Package ‘GAparsimony’
August 4, 2017

Type Package
Title Searching Parsimony Models with Genetic Algorithms
Version 0.9-1
Date 2017-07-31
Description Methodology that combines feature selection, model tuning, and parsimonious model selection with Genetic Algorithms (GA) proposed in {Martinez-de-Pison} (2015) <DOI:10.1016/j.asoc.2015.06.012>. To this objective, a novel GA selection procedure is introduced based on separate cost and complexity evaluations.
Suggests parallel, doParallel, doRNG (>= 1.6), knitr (>= 1.8), lhs, MASS, caret, mlbench, e1071, nnet, kernlab
License GPL (>= 2)
Depends R (>= 3.0), methods, foreach, iterators
Imports stats, graphics, grDevices, utils
URL https://github.com/jpison/GAparsimony
Repository CRAN
NeedsCompilation no
Author F.J. Martinez-de-Pison [aut, cre]
Maintainer F.J. Martinez-de-Pison <fjmartin@unirioja.es>
Date/Publication 2017-08-03 22:53:42 UTC

R topics documented:

  GAparsimony-package ............................................. 2
ga_parsimony ......................................................... 3
ga_parsimony-class .................................................. 13
numericOrNA-class ................................................... 15
parsimony_crossover .................................................. 15
parsimony_importance ............................................... 16
parsimony_monitor .................................................... 17
parsimony_Mutation ................................................... 18
parsimony_Population ............................................... 18
Description

Combines feature selection, model tuning, and parsimonious model selection with GA optimization. GA selection procedure is based on separate cost and complexity evaluations. Therefore, the best individuals are initially sorted by an error fitness function, and afterwards, models with similar costs are rearranged according to model complexity measurement so as to foster models of lesser complexity. The algorithm can be run sequentially or in parallel using an explicit master-slave parallelisation.

Details

GAparsimony package is a new GA wrapper automatic method that efficiently generates prediction models with reduced complexity and adequate generalization capacity. ga_parsimony function is primarily based on combining feature selection and model parameter tuning with a second novel GA selection process (ReRank algorithm) in order to achieve better overall parsimonious models. Unlike other GA methodologies that use a penalty parameter for combining loss and complexity measures into a unique fitness function, the main contribution of this package is that ga_parsimony selects the best models by considering cost and complexity separately. For this purpose, the ReRank algorithm rearranges individuals by their complexity when there is not a significant difference between their costs. Thus, less complex models with similar accuracy are promoted. Furthermore, because the penalty parameter is unnecessary, there is no consequent uncertainty associated with assigning a correct value beforehand. As a result, with GA-PARSIMONY, an automatic method for obtaining parsimonious models is finally made possible.

References

**ga_parsimony**

**Author(s)**

Francisco Javier Martinez de Pison Ascacibar <fjmartin@unirioja.es>

---

**Description**

A GA-based optimization method for searching accurate parsimonious models by combining feature selection, model tuning, and parsimonious model selection (PMS). PMS procedure is based on separate cost and complexity evaluations. The best individuals are initially sorted by an error fitness function, and afterwards, models with similar costs are rearranged according to model complexity measurement so as to foster models of lesser complexity. The algorithm can be run sequentially or in parallel using an explicit master-slave parallelisation.

**Usage**

```r
ga_parsimony(fitness, ..., 
  min_param, max_param, nFeatures, 
  names_param=NULL, names_features=NULL, 
  iter_ini=0, object=NULL, 
  type_ini_pop="improvedLHS", 
  popSize = 50, pcrossover = 0.8, maxiter = 40, 
  feat_thres=0.90, rerank_error = 0.0, iter_start_rerank = 0, 
  pmutation = 0.10, feat_mut_thres=0.10, not_muted=3, 
  elitism = base::max(1, round(popSize * 0.20)), 
  population = parsimony_population, 
  selection = parsimony_nlrSelection, 
  crossover = parsimony_crossover, 
  mutation = parsimony_mutation, 
  keep_history = FALSE, 
  early_stop = maxiter, maxFitness = Inf, suggestions = NULL, 
  parallel = FALSE, 
  monitor = if (interactive()) parsimony_monitor else FALSE, 
  seed_ini = NULL, verbose=FALSE)
```

