Package 'symbolicr'

April 21, 2026

Title Symbolic Regression Framework

Description Find non-linear formulas that fits your input data. You can systematically explore and memorize the possible formulas and it's cross-validation performance, in an incremental fashion. Three main interoperable search functions are available: 1) random.search() performs a random exploration, 2) genetic.search() employs a genetic optimization algorithm, 3) comb.search() combines best results of the first two. For more details see Tomasoni et al. (2026) <doi:10.1208/s12248-026-01232-z>.

Version 1.0.0

URL https://github.com/cosbi-research/symbolicr

Encoding UTF-8

RoxygenNote 7.3.3

Imports data.table, GA, gtools, RcppAlgos, stats, stringr

Suggests knitr, rmarkdown, ggplot2

VignetteBuilder knitr

License AGPL (>= 3)

BugReports https://github.com/cosbi-research/symbolicr/issues

NeedsCompilation no

Author Danilo Tomasoni [aut, cre] (ORCID:
<https://orcid.org/0000-0002-8427-4230>),
Fondazione The Microsoft Research - University of Trento Centre for
Computational and Systems Biology COSBI [cph, fnd]

Maintainer Danilo Tomasoni <tomasoni@cosbi.eu>

Repository CRAN

Date/Publication 2026-04-21 21:00:02 UTC

Contents

`analyze.variables` *Analyze Variables Get information on the relative importance of each variable included in your formula.*

Description

This works by removing one variable at a time from the formula and measuring its new (cross-validation) fitness.

Usage

```
analyze.variables(
  regressors.df,
  y,
  test.formula.df,
  fitness.column,
  transformations,
  transformations_replacement_map,
  custom.abs.mins = list(),
  K = 7,
  N = 10,
  direction = "max",
  max.formula.len = max.formula.len,
  fitness.fun = pe.r.squared.formula.len.fitness,
  cv.norm = TRUE
)
```

Arguments

regressors.df	The dataset that contains the base variables the formula is composed of (column-wise)
y	The independent variable
test.formula.df	A dataset of formulas (contained in 'vars' column) and their fitness values (error measures)
fitness.column	The test.formula.df column containing the target fitness value to be used for analysis
transformations	A list of potentially non-linear transformations that can be applied on top of the squares. Ex. `order 0, transformation=log10 = log10.a`. Input values are x and z, the array of numbers to be transformed (training set only), and the min max statistics (on the global dataset) respectively.
transformations_replacement_map	A list of rules to remove a variable from a term. Ex. `"log_empty_well_p"=c("log10", "empty_well")`
custom.abs.mins	A list of user-defined minimum values for dataset columns.
K	The number of parts the dataset is split into for K-fold cross-validation.
N	The number of times the K-fold validation is repeated, shuffling the dataset row orders before each time.
direction	'max' if the higher the objective in fitness.column the better, 'min' on the contrary.
max.formula.len	The maximum number of terms in the formula
fitness.fun	The function that determine the fitness of a given formula. Defaults to `pe.r.squared.formula.len.fitr`
cv.norm	Normalize regressors after train-validation split in inner cross-validation loop.

Value

A list of "terms that"loss" and "gain" variables, with a measure of the (relative) gain/loss of removing each variable from the initial formula.

Examples

```
# set-up a toy example dataset
x1<-runif(100, min=2, max=67)
x2<-runif(100, min=0.01, max=0.1)
X <- data.frame(x1=x1, x2=x2)
# set up a "true" non-linear relationship
# with some noise
y <- log10(x1^2*x2) + rnorm(100, 0, 0.001)

max.formula.len=1
transformations=list(
```

```
  "log"=function(rdf, x, stats){ log(x) },
  "log_x1_p"=function(rdf, x, stats){ log(rdf$x1 + x) },
  "inv"=function(rdf, x, stats){ 1/x }
)
random.results <- random.search(
  X, y,
  n.squares=2,
  formula.len = max.formula.len,
  N=2,
  K=10,
  transformations = transformations,
  cv.norm=FALSE
)
# compute a unique objective function
random.results$obj <- apply(random.results, MARGIN=1,
 FUN=function(row) pe.r.squared.formula.len.fitness(as.data.frame(t(row)), max.formula.len)
)

# sort by top-N functions (according to obj)
ordered.res <- random.results[order(random.results$obj,decreasing=TRUE),]

# max fitness on all computed formulas
best.obj <- ordered.res[1,'obj']
# analyze top-10 formulas
# select formulas according to criterion above
eligible.res <- ordered.res[seq(10), ]

direction = 'max' # obj should be maximized
sensitivity <- analyze.variables(
  X, y, eligible.res, fitness.column='obj',
  # a list of available term transformations
  transformations=transformations,
  # a list of rules to remove a variable from a term
  # ex.
  #   orig transformation -> base transformation, removed term
  #   "log_empty_well_p"=c("log10", "empty_well")
  transformations_replacement_map=list(
    "log_x1_p"=c("log", "x1")
  ),
  custom.abs.mins=list(),
  K=10,
  N=2,
  direction=direction,
  max.formula.len=max.formula.len,
  fitness.fun=pe.r.squared.formula.len.fitness,
  cv.norm=FALSE
)

# plottable data.frame of quantile losses per-variable
variable.importance.df <- sensitivity[['var.imp']]
```

