Package 'regressoR'

July 1, 2020

Title Regression Data Analysis System

Type Package

Version 1.1.9

Depends R (>= 3.5)

Imports shiny (>= 1.2.0), shinyAce (>= 0.3.3), shinydashboardPlus (>= 0.6.0), shinyWidgets (>= 0.4.4), shinyjs (>= 1.0), flexdashboard (>= 0.5.1.1), neuralnet (>= 1.44.2), rpart (>= 4.1-13), rattle (>= 5.2.0), xgboost (>= 0.81.0.1), colourpicker (>= 1.0), DT (>= 0.5), randomForest (>= 4.6-14), e1071 (>= 1.7-0.1), kknn (>= 1.3.1), corrplot (>= 0.84), ROCR (>= 1.0-7), glmnet (>= 2.0-16), gbm (>= 2.1.5), pls (>= 2.7-1), zip (>= 2.0.0), ggplot2 (>= 3.1.0), dplyr (>= 0.8.0.1), htmltools (>= 0.3.6)

Suggests forcats, gridExtra, tibble, scales, scatterplot3d, psych, dummies, testthat

Description Perform a supervised data analysis on a database through a 'shiny' graphical interface. It includes methods such as linear regression, penalized regression, k-nearest neighbors, decision trees, ada boosting, extreme gradient boosting, random forest, neural networks, deep learning and support vector machines.

License GPL (>= 2)

Encoding UTF-8

LazyData true

URL http://www.promidat.com

RoxygenNote 7.1.0

NeedsCompilation no

Author Oldemar Rodriguez R. [aut, cre], Andres Navarro D. [ctb, prg], Diego Jimenez A. [ctb, prg]

Maintainer Oldemar Rodriguez R. <oldemar.rodriguez@ucr.ac.cr>

Repository CRAN

Date/Publication 2020-07-01 17:00:02 UTC

R topics documented:

`as_string_c` *as_string_c*

Description

creates a string representative of a vector

Usage

```
as_string_c(vect, quote = TRUE)
```

Arguments

vect	a vector with values
quote	a logical value. If TRUE, the values on the vector will be surrounded by quotes.

Examples

```
as_string_c(c("A", "B", "C"))
as_string_c(c(5, 6, 7))
as_string_c(c(5, 6, 7), quote = FALSE)
as_string_c(iris$Species)
```

---

boosting_importance_plot

boosting_importance_plot

Description

generates the code to make the graph of variable importance.

Usage

```
boosting_importance_plot(
  model.var = "modelo.boosting",
  data = "datos.aprendizaje"
)
```

Arguments

model.var	the name of the variable that stores the resulting model.
data	the name of the learning data.

Examples

```
## Not run:
library(gbm)
library(ggplot2)
library(forcats)
library(dplyr)

x <- boosting_model('iris', 'Petal.Length', "model_boosting")
exe(x)

x <- boosting_importance_plot('model_boosting', 'iris')
```

```
exe(x)

## End(Not run)
```

---

boosting_model *boosting_model*

Description

generates the code to create the boosting model.

Usage

```
boosting_model(
  data = "datos.aprendizaje",
  variable.pred = NULL,
  model.var = "modelo.boosting",
  n.trees = 50,
  distribution = "gaussian",
  shrinkage = 0.1
)
```

Arguments

data	the name of the learning data.
variable.pred	the name of the variable to be predicted.
model.var	the name of the variable that stores the resulting model.
n.trees	the n.trees parameter of the model.
distribution	the distribution parameter of the model.
shrinkage	the shrinkage parameter of the model.

See Also

gbm

Examples

```
library(gbm)
library(dplyr)

x <- boosting_model('iris', 'Petal.Length')
exe(x)
print(modelo.boosting)
```

boosting_prediction *boosting_prediction*

Description

generates the code to create the prediction of the boosting model.

Usage

```
boosting_prediction(
  data = "datos.prueba",
  variable.pred = NULL,
  model.var = "modelo.boosting",
  pred.var = "prediccion.boosting",
  n.trees = 50
)
```

Arguments

data	the name of the test data.
variable.pred	the name of the variable to be predicted.
model.var	the name of the variable that stores the resulting model.
pred.var	the name of the variable that stores the resulting prediction.
n.trees	the n.trees parameter of the model.

See Also

gbm

Examples

```
library(gbm)
library(dplyr)
x <- boosting_model('iris', 'Petal.Length', "model_boosting")
exe(x)
print(model_boosting)

x <- boosting_prediction('iris', 'Petal.Length', 'model_boosting', 'my_prediction')
exe(x)
print(my_prediction)
```

calibrate_boosting *calibrate_boosting*

Description

helps to get the maximum of n.minobsinnode and bag.fraction values with which no error is generated in the model.

Usage

```
calibrate_boosting(data)
```

Arguments

data	the name of the learning data.

See Also

gbm

Examples

```
calibrate_boosting(iris)
```

categorical_distribution *categorical_distribution*

Description

makes the graph of the categorical distribution.

Usage

```
categorical_distribution(var)
```

Arguments

var	a vector with the data for the categorical distribution chart.

Examples

```
categorical_distribution(iris$Species)
```

categorical_summary *categorical_summary*

Description

generates the fields for individual categorical analysis.

Usage

```
categorical_summary(data, variable)
```

Arguments

data	a data.frame with the data for analysis.
variable	the name of the variable for analysis.