**Arguments**

- **fitness**: the fitness function, any allowable R function which takes as input an individual chromosome which combines the model parameters to tune and the features to be selected. Fitness function returns a numerical vector with three values: "validation_cost": a robust validation cost measure, J (RMSE, AUC, Logloss). For example, n-repeated cross validation (CV) RMSE for regression or n-repeated CV Logloss or AUC for classification; "testing_cost": testing cost obtained with a testing dataset not included in the validation process. This value is only for checking the generalization capability of the model. NA value can be supplied if there is not a testing data set; "model_complexity": model...
complexity. Can be the number of features, number of support vectors in SVM, sum of power of weights in ANNs, depth in Trees, generalised degrees of freedom (GDF), or another complexity metrics or combination of them.

Note: the chromosome is a concatenated real vector with the model parameters (parameters-chromosome) and the binary selection of the input features (features-chromosome). For example, a chromosome defined as c(10, 0.01, 0.1,1,0,1,0,0) could corresponds to a SVR model parameters C=10 & gamma=0.01, and a selection of three input features (second, third and fifth) from a dataset of 7 features (0110100).

... additional arguments to be passed to the fitness function. This allows to write fitness functions that keep some variables fixed during the search.

min_param a vector of length equal to the model parameters providing the minimum of the search space.

max_param a vector of length equal to the model parameters providing the maximum of the search space.

nFeatures a value specifying the number of maximum input features.

names_param a vector with the name of the model parameters.

names_features a vector with the name of the input features.

iter_ini initial iteration. If 'iter_ini' is > 0, 'object' must be provided to continue the optimization process.

object object of 'ga_parsimony' class to use if 'iter_ini'>0

type_ini_pop method to create the first population with 'parsimony_population' function. This function is called when iter_ini==0 and 'suggestions' are not provided. Methods='randomLHS','geneticLHS','improvedLHS','maximinLHS','optimumLHS','random'. First 5 methods correspond with several latin hypercube sampling.

popSize the population size.

pcrossover the probability of crossover between pairs of chromosomes. Typically this is a large value and by default is set to 0.8.

maxiter the maximum number of iterations to run before the GA process is halted.

feat_thres proportion of selected features in the initial population. It is recommended a high percentage of selected features for the first generations. By default is set to 0.90.

rerank_error when a value is provided, a second reranking process according to the model complexities is called by parsimony_rerank function. Its primary objective is to select individuals with high validation cost while maintaining the robustness of a parsimonious model. This function switches the position of two models if the first one is more complex than the latter and no significant difference is found between their fitness values in terms of cost. Therefore, if the absolute difference between the validation costs are lower than 'rerank_error' they are considered similar. Default value=0.01

iter_start_rerank iteration when ReRanking process is actived. Default=0. Sometimes is useful not to use ReRanking process in the first generations.
pmutation the probability of mutation in a parent chromosome. Usually mutation occurs with a small probability. By default is set to 0.10.

feat_mut_thres probability of the muted 'features-chromosome' to be one. Default value is set to 0.10.

not_muted number of the best elitists that are not muted in each generation. Default value is set to 3.

elitism the number of best individuals to survive at each generation. By default the top 20% individuals will survive at each iteration.

population an R function for randomly generating an initial population. See parsimony_population for available functions.

selection an R function performing selection, i.e. a function which generates a new population of individuals from the current population probabilistically according to individual fitness. See parsimony_nlrSelection for available functions.

crossover an R function performing crossover, i.e. a function which forms offsprings by combining part of the genetic information from their parents. See parsimony_crossover for available functions.

mutation an R function performing mutation, i.e. a function which randomly alters the values of some genes in a parent chromosome. See parsimony_mutation for available functions.

keep_history If it is TRUE keeps in the list object@history each generation. This parameter must set TRUE in order to use 'plot' method or 'parsimony_importance' function.