comb.search *Explore given combinations of formula terms*

Description

Given a set of formulas, this function systematically evaluates all combinations of formula terms, of a given length.

Usage

```
comb.search(
  complete.X.df,
  y,
  combinations,
  K = 7,
  N = 10,
  seed = NULL,
  transformations = list(log10 = function(rdf, x, z) {
     log10(0.1 + abs(z) + x)
 },
    inv = function(rdf, x, z) {
     1/(0.1 + abs(z) + x)
 }),
  custom.abs.mins = list(),
  cv.norm = FALSE
)
```

Arguments

complete.X.df	The dataset that contains the base variables the formula is composed of (column-wise)
y	The independent variable to be predicted with the formula
combinations	A data.frame of combinations of shape (num.combinations, formula.len)
K	The number of parts the dataset is split into for K-fold cross-validation.
N	The number of times the K-fold validation is repeated, shuffling the dataset row orders before each time.
seed	An (optional) seed for deterministic run
transformations	A list of potentially non-linear transformations that can be applied on top of the squares. Ex. `order 0, transformation=log10 = log10.a`
custom.abs.mins	A list of user-defined minimum values for dataset columns.
cv.norm	Normalize regressors after train-validation split in inner cross-validation loop.

Value

A data.frame of formulas and the corresponding cross-validation performance measures (R-squared, absolute relative error, max cooks distance). See also `empty.sample`.

Examples

```
# set-up a toy example dataset
x1<-runif(100, min=2, max=67)
x2<-runif(100, min=0.01, max=0.1)
X <- data.frame(x1=x1, x2=x2)
# set up a "true" non-linear relationship
# with some noise
y <- log10(x1^2*x2) + rnorm(100, 0, 0.001)

max.formula.len=formula.len=1
n.squares=1
K=2
N=2
seed=1001
transformations=list(
  "log"=function(rdf, x, stats){ log(x) },
  "log_x1_p"=function(rdf, x, stats){ log(rdf$x1 + x) },
  "inv"=function(rdf, x, stats){ 1/x }
)
complete.regressors <- regressors.names(X, n.squares, transformations)
# compute combinations up to length formula.len
regressors.list <- lapply(seq(formula.len), function(x) complete.regressors)
combinations <- RcppAlgos::comboGrid(regressors.list, repetition = FALSE)
combinations <- apply(combinations, MARGIN=1, FUN=function(row){
  paste(row, collapse=",")
})
names(combinations)<-NULL
# get missing formulas
# missing <- setdiff(combinations, res$vars)
missing <- combinations
# compute exaustively all missing formulas
res.new <- comb.search(X, y,
                       # data.frame of n.missing.values x formula.len
                       combinations=t(as.data.frame(strsplit(missing,",",fixed=TRUE))),
                       K=K, N=N, seed=seed,
                  transformations=transformations, custom.abs.mins=list(), cv.norm=TRUE)
```

`cross.validate` *Test a non-linear formula*

Description

Test using K-fold cross-validation repeated N times

Usage

```
cross.validate(
  cur.dataset,
  y,
  cur.vars,
  custom.abs.mins,
  K,
  N,
  transformations,
  cv.norm
)
```

Arguments

`cur.dataset`	The dataset to be used for the test
`y`	The independent variable
`cur.vars`	An array of non-linear formula terms to be tested. `cur.vars <- c('a','mul.a.b')` will test the formula `y ~ a + a*b`
`custom.abs.mins`	A list of user-defined minimum values for dataset columns.
`K`	The number of parts the dataset is split into for K-fold cross-validation.
`N`	The number of times the K-fold validation is repeated, shuffling the dataset row orders before each time.
`transformations`	A list of potentially non-linear transformations allowed in `cur.vars`.
`cv.norm`	Normalize regressors after train-validation split in inner cross-validation loop.

Value

A data.frame with cross-validation results for each fold (K) and for each round (N).