Examples

```
if(interactive()) {
  library(shiny)
  library(DT)
  shinyApp(ui = fluidPage(fluidRow(uiOutput("resumen"))),
           server = function(input, output) {
                       output$resumen = renderUI(categorical_summary(iris, "Species"))
           })
}
```

clean_report *clean_report*

Description

clean the full report.

Usage

```
clean_report()
```

Examples

```
new_report(iris, 'iris')
get_report()
clean_report()
get_report()
```

code_deactivate *code_deactivate*

Description

creates the code that deactivates the selected variables of the data.

Usage

```
code_deactivate(variables, d = "datos")
```

Arguments

variables	the name of the variables to be deactivated.
d	the name of the current data.

Examples

```
iris2 <- iris
x <- code_deactivate('Species', 'iris2')
exe(x)
head(iris2)
```

code_load *code_load*

Description

generates data reading code.

Usage

```
code_load(
  row.names = TRUE,
  path = NULL,
  sep = ";",
  sep.dec = ",",
  header = TRUE,
  d.o = "datos.originales",
  d = "datos"
)
```

Arguments

row.names	a logical value indicating whether the data has row names.
path	the path of the file.
sep	the column separator in the file.
sep.dec	the decimal separator in the file.
header	a logical value indicating whether the file has a header.
d.o	the name of the original data.
d	the name of the current data.

Examples

```
code_load(TRUE, "MY/PATH/FILE.csv")
```

---

code_NA *code_NA*

Description

creates the code that imputes the NAs data or removes them.

Usage

```
code_NA(deleteNA = TRUE, d.o = "datos.originales")
```

Arguments

deleteNA	a logical value indicating whether the NAs have to be eliminated or whether they have to be imputed. If TRUE then the NAs are eliminated, otherwise the data is imputed.
d.o	the name of the original data.

Examples

```
iris2 <- iris
x <- code_NA(TRUE, 'iris2')
exe(x)
x <- code_NA(FALSE, 'iris2')
exe(x)
```

code_summary *code_summary*

Description

creates the code for the basic summary of variables.

Usage

```
code_summary(data = "datos")
```

Arguments

data	the name of the current data.

Examples

```
x <- code_summary('iris')
exe(x)
```

code_transf *code_transf*

Description

generate code to transform data.

Usage

```
code_transf(variable, new.type, d.o = "datos.originales", d = "datos")
```

Arguments

variable	the name of the variable to be converted.
new.type	the new type of the variable. Can be categorical, numerical or disjunctive. ('categorico', 'numerico', 'disyuntivo')
d.o	the name of the original data.
d	the name of the current data.

Examples

```
iris2 <- iris
x <-code_transf('Species', 'disyuntivo', 'iris', 'iris2')
exe(x)
head(iris2)
```

coef_lambda *coef_lambda*

Description

generates the code to print the penalized regression coefficients.

Usage

```
coef_lambda(
  data = "datos.aprendizaje",
  variable.pred = NULL,
  model.var = "modelo.rlr",
  lambda = NULL,
  cv.var = "cv.glm"
)
```

Arguments

data	the name of the learning data.
variable.pred	the name of the variable to be predicted.
model.var	the name of the variable that stores the resulting model.
lambda	a numerical value in case you don't want to use the optimal lambda.
cv.var	the variable that stores the optimal lambda.

Examples

```
library(glmnet)
x <- rlr_model('iris', 'Petal.Length')
exe(x)

x <- coef_lambda('iris','Petal.Length', 'modelo.rlr')
exe(x)
```

colnames_empty *colnames_empty*

Description

gets names of the columns or an empty string if the data is NULL.

Usage

```
colnames_empty(data)
```

Arguments

data	a data.frame with the data.

Examples

```
colnames_empty(iris)
colnames_empty(NULL)
```

---

combine_names *combine_names*

Description

combine two string vector by grouping according to the first vector.

Usage

```
combine_names(x, y, sep = ".")
```

Arguments

x	a vector to combine with y. The combination is grouped by this parameter.
y	a vector to combine with x.
sep	a string with the separator characters.

Value

a vector with the combination of x and y.

Examples

```
x = c("A", "B", "C")
y = c("1", "2", "3")
combine_names(x, y)
```

comparative_table *comparative_table*

Description

creates the comparison table.

Usage

```
comparative_table(sel, indices)
```

Arguments

sel	the selection of the models to be shown.
indices	the values to be shown.

Examples

```
models <- list('knnl-mode1' = list(0.11,0.22,0.33,0.44),
               'dtl-mode2'  = list(0.12,0.23,0.34,0.45),
               'rfl-mode1'  = list(0.51,0.42,0.13,0.24))
sel <- c("K Vecinos Más Cercanos-mode1", "Bosques Aleatorios-mode1")
comparative_table(sel, models)
```

correlations_plot *correlations_plot*

Description

generates the code of the correlation chart.

Usage

```
correlations_plot(method = "circle", type = "lower")
```

Arguments

method	the visualization method of correlation matrix to be used.
type	display full matrix, lower triangular or upper triangular matrix.

See Also

corrplot

Examples

```
x <- cor_model('iris')
exe(x)
print(correlacion)

x <- correlations_plot()
exe(x)
```

`cor_model` *cor_model*

Description

generates the code to calculate the correlation matrix.

Usage

```
cor_model(data = "datos")
```

Arguments

`data`	the name of the current data.

Examples

```
x <- cor_model('iris')
exe(x)
correlacion
```

`default_calc_normal` *default_calc_normal*

Description

generates the code that creates the asymmetry table.