early_stop the number of consecutive generations without any improvement in the best fitness value before the GA is stopped.

maxFitness the upper bound on the fitness function after that the GA search is interrupted. Default value is set to +Inf

suggestions a matrix of solutions strings to be included in the initial population. If provided the number of columns must match (object@nParams+object@nFeatures)

parallel a logical argument specifying if parallel computing should be used (TRUE) or not (FALSE, default) for evaluating the fitness function. This argument could also be used to specify the number of cores to employ; by default, this is taken from detectCores. Finally, the functionality of parallelization depends on system OS: on Windows only 'snow' type functionality is available, while on Unix/Linux/Mac OSX both 'snow' and 'multicore' (default) functionalities are available.

monitor a logical or an R function which takes as input the current state of the ga_parsimony-class object and show the evolution of the search. By default, for interactive sessions, the function parsimony_monitor depending on whether or not is an RStudio session, prints the average and best fitness values at each iteration. If set to plot these information are plotted on a graphical device. Other functions can be written by the user and supplied as argument. In non interactive sessions, by default monitor = FALSE so any output is suppressed.

seed_ini an integer value containing the random number generator state. This argument can be used to replicate the results of a GA search. Note that if parallel computing is required, the doRNG package must be installed.

verbose if it is TRUE shows additional information for debugging.
Details

**GAparsimony** package is a new GA wrapper automatic procedure that efficiently generated prediction models with reduced complexity and adequate generalization capacity. The *ga_parsimony* function is primarily based on combining feature selection and model parameter tuning with a second novel GA selection process (*parsimony_rerank* function), in order to achieve better overall parsimonious models. Unlike other GA methodologies that use a penalty parameter for combining loss and complexity measures into a unique fitness function, the main contribution of this package is that *ga_parsimony* selects the best models by considering cost and complexity separately. For this purpose, the ReRank algorithm rearranges individuals by their complexity when there is not a significant difference between their costs. Thus, less complex models with similar accuracy are promoted. Furthermore, because the penalty parameter is unnecessary, there is no consequent uncertainty associated with assigning a correct value beforehand. As a result, with GA-PARSIMONY, an automatic method for obtaining parsimonious models is finally made possible.

Value

Returns an object of class *ga_parsimony-class*. See *ga_parsimony-class* for a description of available slots information.

Author(s)

Francisco Javier Martinez de Pison. <fjmartin@unirioja.es>, EDMANS Group. [http://www.mineriadatos.com](http://www.mineriadatos.com)

References


See Also

*ga_parsimony-class*, *summary.ga_parsimony*, *plot.ga_parsimony*, *parsimony_Population*, *parsimony_Selection*, *parsimony_Crossover*, *parsimony_Mutation*, *parsimony_importance*, *parsimony_rerank*. 

---

**ga_parsimony**
Examples

# This a toy example that shows how to search, for the *iris* database,
# a parsimony classification NNET model with 'GAparsimony'
# and 'caret' packages. Validation errors and iterations have been
# reduced to speedup the process

# Training and testing Datasets
library(caret)

data(iris)
# Z-score of input features
iris_esc <- data.frame(scale(iris[,1:4]),Species=iris[,5])

# Define an 70%/30% train_val/test split of the dataset
set.seed(1234)
inTraining <- createDataPartition(iris_esc$Species, p=.70, list=FALSE)
data_train <- iris_esc[inTraining,]
data_test <- iris_esc[-inTraining,]

# Function to evaluate each SVM individual
# ------------------------------------------
fitness_SVM <- function(chromosome, ...)
{
  # First two values in chromosome are 'C' & 'sigma' of 'svmRadial' method
tuneGrid <- data.frame(C=chromosome[1],sigma=chromosome[2])

  # Next values of chromosome are the selected features (TRUE if > 0.50)
  selec_feat <- chromosome[3:length(chromosome)]>0.50

  # Return -Inf if there is not selected features
  if (sum(selec_feat)<1) return(c(kappa_val=-Inf,kappa_test=-Inf,complexity=Inf))