Examples

```
# set-up a toy example dataset
x1<-runif(100, min=2, max=67)
x2<-runif(100, min=0.01, max=0.1)
X <- data.frame(x1=x1, x2=x2)
# set up a "true" non-linear relationship
# with some noise
y <- log10(x1^2*x2) + rnorm(100, 0, 0.001)

# do actual cross-validation experiments, record in experiments data.frame
# all the validation-set performances.
experiments.0 <- cross.validate(
    X, y,
    cur.vars=c('inv.x1','x2'),
    custom.abs.mins=list(),
    K=2,
```

```
    N=3,
   transformations=list(
      "log"=function(rdf, x, stats){ log(x) },
      "log_x1_p"=function(rdf, x, stats){ log(rdf$x1 + x) },
      "inv"=function(rdf, x, stats){ 1/x }
    ),
    cv.norm=FALSE
)

# summarize cross-validation results by averaging
errs.m <- stats::aggregate(
    cbind(base.pe, base.cor, base.r.squared,
          base.max.pe, base.iqr.pe, base.max.cooksd)~1,
   data=experiments.0, FUN=mean
)
```

`dataset.min.maxs` *Compute dataset column-wise statistics: min, absmin, absmax, projzero*

Description

Compute dataset column-wise statistics: min, absmin, absmax, projzero

Usage

```
dataset.min.maxs(X.df.std, X.mean.sd)
```

Arguments

`X.df.std`	input dataset
`X.mean.sd`	pre-computed column-wise statistics for the dataset, mean and sd

Value

A list of statistics for each column.

min: the minimum of the column values absmin: the minimum of the absolute values of the columns absmax: the maximum of the absolute values of the columns projzero: -mean/sd of the columns, that is the position of the zero in the original, non-normalized space.

empty.sample — *Return an empty data.frame with the columns returned by* random.search

Description

Return an empty data.frame with the columns returned by random.search

Usage

```
empty.sample()
```

Value

An empty data.frame

See Also

random.search

exaustive.search — *Explore all combinations of formula terms*

Description

Explore all combinations of formula terms

Usage

```
exaustive.search(
  complete.X.df,
  y,
  n.squares = 1,
  formula.len = 3,
  K = 7,
  N = 10,
  seed = NULL,
  transformations = list(log10 = function(rdf, x, z) {
     log10(0.1 + abs(z) + x)
 },
    inv = function(rdf, x, z) {
     1/(0.1 + abs(z) + x)
 }),
  custom.abs.mins = list(),
  glob.filepath = NULL,
  chunk.size = NULL,
  cv.norm = FALSE
)
```

Arguments

complete.X.df	The dataset that contains the base variables the formula is composed of (column-wise)
y	The independent variable to be predicted with the formula
n.squares	The maximum order of the polynomial composition of base variables. Ex. order 0 = a, order 1 = a*b, order 2 = a*b*c
formula.len	The number of terms in the formulas that will be randomly sampled.
K	The number of parts the dataset is split into for K-fold cross-validation.
N	The number of times the K-fold validation is repeated, shuffling the dataset row orders before each time.
seed	An (optional) seed for deterministic run
transformations	A list of potentially non-linear transformations that can be applied on top of the squares. Ex. order 0, transformation=log10 = log10.a
custom.abs.mins	A list of user-defined minimum values for dataset columns.
glob.filepath	The path to an rDdata object containing the results of potentially multiple independent previous run, that will be excluded from the current run. The file will be automatically updated whith the new formula evaluation results.
chunk.size	If null, compute all missing formulas. If a number, compute only that number of missing formulas.
cv.norm	Normalize regressors after train-validation split in inner cross-validation loop.

Value

A data.frame of formulas and the corresponding cross-validation performance measures (R-squared, absolute relative error, max cooks distance). See also empty.sample.

Examples

```
# set-up a toy example dataset
x1<-runif(100, min=2, max=67)
x2<-runif(100, min=0.01, max=0.1)
X <- data.frame(x1=x1, x2=x2)
# set up a "true" non-linear relationship
# with some noise
y <- log10(x1^2*x2) + rnorm(100, 0, 0.001)

max.formula.len=formula.len=1
n.squares=1
K=2
N=2
seed=1001
res.new <- exaustive.search(X, y,
                            n.squares=1,
                            formula.len=3,
                            K=2, N=3, seed=NULL,
```

```
                transformations=list(
  "log"=function(rdf, x, stats){ log(x) },
  "log_x1_p"=function(rdf, x, stats){ log(rdf$x1 + x) },
  "inv"=function(rdf, x, stats){ 1/x }
                ),
                custom.abs.mins=list(),
                glob.filepath = NULL,
                chunk.size=NULL, cv.norm=FALSE)
```