Usage

```
default_calc_normal(
  data = "datos",
  label.yes = "Positiva",
  label.no = "Negativa",
  label.without = "Sin Asimetría"
)
```

Arguments

data	the name of the current data.
label.yes	the label for when the asymmetry is positive.
label.no	the label for when the asymmetry is negative.
label.without	the label for when there is no asymmetry.

Examples

```
x <- default_calc_normal('iris')
exe(x)
```

---

default_disp *default_disp*

Description

default_disp

Usage

```
default_disp(data = "datos", vars = NULL, color = "#FF0000AA")
```

Arguments

data	the name of the current data.
vars	a vector of length 2 or 3 with the names of the variables for the graph.
color	the color of the dots on the chart.

Examples

```
library(scatterplot3d)

x <- default_disp('iris', c('Sepal.Length', 'Sepal.Width'))
exe(x)

x <- default_disp('iris', c('Sepal.Length', 'Sepal.Width', 'Petal.Length'))
exe(x)
```

def_code_cat *def_code_cat*

Description

def_code_cat

Usage

```
def_code_cat(data = "datos", variable)
```

Arguments

data	the name of the current data.
variable	the name of the variable for the categorical distribution chart.

Examples

```
x <- def_code_cat('iris', 'Species')
exe(x)
```

def_code_num *def_code_num*

Description

def_code_num

Usage

```
def_code_num(data = "datos", variable, color = "red")
```

Arguments

data	the name of the current data.
variable	the name of the variable for the numerical distribution chart.
color	the color of the chart.

Examples

```
x <- def_code_num('iris', 'Petal.Length')
exe(x)
```

disjunctive_data *disjunctive_data*

Description

convert the columns selected to disjunctive.

Usage

```
disjunctive_data(data, vars)
```

Arguments

data	the dataset to be converted.
vars	a vector with the name of columns.

Examples

```
disjunctive_data(iris, "Species")
```

disp_models *disp_models*

Description

this function generates the call code of the scatter function.

Usage

```
disp_models(prediction, model_name, var_pred, data = "datos.prueba")
```

Arguments

prediction	the name of the prediction object.
model_name	the name of the model.
var_pred	the name of the variable to be predicted.
data	the name of the current data.

Examples

```
disp_models("prediction.knn", "KNN", "Species")
```

dt_model *dt_model*

Description

generates the code to create the decision trees model.

Usage

```
dt_model(
  data = "datos.aprendizaje",
  variable.pred = NULL,
  model.var = "modelo.dt",
  minsplit = 20,
  maxdepth = 15
)
```

Arguments

data	the name of the learning data.
variable.pred	the name of the variable to be predicted.
model.var	the name of the variable that stores the resulting model.
minsplit	the minsplit parameter of the model.
maxdepth	the maxdepth parameter of the model.

See Also

rpart

Examples

```
library(rpart)

x <- dt_model('iris', 'Petal.Length')
exe(x)
print(modelo.dt)
```

dt_plot *dt_plot*

Description

makes the graph of the tree.

Usage

```
dt_plot(model.var = "modelo.dt")
```

Arguments

model.var	the name of the variable that stores the resulting prediction.

Examples

```
## Not run:
library(rpart)

x <- dt_model('iris', 'Petal.Length', model.var = 'model_dt')
exe(x)
print(model_dt)

x <- dt_plot('model_dt')
exe(x)

## End(Not run)
```

dt_prediction *dt_prediction*

Description

generates the code to create the prediction of the decision trees model.

Usage

```
dt_prediction(
  data = "datos.prueba",
  model.var = "modelo.dt",
  pred.var = "prediccion.dt"
)
```

Arguments

data	the name of the test data.
model.var	the name of the variable that stores the resulting model.
pred.var	the name of the variable that stores the resulting prediction.

Examples

```
library(rpart)

x <- dt_model('iris', 'Petal.Length', model.var = 'model_dt')
exe(x)
print(model_dt)

x <- dt_prediction('iris', 'model_dt', 'my_prediction')
exe(x)
print(my_prediction)
```

---

error_plot *error_plot*

Description

makes a warning graphic

Usage

```
error_plot(msg)
```

Arguments

msg	the message to be displayed in the graph

Examples

```
error_plot("My Warning")
```

error_variables *error_variables*

Description

draws an error of missing data.

Usage

```
error_variables(num = T)
```

Arguments

num	if TRUE shows a message of missing numerical variables, if FALSE shows a message of missing categorical variables.

Examples

```
error_variables(TRUE)
error_variables(FALSE)
```

exe *exe*

Description

concat and execute a text in R.

Usage

```
exe(..., envir = options_regressor()$exe.envir)
```

Arguments

...	one or more texts to be concatenated and executed.
envir	the environment in which expr is to be evaluated.

Value

the result of the execute.

Examples

```
exe("5+5")
exe("5","+","5")
exe("plot(iris$Species)")
```

extract_code *extract_code*

Description

gets the code of a function in text form.

Usage

```
extract_code(funcion, envir = parent.frame())
```

Arguments

funcion	the name of the function to be extracted.
envir	the environment in which expr is to be evaluated.

Examples

```
extract_code("cat")
extract_code("plot")

parse(text = extract_code("plot"))
```

fisher_calc *fisher_calc*

Description

calculate the fisher skewness.