  # Extract features from the original DB plus response (last column)
data_train_model <- data_train[,c(selec_feat,TRUE)]
data_test_model <- data_test[,c(selec_feat,TRUE)]

  # Validate each individual with only a 2-CV
  # Yo obtain a robust validation measure
  # use 'repeatedcv' with more folds and times
  # (see 2nd and 3rd examples...)
  train_control <- trainControl(method = "cv",number = 2)

  # train the model
  set.seed(1234)
  model <- train(Species ~ ., data=data_train_model,
                trControl=train_control,
                ...)
ga_parsimony

method = "svmRadial", metric = "Kappa",
    tuneGrid = tuneGrid, verbose = FALSE)

# Extract validation and test accuracy
accuracy_val <- model$results$Accuracy
accuracy_test <- postResample(pred = predict(model, data_test_model),
    obs = data_test_model[, ncol(data_test_model)])[2]

# Obtain Complexity = Num_Features * 1E6 + Number of support vectors
complexity <- sum(selec_feat) * 1E6 + model$finalModel$nSV

# Return (validation accuracy, testing accuracy, model_complexity)
vect_errors <- c(accuracy_val = accuracy_val,
    accuracy_test = accuracy_test, complexity = complexity)
return(vect_errors)
}

# Search the best parsimonious model with GA-PARSIMONY by using Feature Selection,
# Parameter Tuning and Parsimonious Model Selection
#
#library(GA Parsimony)

# Ranges of size and decay
min_param <- c(0.0001, 0.00001)
max_param <- c(0.9999, 0.99999)
names_param <- c("C", "sigma")

# ga_parsimony can be executed with a different set of 'rerank_error' values
rerank_error <- 0.01

GA Parsimony model <- ga_parsimony(fitness = fitness_SVM,
    min_param = min_param,
    max_param = max_param,
    names_param = names_param,
    keep_history = TRUE,
    rerank_error = rerank_error,
    popsize = 10, # Change to 40
    maxiter = 1, # Change to 100
    early_stop = 10,
    feat_thres = 0.90, # Perc selec features in first iter
    feat_mut_thres = 0.10, # Prob. feature to be 1 in mutation
    not_muted = 1,
    parallel = FALSE, # speedup with n cores or TRUE = all
    seed_ini = 1234)

print(paste0("Best Parsimonious SVM with C = ",
    GA Parsimony model@bestsolution["C"],
    " sigma = ",}
```r
GAparsimony_model@bestsolution['sigma'],
"->",
"AccuracyVal",
round(GAparsimony_model@bestsolution['fitnessVal'],6),
"AccuracyTest",
round(GAparsimony_model@bestsolution['fitnessTst'],6),
"Num Features",
round(GAparsimony_model@bestsolution['complexity']/1E6,0),
"Complexity",
round(GAparsimony_model@bestsolution['complexity'],2))

print(summary(GAparsimony_model))

print(parsimony_importance(GAparsimony_model))

#############################################################
### Example 2: Classification ###
#############################################################

# This example shows how to search, for the *Sonar* database, a parsimony classification SVM model with 'GAparsimony' and 'caret' packages.

# Training and testing Datasets
library(caret)
library(GAparsimony)
library(mlbench)
data(Sonar)

set.seed(1234)
inTraining <- createDataPartition(Sonar$Class, p=.80, list=FALSE)
data_train <- Sonar[inTraining,]
data_test <- Sonar[-inTraining,]

# Function to evaluate each SVM individual
# ------------------------------
fitness_SVM <- function(chromosome, ...)
{
  # First two values in chromosome are 'C' & 'sigma' of 'svmRadial' method
tuneGrid <- data.frame(C=chromosome[1],sigma=chromosome[2])

  # Next values of chromosome are the selected features (TRUE if > 0.50)
selec_feat <- chromosome[3:length(chromosome)]>0.50

  # Return -Inf if there is not selected features
if (sum(selec_feat)<1) return(c(kappa_val=-Inf,kappa_test=-Inf,complexity=Inf))

  # Extract features from the original DB plus response (last column)
data_train_model <- data_train[,c(selec_feat,TRUE)]
data_test_model <- data_test[,c(selec_feat,TRUE)]

  # How to validate each individual
```
#'repeats' could be increased to obtain a more robust validation metric. Also, 
#'number' of folds could be adjusted to improve the measure.
train_control <- trainControl(method = "repeatedcv", number = 10, repeats = 10)