---

genetic.search *Genetic Algorithm for non-linear formula optimization*

Description

Starting from an (optional) list of promising formulas, generated with other methods such as `random.search`, explore combinations (crossover) and alterations (mutation) of them in order maximize the value of the fitness function that defaults to `pe.r.squared.formula.len.fitness`

Usage

```
genetic.search(
  complete.X.df,
  y,
  n.squares = 1,
  max.formula.len = 4,
  seed = NULL,
  fitness.fun = pe.r.squared.formula.len.fitness,
  transformations = list(log10 = function(rdf, x, z) {
     log10(0.1 + abs(z$min) + x)

    }, inv = function(rdf, x, z) {
     1/(0.1 + abs(z$min) + x)
 }),
  custom.abs.mins = list(),
  glob.filepath = NULL,
  local.filepath = NULL,
  memoization = FALSE,
  monitor = monitor.formula.fun,
  maxiter = 100,
  N = 2,
  K = 7,
  pcrossover = 0.2,
  popSize = 50,
  pmutation = 0.8,
  keepBest = FALSE,
  cv.norm = FALSE,
  best.vars.l = list()
)
```

Arguments

complete.X.df	The dataset that contains the base variables the formula is composed of (column-wise)
y	The independent variable to be predicted with the formula
n.squares	The maximum order of the polynomial composition of base variables. Ex. order 0 = a, order 1 = a*b, order 2 = a*b*c
max.formula.len	The maximum number of terms in the formula
seed	An (optional) seed for deterministic run
fitness.fun	The function that determine the fitness of a given formula. Defaults to pe.r.squared.formula.len.fitr
transformations	A list of potentially non-linear transformations that can be applied on top of the squares. Ex. order 0, transformation=log10 = log10.a. Input values are x and z, the array of numbers to be transformed (training set only), and the min max statistics (on the global dataset) respectively.
custom.abs.mins	A list of user-defined minimum values for dataset columns.
glob.filepath	Has effect only if memoization=TRUE. The path to an rDdata object containing the results of potentially multiple independent previous run.
local.filepath	Has effect only if memoization=TRUE. The path to an rData object where the results of the current run will be stored. If it already exists, the new results will be appended.
memoization	If TRUE test results will be stored in res.filepath
monitor	Function that will be called on every iteration with the current solutions. Defaults to monitor.formula.fun
maxiter	Maximum number of genetic evolution epochs
N	The number of times the K-fold validation is repeated, shuffling the dataset row orders before each time.
K	The number of parts the dataset is split into for K-fold cross-validation.
pcrossover	The probability of crossover between pairs of chromosomes. Typically this is a large value and by default is set to 0.8.
popSize	The population size, the number of formulas considered for genetic evolution.
pmutation	The probability of mutation in a parent chromosome. Usually mutation occurs with a small probability, and by default is set to 0.1.
keepBest	If TRUE, the value best.iter of the list returned, will contain a list of the best formulas at each evolution iteration.
cv.norm	Normalize regressors after train-validation split in inner cross-validation loop.
best.vars.l	A list of formulas. Each formula is an array of strings, each string is the textual representation of a formula term. Ex. cur.vars.l <- list(c('a','mul.a.b')) will test the formula y ~ a + a*b. This list is used as a starting point for genetic evolution. It may come from a-priori knowledge or by extracting the most promising results from random.search.

Value

A list with three values: list(best: the best formula found overall best.iter: a list of best formulas, one for each evolution iteration results: The output of the GA::ga function, with all the solutions, hyperparameters, etc.)

