ntiles` that aggregates the output of `prepare_scores_and_ntiles` to create a dataframe with aggregated actuals and predictions. In most cases, you do not need to use it since the `plotting_scope` function will call this function automatically.

`https://github.com/modelplot/modelplotr` for details on the package

`https://modelplot.github.io/` for our blog on the value of the model plots

Examples

```
## Not run:
# load example data (Bank clients with/without a term deposit - see ?bank_td for details)
data("bank_td")

# prepare data for training model for binomial target has_td and train models
train_index =  sample(seq(1, nrow(bank_td)),size = 0.5*nrow(bank_td) ,replace = FALSE)
train = bank_td[train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
test = bank_td[-train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]

#train models using mlr...
trainTask <- mlr::makeClassifTask(data = train, target = "has_td")
testTask <- mlr::makeClassifTask(data = test, target = "has_td")
mlr::configureMlr() # this line is needed when using mlr without loading it (mlr::)
task = mlr::makeClassifTask(data = train, target = "has_td")
lrn = mlr::makeLearner("classif.randomForest", predict.type = "prob")
rf = mlr::train(lrn, task)
lrn = mlr::makeLearner("classif.multinom", predict.type = "prob")
mnl = mlr::train(lrn, task)
#... or train models using caret...
# setting caret cross validation, here tuned for speed (not accuracy!)
fitControl <- caret::trainControl(method = "cv",number = 2,classProbs=TRUE)
# random forest using ranger package, here tuned for speed (not accuracy!)
rf = caret::train(has_td ~.,data = train, method = "ranger",trControl = fitControl,
                tuneGrid = expand.grid(.mtry = 2,.splitrule = "gini",.min.node.size=10))
# mnl model using glmnet package
mnl = caret::train(has_td ~.,data = train, method = "glmnet",trControl = fitControl)
#... or train models using h2o...
```

```
h2o::h2o.init()
h2o::h2o.no_progress()
h2o_train = h2o::as.h2o(train)
h2o_test = h2o::as.h2o(test)
gbm <- h2o::h2o.gbm(y = "has_td",
                          x = setdiff(colnames(train), "has_td"),
                          training_frame = h2o_train,
                          nfolds = 5)
#... or train models using keras.
x_train <- as.matrix(train[,-1]); y=train[,1]; y_train <- keras::to_categorical(as.numeric(y)-1);
`%>%` <- magrittr::`%>%`
nn <- keras::keras_model_sequential() %>%
keras::layer_dense(units = 16,kernel_initializer = "uniform",activation = 'relu',
                   input_shape = NCOL(x_train))%>%
  keras::layer_dense(units = 16,kernel_initializer = "uniform", activation='relu') %>%
  keras::layer_dense(units = length(levels(train[,1])),activation='softmax')
nn %>% keras::compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=c('accuracy'))
nn %>% keras::fit(x_train,y_train,epochs = 20,batch_size = 1028,verbose=0)

# preparation steps
scores_and_ntiles <- prepare_scores_and_ntiles(datasets=list("train","test"),
                      dataset_labels = list("train data","test data"),
                      models = list("rf","mnl", "gbm","nn"),
                      model_labels = list("random forest","multinomial logit",
                                "gradient boosting machine","artificial neural network"),
                      target_column="has_td")
plot_input <- plotting_scope(prepared_input = scores_and_ntiles)
plot_cumgains(data = plot_input)
plot_cumlift(data = plot_input)
plot_response(data = plot_input)
plot_cumresponse(data = plot_input)
plot_multiplot(data = plot_input)
plot_costsrevs(data=plot_input,fixed_costs=1000,variable_costs_per_unit=10,profit_per_unit=50)
plot_profit(data=plot_input,fixed_costs=1000,variable_costs_per_unit=10,profit_per_unit=50)
plot_roi(data=plot_input,fixed_costs=1000,variable_costs_per_unit=10,profit_per_unit=50)

## End(Not run)
```

---

prepare_scores_and_ntiles_keras

Build a dataframe containing Actuals, Probabilities and Ntiles for keras models

---

Description

Build dataframe object that contains actuals and predictions on the target variable for each input list in `inputlists` and each (sequential/functional API) keras model in `models`

Usage

```
prepare_scores_and_ntiles_keras(
  inputlists,
  inputlist_labels,
  outputlists,
  select_output_index = 1,
  models,
  model_labels,
  targetclass_labels,
  ntiles = 10
)
```

Arguments

`inputlists`	List of Strings. A list of list names, referring to the input list objects to include in model evaluation.
`inputlist_labels`	List of Strings. A list of labels for the inputlists, shown in plots. When inputlist_labels is not specified, the names from `inputlists` are used.
`outputlists`	List of Strings. A list of list names, referring to the output list objects to include in model evaluation.
`select_output_index`	Integer. The index of the output of `outputlists` to evaluate and show in plots. Only relevant for multi-output models, default index value for multi-output models: 1.
`models`	List of Strings. List of the names of the keras model objects, containing parameters to apply models to datasets. To use this function, model objects need to be generated by the keras package. Both models created with `keras_model_sequential()` as well as models created with the keras functional API are supported by modelplotr.
`model_labels`	List of Strings. Labels for the models, shown in plots. When model_labels is not specified, the names from `moddels` are used.
`targetclass_labels`	List of Strings. A list of names to use in plots for the target class values for the selected output. If not specified, the model output column indices are used. Specify the labels in the same order as the model output columns.
`ntiles`	Integer. Number of ntiles. The ntile parameter represents the specified number of equally sized buckets the observations in each dataset are grouped into. By default, observations are grouped in 10 equally sized buckets, often referred to as deciles.

Value

Dataframe. A dataframe is built, based on the `datasets` and `models` specified. It contains the dataset name, actuals on the `target_column` , the predicted probabilities for each target class (eg. unique target value) and attribution to ntiles in the dataset for each target class.