# train the model
set.seed(1234)
model <- train(Class ~ ., data=data_train_model, trControl=train_control, 
method="svmRadial", metric="Kappa", 
tuneGrid=tuneGrid, verbose=FALSE)

# Extract kappa statistics (repeated k-fold CV and testing kappa)
kappa_val <- model$results$Kappa
kappa_test <- postResample(pred=predict(model, data_test_model), 
obs=data_test_model[,ncol(data_test_model)])[2]

# Obtain Complexity = Num_Features*1E6+Number of support vectors 
complexity <- sum(selec_feat)*1E6+model$finalModel$nSV

# Return(-validation error, -testing error, model_complexity) 
vect_errors <- c(kappa_val=kappa_val,kappa_test=kappa_test,complexity=complexity) 
return(vect_errors)

# Search the best parsimonious model with GA-PARSIMONY by using feature selection, 
# Parameter Tuning and Parsimonious Model Selection
library(ga_parsimony)

# Ranges of size and decay 
min_param <- c(0.0001, 0.0001)
max_param <- c(99.9999, 0.9999)
names_param <- c("C","sigma")

# ga_parsimony can be executed with a different set of 'rerank_error' values 
rerank_error <- 0.001

# 40 individuals per population, 100 max generations with an early stopping 
# of 10 generations (CAUTION! 7.34 minutes with 8 cores)!!!!
GAparsimony_model <- ga_parsimony(fitness=fitness_SVM, 
min_param=min_param, 
max_param=max_param, 
names_param=names_param, 
nFeatures=ncol(data_train)-1, 
names_features=colnames(data_train)[-ncol(data_train)], 
keep_history = TRUE, 
rerank_error = rerank_error, 
popSize = 40, 
maxiter = 100, 
early_stop=10, 
feat_thres=0.90,# Perc selec features in first iter 
feat_mut_thres=0.10,# Prob. feature to be 1 in mutation 
parallel = TRUE, seed_ini = 1234)
print(paste0("Best Parsimonious SVM with C = ",
GAparsimony_model@bestsolution['C'],
" sigma = ",
GAparsimony_model@bestsolution['sigma'],
" -> ",
" KappaVal = ",
round(GAparsimony_model@bestsolution['fitnessVal'],6),
" KappaTst = ",
round(GAparsimony_model@bestsolution['fitnessTst'],6),
" Num Features = ",
round(GAparsimony_model@bestsolution['complexity']/1E6,0),
" Complexity = ",
round(GAparsimony_model@bestsolution['complexity'],2))

print(summary(GAparsimony_model))

# Plot GA evolution ('keep_history' must be TRUE)
elitists <- plot(GAparsimony_model)

# Percentage of appearance of each feature in elitists
print(parsimony_importance(GAparsimony_model))

#########################################################
### Example 3: Regression ###
#########################################################

# This example shows how to search, for the *Boston* database, a parsimony regressor ANN
# model with 'GAparsimony' and 'caret' packages.