Usage

```
fisher_calc(x, na.rm = FALSE)
```

Arguments

x	a vector with the data to make the calculation.
na.rm	a logical value indicating whether the NAs have to be eliminated.

Examples

```
fisher_calc(iris$Petal.Length)
```

general_indices *general_indices*

Description

calculates indices to measure accuracy of a model.

Usage

```
general_indices(real, prediccion)
```

Arguments

real	the real values in traning-testing.
prediccion	the prediction values in traning-testing.

Value

a list with the Correlation, Relative Error, Mean Absolute Error and Root Mean Square Error.

Examples

```
real <- rnorm(45)
prediction <- rnorm(45)
model <- "KNN"
general_indices(real, prediction)
```

get_env_report *get_env_report*

Description

gets the environment where the list is found with the report.

Usage

```
get_env_report()
```

Examples

```
e <- get_env_report()
e$codigo.reporte
```

get_report *get_report*

Description

gets the list of report values.

Usage

```
get_report()
```

Examples

```
get_report()
```

gg_color_hue *gg_color_hue*

Description

create colors.

Usage

```
gg_color_hue(n)
```

Arguments

n	an integer specifying the number of colors to create.

Value

color-coded vector

Examples

```
col <- gg_color_hue(3)
plot(iris$Species, col = col)
```

importance_plot_rf *importance_plot_rf*

Description

graphs the importance of variables for the random forest model.

Usage

```
importance_plot_rf(model.rf, title.1, title.2)
```

Arguments

model.rf	a random forest model.
title.1	the title of the first chart.
title.2	the title of the second chart.

See Also

randomForest

Examples

```
library(randomForest)
x <- rf_model('iris', 'Petal.Length')
exe(x)
importance_plot_rf(modelo.rf, translate('impVarA'), translate('impVarRSS'))
```

init_regressor *This function will start regressoR*

Description

An interactive 'Shiny' application for data regression.

Usage

```
init_regressor()
```

Details

Start regressoR

This starts the regressoR application on the user's local computer.

Value

Nothing

Examples

```
if(interactive()){
  init_regressor()
}
```

`insert_report` *insert_report*

Description

inserts an element in the report in the current section.

Usage

```
insert_report(
  id,
  title = NA,
  ...,
  interpretation = TRUE,
  is.chunk = TRUE,
  add = FALSE
)
```

Arguments

`id`	a string with the key of what is inserted in the report.
`title`	the title of the content, if there is no title is NA.
`...`	the content to be inserted.
`interpretation`	a logical value indicating whether a label has to be inserted for interpretation.
`is.chunk`	a logical value indicating whether the content has to be enclosed in a chunk.
`add`	a logical value indicating if the content has to be added to what is a before.

Examples

```
new_report(iris, "iris")
insert_report("1_part", 'Title 1', 'head(iris)\n', 'summary(iris)')
get_report()
clean_report()
```

kkn_model *kkn_model*

Description

generates the code to create the k nearest neighbors model.

Usage

```
kkn_model(
  data = "datos.aprendizaje",
  variable.pred = NULL,
  scale = TRUE,
  kmax = 7,
  kernel = "optimal",
  model.var = "modelo.knn",
  distance = 2
)
```

Arguments

data	the name of the learning data.
variable.pred	the name of the variable to be predicted.
scale	the scale parameter of the model.
kmax	the kmax parameter of the model.
kernel	the kernel parameter of the model.
model.var	the name of the variable that stores the resulting model.
distance	the distance parameter of the model.

See Also

train.kknn

Examples

```
library(kknn)
x <- kkn_model('iris', 'Petal.Length')
exe(x)
print(modelo.knn)
```

kkn_prediction *kkn_prediction*

Description

generates the code to create the prediction of the k nearest neighbors model.

Usage

```
kkn_prediction(
  data = "datos.prueba",
  variable.pred = NULL,
  model.var = "modelo.knn",
  pred.var = "prediccion.knn"
)
```

Arguments

data	the name of the test data.
variable.pred	the name of the variable to be predicted.
model.var	the name of the variable that stores the resulting model.
pred.var	the name of the variable that stores the resulting prediction.

Examples

```
library(kknn)
library(dplyr)

x <- kkn_model('iris', 'Petal.Length', model.var = 'model_knn')
exe(x)
print(model_knn)

x <- kkn_prediction('iris', 'Petal.Length', 'model_knn', 'my_prediction')
exe(x)
print(my_prediction)
```

models_mode *models_mode*

Description

transforms the names of a list from key-mode form to value-mode form.

Usage

```
models_mode(list.names = list())
```

Arguments

list.names	a list whose names function as keys for translate. The names have to have the key-mode form.

Examples

```
x <- list('knnl-mode1' = 1, 'knnl-mode2' = 2, 'knnl-mode2' = 5)
models_mode(x)
```

---

new_col *new_col*

Description

creates a new column.

Usage

```
new_col(data, name = "new_", values = NA)
```

Arguments

data	the data.frame to join with the new column.
name	the name of the new column.
values	the values of the new column.

Examples

```
new_col(iris)
new_col(iris, "var1", c(1,2,3))
```

new_report *new_report*

Description

creates a new report section within the list. All new reports section store data and data names as headers.

Usage

```
new_report(data, name = "")
```

Arguments

data	the data that is stored in the report list
name	the name of the stored data

Examples

```
new_report(iris, "iris")
get_report()
clean_report()
```

new_section_report *new_section_report*

Description

creates a new section in the report, this way you can overwrite keys and delete an element only affects the current section.