See Also

random.search

Examples

```
# set-up a toy example dataset
x1<-runif(100, min=2, max=67)
x2<-runif(100, min=0.01, max=0.1)
X <- data.frame(x1=x1, x2=x2)
# set up a "true" non-linear relationship
# with some noise
y <- log10(x1^2*x2) + rnorm(100, 0, 0.001)

# stats:
#    list(
#       "min"=..,
#       "absmin"=..,
#       # zero in original space projected in std space
#       "projzero"=prjzero
#    )
transformations=list(
  "log"=function(rdf, x, stats){ log(x) },
  "log_x1_p"=function(rdf, x, stats){ log(rdf$x1 + x) },
  "inv"=function(rdf, x, stats){ 1/x }
)

# empty datastore
datastore='test.rData'
genetic.datastore='test.genetic.rData'
random.datastore='test.random.rData'
saveRDS(empty.sample(), datastore)
saveRDS(empty.sample(), genetic.datastore)
saveRDS(empty.sample(), random.datastore)

random.res <- random.search(
     X, y,
     n.squares=2,
     formula.len=1,
     maxiter=10,
     # formulas from previous runs
     glob.filepath = datastore,
     # formulas computed by this call
     local.filepath = random.datastore,
     transformations = transformations,
     memoization=TRUE
 )
```

```
res <- readRDS(datastore)
# check random.res...
# append to global results to avoid recompute
res <- rbind(res, random.res)
saveRDS(res, datastore)

# search more with genetic algorithm without
# re-analyzing already analyzed formulas
# stored in datastore
best.finetuned <- genetic.search(
 X,
 y,
 n.squares=1,
 maxiter=10,
 glob.filepath=datastore,
 local.filepath=genetic.datastore,
 memoization=TRUE,
 pcrossover=0.2,
 pmutation=0.8,
 seed=NULL,
 max.formula.len=4,
 keepBest=TRUE,
 K=2,
 N=3,
 popSize = 5,
 fitness.fun=pe.r.squared.formula.len.fitness,
 # multiple best initial guess
 best.vars.l = list(
       c('mul.x1.x2','x2')
 )
)

file.remove(datastore)
file.remove(genetic.datastore)
file.remove(random.datastore)
```

`monitor.formula.fun` *Default formula to monitor genetic evolution*

Description

Default formula to monitor genetic evolution

Usage

```
monitor.formula.fun(obj)
```

Arguments

obj	The solution object of the GA package.

Value

No return value, prints out current best fitness value for tracing purposes.

See Also

genetic.search

---

normalize *Normalize a dataset*

Description

Normalize a dataset

Usage

```
normalize(X.df, custom.mins)
```

Arguments

X.df	The input dataset to be normalized
custom.mins	A list of user-defined minimum values for dataset columns.

Value

A list with two values:

```
The input dataset normalized by subtracting the mean and dividing by the standard deviation.

 A list with some column statistic: std.min, mean, sd
```

Examples

```
# set-up a toy example dataset
x1<-runif(100, min=2, max=67)
x2<-runif(100, min=0.01, max=0.1)
X <- data.frame(x1=x1, x2=x2)
# set up a "true" non-linear relationship
# with some noise
y <- log10(x1^2*x2) + rnorm(100, 0, 0.001)
regressors <- c('x1','x2')
best.vars <- c('inv.mul.x1.x2','mul.x1.x2','x2')

transformations=list(
```

```
  "log"=function(rdf, x, stats){ log(x) },
  "log_x1_p"=function(rdf, x, stats){ log(rdf$x1 + x) },
  "inv"=function(rdf, x, stats){ 1/x }
)

# parse variables
parsed.vars <- symbolicr::parse.vars(best.vars, regressors, transformations)
# standardize
norm.res <- symbolicr::normalize(X, custom.mins=list())
X.std <- norm.res$X.std
X.mean.sd <-norm.res$mean.sd
# compute regressors
X.def <- symbolicr::regressors(X.std, parsed.vars,
                  transformations, X.mean.sd, regressors.min.values=NULL)
X.min.values <- X.def$min.values
formula.df <- X.def$regressors

# extract coefficients of given formula
dataset.std <- cbind(formula.df, y)
cur.formula.str <- paste0('y',"~",paste(best.vars, collapse=' + '))
base.lm <- lm(as.formula(cur.formula.str), data=formula.df)
base.lm$coefficients
```

`normalize.test` *Normalize a dataset with pre-defined column mean and standard deviation*

Description

Normalize a dataset with pre-defined column mean and standard deviation

Usage

```
normalize.test(X.df, mean.sd)
```

Arguments

`X.df`	The input dataset to be normalized
`mean.sd`	list of user-defined mean and sd values for every dataset column.

Value

The input dataset normalized by subtracting the mean and dividing by the standard deviation.

parse.vars *Parse text representation of a non-linear formula term*

Description

Return a list with the non-linear function applied and the base terms of the formula.

Usage

```
parse.vars(cur.vars, base.regressors, transformations = list())
```

Arguments

cur.vars	An array of non-linear formula terms. Ex. `cur.vars <- c('a','mul.a.b')` represents the formula `y ~ a + a*b`
base.regressors	The `names(X)` where X is the input dataset.
transformations	A list of potentially non-linear transformations allowed in `cur.vars`.

Value

A list with the base regressors and the transformation function to be applied

See Also

random.search

genetic.search

serialize.vars

Examples

```
parse.vars(
 c('log.a', 'mul.a.mul.b.c'),
 base.regressors=c('a','b','c'),
 transformations=list('log'=log)
)
```

`pe.r.squared.formula.len.fitness` *Default fitness function for* `genetic.search`

Description

Default fitness function for `genetic.search`

Usage

```
pe.r.squared.formula.len.fitness(errs.m, max.formula.len)
```

Arguments

`errs.m`	a 1-row data.frame with cross-validation results
`max.formula.len`	the maximum length of the formula we are searching for

Value

A single double value, the bigger the value the better the formula.