# Load Boston database and scale it
library(MASS)
data(Boston)
Boston_scaled <- data.frame(scale(Boston))

# Define an 80%/20% train/test split of the dataset
set.seed(1234)
trainIndex <- createDataPartition(Boston[, "medv"], p=0.80, list=FALSE)
data_train <- Boston_scaled[trainIndex,]
data_test <- Boston_scaled[-trainIndex,]
# Restore 'Response' to original values
data_train[, ncol(data_train)] <- Boston$medv[trainIndex]
data_test[, ncol(data_test)] <- Boston$medv[-trainIndex]
print(dim(data_train))
print(dim(data_test))

# Function to evaluate each ANN individual
# --------------------------------------------
fitness_NNET <- function(chromosome, ...)
\[
\begin{align*}
\text{# First two values in chromosome are 'size' & 'decay' of 'nnet' method} \\
tuneGrid <- \text{data.frame(size=round(chromosome[1]), decay=chromosome[2])}
\end{align*}
\]

\[
\begin{align*}
\text{# Next values of chromosome are the selected features (TRUE if > 0.50)} \\
\text{selec_feat <- chromosome[3:length(chromosome)]>0.50} \\
\text{if (sum(selec_feat)<1) return(c(rmse_val=-Inf, rmse_test=-Inf, complexity=Inf))}
\end{align*}
\]

\[
\begin{align*}
\text{# Extract features from the original DB plus response (last column)} \\
data_train_model <- \text{data_train[,c(selec_feat,TRUE)]} \\
data_test_model <- \text{data_test[,c(selec_feat,TRUE)]}
\end{align*}
\]

\[
\begin{align*}
\text{# How to validate each individual} \\
\text{# 'repeats' could be increased to obtain a more robust validation metric. Also,} \\
\text{# 'number' of folds could be adjusted to improve the measure.} \\
\text{train_control <- trainControl(method = "repeatedcv", number = 10, repeats = 5)} \\
\end{align*}
\]

\[
\begin{align*}
\text{# train the model} \\
\text{set.seed(1234)} \\
\text{model <- train(medv ~ ., data=data_train_model, trControl=train_control,} \\
\text{ method="nnet", tuneGrid=tuneGrid, trace=FALSE, linout = 1)}
\end{align*}
\]

\[
\begin{align*}
\text{# Extract errors} \\
rmse_val <- model$results$RMSE \\
rmse_test <- sqrt(mean((unlist(predict(model, newdata = data_test_model)) -} \\
\text{data_test_model$medv)^2)))} \\
\text{# Obtain Complexity = Num_Features*1E6+sum(neural_weights^2)} \\
\text{complexity <- sum(selec_feat)*1E6+sum(model$finalModel$wts*model$finalModel$wts)}
\end{align*}
\]

\[
\begin{align*}
\text{# Return(-validation error, -testing error, model_complexity)} \\
\text{# errors are negative because GA-PARSIMONY tries to maximize values} \\
\text{vect_errors <- c(rmse_val=-rmse_val, rmse_test=-rmse_test, complexity=complexity)} \\
\text{return(vect_errors)}
\end{align*}
\]

---

library(GAparsonomy)

\[
\begin{align*}
\text{# Ranges of size and decay} \\
\text{min_param <- c(1, 0.0001)} \\
\text{max_param <- c(25 , 0.9999)} \\
\text{names_param <- c("size","decay")}
\end{align*}
\]

\[
\begin{align*}
\text{# ga_parsimony can be executed with a different set of 'rerank_error' values} \\
\text{rerank_error <- 0.01}
\end{align*}
\]

\[
\begin{align*}
\text{# 40 individuals per population, 100 max generations with an early stopping} \\
\text{# of 10 generations (CAUTION! 33.89 minutes with 8 cores)!!!!!!} \\
\text{GAparsonomy_model <- ga_parsimony(Fitness=fittest_NNET,} \\
\text{                min_param=min_param,}
\end{align*}
\]
ga_parsimony-class

Class "ga_parsimony"

Description
An S4 class for searching parsimonious models by feature selection and parameter tuning with genetic algorithms.

Objects from the Class
Objects can be created by calls to the ga_parsimony function.