Usage

```
new_section_report()
```

Examples

```
new_report(iris, 'iris')
insert_report('1_part', 'Title 1', 'head(iris)\n', 'summary(iris)')
get_report()

remove_report_elem('1_part')
get_report()

new_section_report()
insert_report('1_part', 'Title 1', 'head(iris)\n', 'summary(iris)')
```

```
get_report()

new_section_report()
insert_report('1_part', 'Title 1', 'head(iris)\n', 'summary(iris)')
get_report()

remove_report_elem('1_part')
get_report()

clean_report()
```

---

nn_model *nn_model*

Description

generates the code to create the neural network model.

Usage

```
nn_model(
  data = "datos.aprendizaje",
  variable.pred = NULL,
  model.var = "modelo.nn",
  mean.var = "mean.nn",
  sd.var = "sd.nn",
  threshold = 0.01,
  stepmax = 1000,
  cant.hidden = 2,
  ...
)
```

Arguments

data	the name of the learning data.
variable.pred	the name of the variable to be predicted.
model.var	the name of the variable that stores the resulting model.
mean.var	the name of the variable that stores the mean of the columns.
sd.var	the name of the variable that stores the standard deviation of the columns.
threshold	the threshold parameter of the model.
stepmax	the stepmax parameter of the model.
cant.hidden	the quantity of hidden layers that are going to be used.
...	a vector with the number of nodes in each hidden layer.

See Also

neuralnet

Examples

```
## Not run:
library(neuralnet)
library(dummies)

x <- nn_model('iris', 'Petal.Length','modelo.nn', 'mean.nn', 'sd.nn', 0.05, 2000, 3, 30, 50, 80)
exe(x)

print(modelo.nn)
print(mean.nn)
print(sd.nn)

## End(Not run)
```

---

nn_plot *nn_plot*

Description

generates the code to create the graph of the neural network.

Usage

```
nn_plot(model.var = "modelo.nn")
```

Arguments

model.var	the name of the variable that stores the resulting model.

Examples

```
## Not run:
library(neuralnet)
library(dummies)
library(dplyr)

x <- nn_model('iris', 'Petal.Length','modelo.nn', 'mean.nn', 'sd.nn', 0.05, 2000, 3, 10, 10, 10)
exe(x)

x <- nn_plot('modelo.nn')
exe(x)

## End(Not run)
```

nn_prediction *nn_prediction*

Description

generates the code to create the prediction of the neural network model.

Usage

```
nn_prediction(
  data = "datos.prueba",
  variable.pred = NULL,
  model.var = "modelo.nn",
  pred.var = "prediccion.nn",
  mean.var = "mean.nn",
  sd.var = "sd.nn"
)
```

Arguments

data	the name of the test data.
variable.pred	the name of the variable to be predicted.
model.var	the name of the variable that stores the resulting model.
pred.var	the name of the variable that stores the resulting prediction.
mean.var	the name of the variable that stores the mean of the columns.
sd.var	the name of the variable that stores the standard deviation of the columns.

See Also

compute

Examples

```
## Not run:
library(neuralnet)
library(dummies)
library(dplyr)

x <- nn_model('iris', 'Petal.Length','modelo.nn', 'mean.nn', 'sd.nn', 0.05, 2000, 3, 30, 50, 30)
exe(x)

x <- nn_prediction('iris', 'Petal.Length')
exe(x)
print(prediccion.nn)

## End(Not run)
```

normal_default *normal_default*

Description

generates the code of the normality test.

Usage

```
normal_default(
  data = "datos",
  vars = NULL,
  color = "#00FF22AA",
  labelcurva = "Curva Normal"
)
```

Arguments

data	the name of the current data.
vars	the variable for analysis. It has to be numeric.
color	the color of the histogram.
labelcurva	label for the curve.

Examples

```
x <- normal_default('iris', 'Sepal.Length')
exe(x)
```

numerical_distribution *numerical_distribution*

Description

makes the graph of the numerical distribution.

Usage

```
numerical_distribution(var, var.name, color)
```

Arguments

var	a vector with the data for the numerical distribution chart.
var.name	the name of the variable.
color	the color of the chart.

Examples

```
numerical_distribution(iris[,'Sepal.Length'], 'Sepal.Length', 'red')
```

`numerical_summary` *numerical_summary*

Description

generates the fields for individual numerical analysis.

Usage

```
numerical_summary(data, variable)
```

Arguments

`data`	a data.frame with the data for analysis.
`variable`	the name of the variable for analysis.

Examples

```
if(interactive()) {
  library(shiny)
  library(DT)
  shinyApp(ui = fluidPage(fluidRow(uiOutput("resumen"))),
           server = function(input, output) {
                      output$resumen = renderUI(numerical_summary(iris, "Sepal.Width"))
           })
}
```

`options_regressor` *options_regressor*

Description

options_regressor

Usage

```
options_regressor(...)
```

Arguments

`...`	any options can be defined, using name = value or a character string holding an option name.

Examples

```
options_regressor("language")
options_regressor(language = "en")
options_regressor("language")
```

`pairs_power` *pairs_power*

Description

pairs_power

Usage

```
pairs_power(data = "datos")
```

Arguments

`data`	the name of the current data.

See Also

`pairs.panels`

Examples

```
## Not run:
library(psych)
x <- pairs_power('iris')
exe(x)

## End(Not run)
```

`partition_code` *partition_code*

Description

creates the partition code for testing and learning data.