When you build scores_and_ntiles yourself

To make plots with modelplotr, is not required to use this function to generate input for function plotting_scope You can create your own dataframe containing actuals and predictions and ntiles, See build_input_yourself for an example to build the required input for plotting_scope or aggregate_over_ntiles yourself, within r or even outside of r.

See Also

modelplotr for generic info on the package moddelplotr

vignette('modelplotr')

plotting_scope for details on the function plotting_scope that transforms a dataframe created with prepare_scores_and_ntiles or aggregate_over_ntiles to a dataframe in the required format for all modelplotr plots.

aggregate_over_ntiles for details on the function aggregate_over_ntiles that aggregates the output of prepare_scores_and_ntiles to create a dataframe with aggregated actuals and predictions. In most cases, you do not need to use it since the plotting_scope function will call this function automatically.

https://github.com/modelplot/modelplotr for details on the package

https://modelplot.github.io/ for our blog on the value of the model plots

Examples

```
## Not run:
# load example data (Bank clients with/without a term deposit - see ?bank_td for details)
data("bank_td")

# prepare data for training model for binomial target has_td and train models
train_index =  sample(seq(1, nrow(bank_td)),size = 0.5*nrow(bank_td) ,replace = FALSE)
train = bank_td[train_index,]
test = bank_td[-train_index,]

train_seq = bank_td[train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]
test_seq = bank_td[-train_index,c('has_td','duration','campaign','pdays','previous','euribor3m')]


#train keras models using keras_model_sequential() .
x_train <- as.matrix(train[,-c(1:2)]); y_train <- 2-as.numeric(train[,1]);
input_train = list(x_train); output_train = list(y_train)
x_test  <- as.matrix(test[,-c(1:2)]);  y_test <- 2-as.numeric(test[,1]);
input_test = list(x_test); output_test = list(y_test)

`%>%` <- magrittr::`%>%`
nn_seq <- keras::keras_model_sequential() %>%
 keras::layer_dense(units = 16,kernel_initializer = "uniform",activation = 'relu',
                    input_shape = NCOL(x_train))%>%
 keras::layer_dense(units = 16,kernel_initializer = "uniform", activation='relu') %>%
 keras::layer_dense(units = 1,activation='sigmoid')
nn_seq %>% keras::compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=c('accuracy'))
nn_seq %>% keras::fit(input_train,output_train,epochs = 20,batch_size = 1028,verbose=0)
```

```
scores_and_ntiles <- prepare_scores_and_ntiles_keras(inputlists = list("input_train","input_test"),
                          inputlist_labels = list("train data","test data"),
                          models = list("nn_seq"),
                          model_labels = list("keras sequential model"),
                          outputlists = list("output_train","output_test"),
                          select_output_index = 1,
                          targetclass_labels = list("no.term.deposit","term.deposit"),
                          ntiles = 10)

plot_input <- plotting_scope(prepared_input = scores_and_ntiles,scope = "compare_datasets")
plot_cumgains(data = plot_input)
plot_cumlift(data = plot_input)
plot_response(data = plot_input)
plot_cumresponse(data = plot_input)
plot_multiplot(data = plot_input)


#... or train keras models using keras functional api (multi-input / multi-output is supported).
x1_train <- as.matrix(train[,c(3:4)]); y1_train <- as.numeric(train[,1])-1;
x2_train <- as.matrix(train[,c(5:7)]); y2_train <- keras::to_categorical(as.numeric(train[,2])-1,
                                                                   num_classes = 4);
input_train = list(x1_train,x2_train); output_train = list(y1_train,y2_train)
x1_test <- as.matrix(test[,c(3:4)]); y1_test <- as.numeric(test[,1])-1;
x2_test <- as.matrix(test[,c(5:7)]); y2_test <- keras::to_categorical(as.numeric(test[,2])-1,
                                                                   num_classes = 4);
input_test = list(x1_test,x2_test); output_test = list(y1_test,y2_test)

x1_input <- keras::layer_input(shape = NCOL(x1_train))
x2_input <- keras::layer_input(shape = NCOL(x2_train))
concatenated <- keras::layer_concatenate(list(x1_input, x2_input)) %>%
 keras::layer_dense(units = 16,kernel_initializer = "uniform", activation='relu') %>%
 keras::layer_dense(units = 16,kernel_initializer = "uniform", activation='relu')
y1_output <- concatenated %>% keras::layer_dense(1, activation = "sigmoid", name = "has_td")
y2_output <- concatenated %>% keras::layer_dense(4, activation = "softmax", name = "td_type")
nn_api <- keras::keras_model(list(x1_input,x2_input), list(y1_output,y2_output))
nn_api %>% keras::compile(optimizer = "rmsprop",
                         loss = c("binary_crossentropy","categorical_crossentropy"))
nn_api %>% keras::fit(list(x1_train, x2_train),list(y1_train, y2_train),20,batch_size = 1028)

scores_and_ntiles <- prepare_scores_and_ntiles_keras(inputlists = list("input_train","input_test"),
                          inputlist_labels = list("train data","test data"),
                          models = list("nn_api"),
                          model_labels = list("keras api model"),
                          outputlists = list("output_train","output_test"),
                          select_output_index = 2,
                  targetclass_labels = list('no.td','td.type.A','td.type.B','td.type.C'),
                          ntiles = 100)
plot_input <- plotting_scope(prepared_input=scores_and_ntiles,scope="compare_targetclasses")
plot_cumgains(data = plot_input)
plot_cumlift(data = plot_input)
plot_response(data = plot_input)
plot_cumresponse(data = plot_input)
```

```
plot_multiplot(data = plot_input)

## End(Not run)
```

Index