Examples

```
# set-up a toy example dataset
x1<-runif(100, min=2, max=67)
x2<-runif(100, min=0.01, max=0.1)
X <- data.frame(x1=x1, x2=x2)
# set up a "true" non-linear relationship
# with some noise
y <- log10(x1^2*x2) + rnorm(100, 0, 0.001)

genetic.search(
 X,
 y,
 n.squares=1,
 maxiter=10,
 glob.filepath=NULL,
 local.filepath=NULL,
 memoization=FALSE,
 pcrossover=0.2,
 pmutation=0.8,
 seed=NULL,
 max.formula.len=2,
 keepBest=TRUE,
 K=2,
 N=3,
 popSize = 5,
 fitness.fun=pe.r.squared.formula.len.fitness,
```

```
 best.vars.1 = list(
       c('x1','x2')
 )
)
```

`pred.vs.obs` *Plot "predicted vs observed" using a given formula*

Description

Predicted values are obtained out-of-sample using K-fold cross-validation repeated N times and averaged.

Usage

```
pred.vs.obs(
  complete.X.df,
  y,
  cur.vars,
  custom.abs.mins,
  K,
  N,
  transformations = list(log10 = function(rdf, x, z) {
     log10(0.1 + abs(z$min) + x)

    }, inv = function(rdf, x, z) {
     1/(0.1 + abs(z$min) + x)
 }),
  cv.norm = TRUE,
  errors.x = 3.2,
  errors.y = 5,
  with.names = FALSE
)
```

Arguments

`complete.X.df`	The dataset to be used for the test
`y`	The independent variable
`cur.vars`	An array of non-linear formula terms to be tested. `cur.vars <- c('a','mul.a.b')` will test the formula `y ~ a + a*b`
`custom.abs.mins`	A list of user-defined minimum values for dataset columns.
`K`	The number of parts the dataset is split into for K-fold cross-validation.
`N`	The number of times the K-fold validation is repeated, shuffling the dataset row orders before each time.

transformations	A list of potentially non-linear transformations allowed in cur.vars.
cv.norm	Normalize regressors after train-validation split in inner cross-validation loop.
errors.x	position of R^2 and PE in plot (x axis)
errors.y	position of R^2 and PE in plot (y axis)
with.names	Plot also product name

Value

ggplot2::ggplot object ready to be plotted

Examples

```
# set-up a toy example dataset
x1<-runif(100, min=2, max=67)
x2<-runif(100, min=0.01, max=0.1)
X <- data.frame(x1=x1, x2=x2)
# set up a "true" non-linear relationship
# with some noise
y <- log10(x1^2*x2) + rnorm(100, 0, 0.001)
n.squares=1
K=2
N=2
seed=1001

transformations=list(
  "log"=function(rdf, x, stats){ log(x) },
  "log_x1_p"=function(rdf, x, stats){ log(rdf$x1 + x) },
  "inv"=function(rdf, x, stats){ 1/x }
)

best.f <- list(
  c('inv.mul.x1.x2','x1'),
  c('inv.mul.x1.x2','x2')
)
# analyze different formulas with predicted vs observed plot.
# the more data points align round the x=y line, the more
# the prediction of the corresponding formula is accurate
plots <- lapply(best.f, function(test.f){
    pred.vs.obs(X,
                y, test.f,
                list(), K, N,
                transformations,
                cv.norm = FALSE,
                errors.x=2.5, errors.y=0.5
   )
})
```

`random.search` *Random search for non-linear formula optimization*

Description

Randomly sample and test different formulas with cross-validation.

Usage

```
random.search(
  complete.X.df,
  y,
  n.squares = 1,
  formula.len = 3,
  K = 7,
  N = 10,
  seed = NULL,
  transformations = list(log10 = function(rdf, x, z) {
     log10(0.1 + abs(z$min) + x)

    }, inv = function(rdf, x, z) {
     1/(0.1 + abs(z$min) + x)
 }),
  custom.abs.mins = list(),
  maxiter = 100,
  glob.filepath = NULL,
  local.filepath = NULL,
  memoization.interval = 50,
  memoization = FALSE,
  cv.norm = FALSE
)
```

Arguments

`complete.X.df`	The dataset that contains the base variables the formula is composed of (column-wise)
`y`	The independent variable to be predicted with the formula
`n.squares`	The maximum order of the polynomial composition of base variables. Ex. `order 0 = a`, `order 1 = a*b`, `order 2 = a*b*c`
`formula.len`	The number of terms in the formulas that will be randomly sampled.
`K`	The number of parts the dataset is split into for K-fold cross-validation.
`N`	The number of times the K-fold validation is repeated, shuffling the dataset row orders before each time.
`seed`	An (optional) seed for deterministic run

`transformations`
: A list of potentially non-linear transformations that can be applied on top of the squares. Ex. `order 0, transformation=log10 = log10.a`. Input values are x and z, the array of numbers to be transformed (training set only), and the min max statistics (on the global dataset) respectively.