Usage

```
partition_code(
  data = "datos",
  p = 50,
  variable = NULL,
  semilla = 5,
  perm.semilla = FALSE
)
```

Arguments

data	the name of the current data.
p	the percentage of data for the learning data.
variable	the name of the variable to be predicted.
semilla	a number with the random seed.
perm.semilla	a logical value indicating whether the random seed should be used.

Examples

```
x <- partition_code('iris', 75, 'Species', 555, TRUE)
exe(x)
head(datos.aprendizaje)
head(datos.prueba)
```

---

plot_coef_lambda *plot_coef_lambda*

Description

generates the code to plot the penalized regression coefficients.

Usage

```
plot_coef_lambda(model.var = "modelo.rlr", lambda = NULL, cv.var = "cv.glm")
```

Arguments

model.var	the name of the variable that stores the resulting model.
lambda	a numerical value in case you don't want to use the optimal lambda.
cv.var	the variable that stores the optimal lambda.

Examples

```
library(glmnet)
x <- rlr_model('iris', 'Petal.Length')
exe(x)

x <- plot_coef_lambda('modelo.rlr')
exe(x)
```

`plot_pred_rd` *plot_pred_rd*

Description

graph of variance explained in the predictors according to components used.

Usage

```
plot_pred_rd(model, n.comp = "n.comp.rd")
```

Arguments

`model`	a dimension reduction model.
`n.comp`	the name of the variable that stores the optimum number of components.

Examples

```
library(pls)

x <- rd_model('iris', 'Petal.Length')
exe(x)

plot_pred_rd(modelo.rd)
```

`plot_real_prediction` *plot_real_prediction*

Description

scatter plot between the actual value of the variable to be predicted and the prediction of the model.

Usage

```
plot_real_prediction(real, prediction, model = "")
```

Arguments

real	the real values in traning-testing.
prediction	the prediction values in traning-testing.
model	the name of the model of the scatter plot.

Examples

```
real <- rnorm(45)
prediction <- rnorm(45)
model <- "KNN"
plot_real_prediction(real, prediction, model)
```

---

plot_RMSE *plot_RMSE*

Description

graph the root mean square error of cross validation according to components used.

Usage

```
plot_RMSE(model, n.comp = "n.comp.rd")
```

Arguments

model	a dimension reduction model.
n.comp	the name of the variable that stores the optimum number of components.

Examples

```
library(pls)

x <- rd_model('iris', 'Petal.Length')
exe(x)

plot_RMSE(modelo.rd)
```

plot_var_pred_rd *plot_var_pred_rd*

Description

graph of the variance explained in the variable to predict according to the components used.

Usage

```
plot_var_pred_rd(model, n.comp = "n.comp.rd")
```

Arguments

model	a dimension reduction model.
n.comp	the name of the variable that stores the optimum number of components.

Examples

```
library(pls)

x <- rd_model('iris', 'Petal.Length')
exe(x)

plot_var_pred_rd(modelo.rd)
```

rd_model *rd_model*

Description

generates the code to create the dimension reduction model.

Usage

```
rd_model(
  data = "datos.aprendizaje",
  variable.pred = NULL,
  model.var = "modelo.rd",
  n.comp = "n.comp.rd",
  mode = options_regressor("rd.mode"),
  scale = TRUE
)
```

Arguments

data	the name of the learning data.
variable.pred	the name of the variable to be predicted.
model.var	the name of the variable that stores the resulting model.
n.comp	the name of the variable that stores the optimum number of components.
mode	the method of dimension reduction is defined as mode=1 is the MCP, and mode=0 the ACP.
scale	the scale parameter of the model.

Examples

```
library(pls)

x <- rd_model('iris', 'Petal.Length')
exe(x)
print(modelo.rd)
```

---

rd_prediction *rd_prediction*

Description

generates the code to create the prediction of the dimension reduction model.

Usage

```
rd_prediction(
  data = "datos.prueba",
  model.var = "modelo.svm",
  pred.var = "prediccion.rd",
  n.comp = "n.comp.rd",
  ncomp = NULL
)
```

Arguments

data	the name of the test data.
model.var	the name of the variable that stores the resulting model.
pred.var	the name of the variable that stores the resulting prediction.
n.comp	the name of the variable that stores the optimum number of components.
ncomp	a numerical value in case you don't want to use the optimum number of components.

Examples

```
library(pls)

x <- rd_model('iris', 'Petal.Length')
exe(x)
print(modelo.rd)

x <- rd_prediction('iris', 'modelo.rd', 'my_prediction')
exe(x)
print(my_prediction)
```

`rd_type` *rd_type*

Description

returns the name of the method of dimension reduction.

Usage

```
rd_type(mode.rd = options_regressor("rd.mode"))
```

Arguments

`mode.rd`	the method of dimension reduction is defined as mode=1 is the MCP, and mode=0 the ACP.

Examples

```
rd_type(1)
rd_type(0)
```

`remove_report_elem` *remove_report_elem*

Description

removes an element from the report according to its key in the current section.

Usage

```
remove_report_elem(id)
```

Arguments

id	a string with the key of what is removed in the report.

Examples

```
new_report(iris, 'iris')
insert_report('1_part', 'Title 1', 'head(iris)\n', 'summary(iris)')
get_report()
remove_report_elem('1_part')
get_report()
clean_report()
```

---

render_index_table *render_index_table*

Description

creates a reactive table for indices panels.

Usage

```
render_index_table(table)
```

Arguments

table	the data.frame to be converted

Examples

```
if(interactive()) {
   library(shiny)
   shinyApp(
      ui = fluidPage(fluidRow(column(12, tableOutput('tbl')))),
      server = function(input, output) {
         output$tbl = render_index_table(iris)
      }
   )
}
```

render_table_data *render_table_data*

Description

create a table for the shiny application and render it.

Usage

```
render_table_data(
  data,
  editable = TRUE,
  dom = "frtip",
  pageLength = 10,
  scrollY = "27vh",
  server = T
)
```

Arguments

data	a data.frame to create a the table.
editable	whether to make an editable table. The default value is TRUE.
dom	define the table control elements to appear on the page and in what order.
pageLength	the number of rows to show. The default value is 10.
scrollY	the heigth of the table.
server	whether to use server-side processing. If TRUE, then the data is kept on the server and the browser requests a page at a time; if FALSE, then the entire data frame is sent to the browser at once.

Value

a shiny.render.function

Examples

```
if(interactive()) {
   library(shiny)
   library(DT)
   shinyApp(
      ui = fluidPage(fluidRow(column(12, DTOutput('tbl')))),
      server = function(input, output) {
         output$tbl = render_table_data(iris)
      }
   )
}
```

rf_model *rf_model*

Description

generates the code to create the random forest model.

Usage

```
rf_model(
  data = "datos.aprendizaje",
  variable.pred = NULL,
  model.var = "modelo.rf",
  ntree = 500,
  mtry = 1
)
```

Arguments

data	the name of the learning data.
variable.pred	the name of the variable to be predicted.
model.var	the name of the variable that stores the resulting model.
ntree	the ntree parameter of the model.
mtry	the mtry parameter of the model.

See Also

randomForest

Examples

```
library(randomForest)
x <- rf_model('iris', 'Petal.Length')
exe(x)
print(modelo.rf)
```

rf_prediction *rf_prediction*

Description

generates the code to create the prediction of the random forest model.

Usage

```
rf_prediction(
  data = "datos.prueba",
  variable.pred = NULL,
  model.var = "modelo.rf",
  pred.var = "prediccion.rf"
)
```

Arguments

data	the name of the test data.
variable.pred	the name of the variable to be predicted.
model.var	the name of the variable that stores the resulting model.
pred.var	the name of the variable that stores the resulting prediction.

Examples

```
library(randomForest)
library(dplyr)

x <- rf_model('iris', 'Petal.Length', model.var = 'model_rf')
exe(x)
print(model_rf)

x <- rf_prediction('iris', 'Petal.Length', 'model_rf', 'my_prediction')
exe(x)
print(my_prediction)
```

rlr_model *rlr_model*

Description

generates the code to create the penalized regression model.

Usage

```
rlr_model(
  data = "datos.aprendizaje",
  variable.pred = NULL,
  model.var = "modelo.rlr",
  cv.var = "cv.glm",
  alpha = 0,
  standardize = TRUE
)
```

Arguments

data	the name of the learning data.
variable.pred	the name of the variable to be predicted.
model.var	the name of the variable that stores the resulting model.
cv.var	the variable that stores the optimal lambda.
alpha	the alpha parameter of the model.
standardize	the standardize parameter of the model.

See Also

glmnet, cv.glmnet

Examples

```
library(glmnet)
x <- rlr_model('iris', 'Petal.Length')
exe(x)
print(modelo.rlr)
```

---

rlr_prediction *rlr_prediction*

Description

generates the code to create the prediction of the penalized regression model.

Usage

```
rlr_prediction(
  data.a = "datos.aprendizaje",
  data.p = "datos.prueba",
  variable.pred = NULL,
  model.var = "modelo.rlr",
  pred.var = "prediccion.rlr",
```

```
  lambda = NULL,
  cv.var = "cv.glm"
)
```

Arguments

data.a	the name of the learning data.
data.p	the name of the test data.
variable.pred	the name of the variable to be predicted.
model.var	the name of the variable that stores the resulting model.
pred.var	the name of the variable that stores the resulting prediction.
lambda	a numerical value in case you don't want to use the optimal lambda.
cv.var	the variable that stores the optimal lambda.

Examples

```
library(glmnet)
x <- rlr_model('iris', 'Petal.Length')
exe(x)
print(modelo.rlr)

x <- rlr_prediction('iris', 'iris', 'Petal.Length', pred.var = 'my_prediction')
exe(x)
print(my_prediction)
```

---

rlr_type *rlr_type*

Description

returns the name of the penalty according to the alpha.

Usage

```
rlr_type(alpha_rlr = options_regressor("rlr.alpha"))
```

Arguments

alpha_rlr	the penalty is defined as alpha=1 is the lasso penalty, and alpha=0 the ridge penalty.

See Also

glmnet

Examples

```
rlr_type(1)
rlr_type(0)
```

`rl_coeff` *rl_coeff*

Description

generates the code to get the information of the coefficients of the linear regression model

Usage

```
rl_coeff(model.var = "modelo.rl")
```

Arguments

`model.var`	the name of the variable that stores the resulting model.

Examples

```
x <- rl_model('iris', 'Petal.Length', 'model_rl')
exe(x)
print(model_rl)

x <- rl_coeff('model_rl')
exe(x)

print(df.rl)
print(r2)
```

`rl_model` *rl_model*

Description

generates the code to create the linear regression model.