`custom.abs.mins`
: A list of user-defined minimum values for dataset columns.

`maxiter`
: Maximum number of genetic evolution epochs

`glob.filepath`
: Has effect only if memoization=TRUE. The path to an rDdata object containing the results of potentially multiple independent previous run.

`local.filepath`
: Has effect only if memoization=TRUE. The path to an rData object where the results of the current run will be stored. If it already exists, the new results will be appended.

`memoization.interval`
: The number of formulas to sample at each iteration, and the frequency of update of `res.filepath` if memoization=TRUE.

`memoization`
: If TRUE test results will be stored in `local.filepath`

`cv.norm`
: Normalize regressors after train-validation split in inner cross-validation loop.

Value

A data.frame of formulas and the corresponding cross-validation performance measures (R-squared, absolute relative error, max cooks distance). See also `empty.sample`.

See Also

genetic.search

cross.validate

empty.sample

Examples

```
# set-up a toy example dataset
x1<-runif(50, min=2, max=67)
x2<-runif(50, min=0.01, max=0.1)
X <- data.frame(x1=x1, x2=x2)
# set up a "true" non-linear relationship
# with some noise
y <- log10(x1^2*x2) + rnorm(50, 0, 0.001)

# stats:
#    list(
#       "min"=..,
#       "absmin"=..,
#       # zero in original space projected in std space
#       "projzero"=prjzero
#    )
transformations=list(
```

```
  "log"=function(rdf, x, stats){ log(x) },
  "log_x1_p"=function(rdf, x, stats){ log(rdf$x1 + x) }
)

random.res <- random.search(
     X, y,
     n.squares=1,
     formula.len=1,
     maxiter=1,
     # formulas from previous runs
     glob.filepath = NULL,
     # formulas computed by this call
     local.filepath = NULL,
     transformations = transformations,
     memoization=FALSE
 )
```

`regressors` *Compute regressors of a given non-linear formula*

Description

Compute regressors of a given non-linear formula

Usage

```
regressors(
  base.X.df.std,
  parsed.vars,
  transformations,
  X.mean.sd,
  regressors.min.values = NULL
)
```

Arguments

`base.X.df.std`	A data.frame with the base variables that compose the terms of the non-linear formula
`parsed.vars`	The results of the symbolicr::parsed.vars
`transformations`	A list of potentially non-linear transformations that can be used in the formula terms.
`X.mean.sd`	The `mean.sd` list entry of the returned value of `normalize`
`regressors.min.values`	The output of `dataset.min.maxs`. If NULL, minimum values are computed on `base.X.df.std`. Use this parameter if you want to fix the minimum values, for instance if you compute it in a subset of the full dataset, as happens in the cross-validation.

Value

Return a data.frame with the columns corresponding to each formula term.

See Also

random.search

genetic.search

Examples

```
# set-up a toy example dataset
x1<-runif(100, min=2, max=67)
x2<-runif(100, min=0.01, max=0.1)
X <- data.frame(x1=x1, x2=x2)
# set up a "true" non-linear relationship
# with some noise
y <- log10(x1^2*x2) + rnorm(100, 0, 0.001)
regressors <- c('x1','x2')
best.vars <- c('inv.mul.x1.x2','mul.x1.x2','x2')

transformations=list(
  "log"=function(rdf, x, stats){ log(x) },
  "log_x1_p"=function(rdf, x, stats){ log(rdf$x1 + x) },
  "inv"=function(rdf, x, stats){ 1/x }
)

# parse variables
parsed.vars <- symbolicr::parse.vars(best.vars, regressors, transformations)
# standardize
norm.res <- symbolicr::normalize(X, custom.mins=list())
X.std <- norm.res$X.std
X.mean.sd <-norm.res$mean.sd
# compute regressors
symbolicr::regressors(X.std, parsed.vars,
                  transformations, X.mean.sd, regressors.min.values=NULL)
```

---

regressors.names *Compute all possible formula terms from a given dataset*

Description

Compute all possible formula terms from a given dataset

Usage

```
regressors.names(complete.X.df, n.squares, transformations)
```

Arguments

`complete.X.df`	The dataset that contains the base variables the formula is composed of (column-wise)
`n.squares`	The maximum order of the polynomial composition of base variables. Ex. `order 0 = a`, `order 1 = a*b`, `order 2 = a*b*c`
`transformations`	A list of potentially non-linear transformations that can be applied on top of the squares. Ex. `order 0, transformation=log10 = log10.a`. Input values are x and z, the array of numbers to be transformed (training set only), and the min max statistics (on the global dataset) respectively.

Value

An array of character with all the generated names for the generated variables

Examples

```
# set-up a toy example dataset
x1<-runif(100, min=2, max=67)
x2<-runif(100, min=0.01, max=0.1)
X <- data.frame(x1=x1, x2=x2)

n.squares=2

transformations=list(
  "log"=function(rdf, x, stats){ log(x) },
  "log_x1_p"=function(rdf, x, stats){ log(rdf$x1 + x) },
  "inv"=function(rdf, x, stats){ 1/x }
)

regressors <- c('x1','x2')

regressors.names(X, n.squares, transformations)
```

`serialize.vars` *Parse text representation of a non-linear formula term*

Description

Return a list with the non-linear function applied and the base terms of the formula.

Usage

```
serialize.vars(parsed.vars, base.regressors, transformations = list())
```

Arguments

`parsed.vars`	A list with the base regressors and the transformation function to be applied
`base.regressors`	The `names(X)` where X is the input dataset.
`transformations`	A list of potentially non-linear transformations allowed in `cur.vars`.

Value

An array of serialized non-linear formula terms. Ex. `cur.vars <- c('a','mul.a.b')` represents the formula `y ~ a + a*b`

See Also

random.search

genetic.search

parse.vars

Examples

```
serialize.vars(
 list(
   list("transformation.name"="log", "prods"=c("a")),
   list("transformation.name"=NULL, "prods"=c("a","b","c"))
 ),
 base.regressors=c('a','b','c'),
 transformations=list('log'=log)
)
```

---

test.formula *Test a non-linear formula, get an aggregated result.*

Description

Test using K-fold cross-validation repeated N times

Usage

```
test.formula(
  complete.X.df,
  y,
  cur.vars,
  custom.abs.mins,
  K,
  N,
  transformations = list(log10 = function(rdf, x, z) {
```

```
     log10(0.1 + abs(z$min) + x)

    }, inv = function(rdf, x, z) {
     1/(0.1 + abs(z$min) + x)
 }),
  cv.norm = TRUE
)
```

Arguments

complete.X.df	The dataset to be used for the test
y	The independent variable
cur.vars	An array of non-linear formula terms to be tested. cur.vars <- c('a','mul.a.b') will test the formula y ~ a + a*b
custom.abs.mins	A list of user-defined minimum values for dataset columns.
K	The number of parts the dataset is split into for K-fold cross-validation.
N	The number of times the K-fold validation is repeated, shuffling the dataset row orders before each time.
transformations	A list of potentially non-linear transformations allowed in cur.vars.
cv.norm	Normalize regressors after train-validation split in inner cross-validation loop.

Value

A 1-row data.frame with error metrics: base.pe, base.cor, base.r.squared, base.max.pe, base.iqr.pe, base.max.cooksd, base.max.cooksd.name

Examples

```
# set-up a toy example dataset
x1<-runif(100, min=2, max=67)
x2<-runif(100, min=0.01, max=0.1)
X <- data.frame(x1=x1, x2=x2)
# set up a "true" non-linear relationship
# with some noise
y <- log10(x1^2*x2) + rnorm(100, 0, 0.001)

 test.formula(
      X, y,
      cur.vars=c('inv.x1','x2'),
      custom.abs.mins=list(),
      K=2,
      N=3,
      transformations=list(
      # stats:
      #    list(
      #       "min"=..,
      #       "absmin"=..,
```

```
        #       # zero in original space projected in std space
        #       "projzero"=prjzero
        #    )
          "log10"=function(rdf, x, stats){
              log10(0.1+abs(stats$min)+x)
          },
          "inv"=function(rdf, x, stats){
              1/(0.1+abs(stats$min)+x)
          }
        ),
        cv.norm=FALSE
    )
```

`transformations.names` *Get transformed regressors names*

Description

Get transformed regressors names

Usage

```
transformations.names(regressors, transformations)
```

Arguments

`regressors`	array of regressors of type string
`transformations`	list of transformations

Value

array of transformed regressors

Index