Usage

```
rl_model(
  data = "datos.aprendizaje",
  variable.pred = NULL,
  model.var = "modelo.rl"
)
```

Arguments

data	the name of the learning data.
variable.pred	the name of the variable to be predicted.
model.var	the name of the variable that stores the resulting model.

See Also

lm

Examples

```
x <- rl_model('iris', 'Petal.Length')
exe(x)
print(modelo.rl)
```

---

rl_prediction *rl_prediction*

Description

generates the code to create the prediction of the linear regression model.

Usage

```
rl_prediction(
  data = "datos.prueba",
  model.var = "modelo.rl",
  pred.var = "prediccion.rl"
)
```

Arguments

data	the name of the test data.
model.var	the name of the variable that stores the resulting model.
pred.var	the name of the variable that stores the resulting prediction.

See Also

predict

Examples

```
x <- rl_model('iris', 'Petal.Length', 'model_rl')
exe(x)
print(model_rl)

x <- rl_prediction('iris', 'model_rl', 'my_prediction')
exe(x)
print(my_prediction)
```

---

summary_indices *summary_indices*

Description

summarizes a variable by returning the minimum, first quartile, third quartile and maximum value.

Usage

```
summary_indices(data)
```

Arguments

data	a numeric vector.

Examples

```
summary_indices(iris$Sepal.Length)
```

---

svm_model *svm_model*

Description

generates the code to create the support vector machines model.

Usage

```
svm_model(
  data = "datos.aprendizaje",
  variable.pred = NULL,
  model.var = "modelo.svm",
  scale = TRUE,
  kernel = "linear"
)
```

Arguments

data	the name of the learning data.
variable.pred	the name of the variable to be predicted.
model.var	the name of the variable that stores the resulting model.
scale	the scale parameter of the model.
kernel	the kernel parameter of the model.

See Also

svm

Examples

```
library(e1071)
x <- svm_model('iris', 'Petal.Length')
exe(x)
print(modelo.svm)
```

---

svm_prediction *svm_prediction*

Description

generates the code to create the prediction of the support vector machines model.

Usage

```
svm_prediction(
  data = "datos.prueba",
  variable.pred = NULL,
  model.var = "modelo.svm",
  pred.var = "prediccion.svm"
)
```

Arguments

data	the name of the test data.
variable.pred	the name of the variable to be predicted.
model.var	the name of the variable that stores the resulting model.
pred.var	the name of the variable that stores the resulting prediction.

Examples

```
library(e1071)
library(dplyr)

x <- svm_model('iris', 'Petal.Length', model.var = 'model_svm')
exe(x)
print(model_svm)

x <- svm_prediction('iris', 'Petal.Length', 'model_svm', 'my_prediction')
exe(x)
print(my_prediction)
```

---

tb_predic *tb_predic*

Description

creates comparison table between prediction and real data (test data).

Usage

```
tb_predic(real, predic.var)
```

Arguments

real	a data.frame with the real values.
predic.var	a vector with the prediction value.

Examples

```
if(interactive()) {
  library(shiny)
  library(DT)
  shinyApp(
    ui = fluidPage(fluidRow(column(12, DTOutput('tbl')))),
   server = function(input, output) {
     real <- iris[,'Petal.Width',drop = F]
     pred <- sample(iris$Petal.Width, nrow(iris), replace =  T)
     output$tbl = DT::renderDT(tb_predic(real, pred))
   })
}
```

translate *translate*

Description

translates text id into current language.

Usage

```
translate(text, language = options_regressor("language"))
```

Arguments

text	the id for the text.
language	the language to choose. It can be "es" or "en".

Examples

```
translate("knnl")
translate("knnl", "en")
```

validate_pn_data *validate_pn_data*

Description

verify that a data.frame has the same columns with the same types.

Usage

```
validate_pn_data(x, y, var.pred = "")
```

Arguments

x	a data.frame with criteria to compare.
y	a data.frame to be comprared.
var.pred	a vector with the names of variables to be excluded from the comparison.

Examples

```
## Not run:
validate_pn_data(iris, cars)
validate_pn_data(iris, iris)
x <- iris
x$Species <- as.numeric(x$Species)
validate_pn_data(iris, x)

## End(Not run)
```

`var_categorical` *var_categorical*

Description

gets only the categorical columns.

Usage

```
var_categorical(data)
```

Arguments

`data`	the dataset.

Value

a vector with the names of the categorical columns.

Examples

```
var_categorical(iris)
```

`var_numerical` *var_numerical*

Description

gets only the numerical columns.

Usage

```
var_numerical(data)
```

Arguments

data	the dataset.

Value

a vector with the names of the numerical columns.

Examples

```
var_numerical(iris)
```

---

word_report *word_report*

Description

creates a header for the report that allows you to generate a word file.

Usage

```
word_report(
  title = "Sin Titulo",
  name = "PROMiDAT",
  order_by_regressor = TRUE,
  extra = ""
)
```

Arguments

title	report title.
name	name of the author of the report.
order_by_regressor	it's the order for the default "regressoR" report.
extra	a string with any extra code you want to add to the configuration chunk.

Examples

```
new_report(iris, 'iris')

new_section_report()
insert_report('1_part', 'Title 1', 'head(iris)\n', 'summary(iris)')

new_section_report()
insert_report('1_part', 'Title 1', 'head(iris)\n', 'summary(iris)')

word_report(order_by_regressor = FALSE)
```

Index