Slots
- call an object of class "call" representing the matched call;
- min_param a vector of length equal to the model parameters providing the minimum of the search space;
max_param a vector of length equal to the model parameters providing the maximum of the search space;

nParams a value specifying the number of model parameter to be tuned;

feat_thres proportion of selected features in the initial population. It is recommended a high percentage of selected features for the first generations;

feat_mut_thres threshold to consider a random number between 0 and 1 is considered one if a value of the parameters-chromosome is muted. Default value is set to 0.5;

not_muted number of the best elitists that are not muted. Default value is set to 3;

rerank_error when a value distinct to zero is provided a second reranking process according to the model complexities is called by 'parsimonyReRank' function. Its primary objective is to select individuals with high validation cost while maintaining the robustness of a parsimonious model. This function switches the position of two models if the first one is more complex than the latter and no significant difference is found between their fitness values in terms of cost. Therefore, if the absolute difference between the validation costs are lower than 'rerank_error' they are considered similar. Default value=0.01;

nFeatures a value specifying the number of maximum input features;

names_param a vector with the name of the model parameters;

names_features a vector with the name of the input features;

popSize the population size;

iter the actual (or final) iteration of GA search;

iter_start_rerank iteration when ReRanking process is actived. Default=0. Sometimes is useful not to use ReRanking process in the first generations;

early_stop the number of consecutive generations without any improvement in the best fitness value before the GA is stopped;

maxiter the maximum number of iterations to run before the GA search is halted;

minutes_gen elapsed time of this generation (in minutes);

minutes_total total elapsed time (in minutes);

suggestions a matrix of user provided solutions and included in the initial population;

population the current (or final) population;

elitism the number of best fitness individuals to survive at each generation;

crossover the crossover probability;

mutation the mutation probability;

fitnessval the values of validation cost for the current (or final) population;

fitnessstst the values of testing cost for the current (or final) population;

complexity the values of model complexities for the current (or final) population;

summary a matrix of summary statistics for fitness values at each iteration (along the rows);

bestSollist a list with the best solution of all iterations;

bestfitnessval the validation cost of the best solution at the last iteration;

bestfitnessTst the testing cost of the best solution at the last iteration;

bestcomplexity the model complexity of the best solution at the last iteration;

bestsolution the best solution at the last iteration;

history a list with the population of all iterations;
numericOrNA-class

Author(s)
Francisco Javier Martinez-de-Pison. <fjmartin@unirioja.es>. EDMANS Group. http://www.mineriadatos.com

See Also
For examples of usage see ga_parsimony.

---

numericOrNA-class

Virtual Class "numericOrNA" - Simple Class for subassignment Values

Description
The class "numericOrNA" is a simple class union (setClassUnion) of "numeric" and "logical".

Objects from the Class
Since it is a virtual Class, no objects may be created from it.

Examples
showClass("numericOrNA")

---

parsimony_crossover

Crossover operators in GA-PARSIMONY

Description
Functions implementing particular crossover genetic operator for GA-PARSIMONY. Method uses for model parameters Heuristic Blending and random swapping for binary selected features.

Usage
parsimony_crossover(object, parents, alpha=0.1, perc_to_swap=0.5, ...)

Arguments
- object: An object of class "ga_parsimony", usually resulting from a call to function ga_parsimony.
- parents: A two-rows matrix of values indexing the parents from the current population.
- alpha: A tuning parameter for the Heuristic Blending outer bounds [Michalewicz, 1991]. Typical and default value is 0.1.
- perc_to_swap: Percentage of features for swapping in the crossovering process.
- ... Further arguments passed to or from other methods.
parsimony_importance

Value

Return a list with two elements:

- **children**: Matrix of dimension 2 times the number of decision variables containing the generated offspings.
- **fitnessval**: Vector of length 2 containing the fitness validation values for the offsprings. A value `NA` is returned if an offspring is different (which is usually the case) from the two parents.
- **fitnesstst**: Vector of length 2 containing the fitness with the test database (if it was supplied), for the offsprings. A value `NA` is returned if an offspring is different (which is usually the case) from the two parents.
- **complexity**: Vector of length 2 containing the model complexity for the offsprings. A value `NA` is returned if an offspring is different (which is usually the case) from the two parents.

Author(s)

Francisco Javier Martinez de Pison. `<fjmartin@unirioja.es>`. EDMANS Group. [http://www.mineriadatos.com](http://www.mineriadatos.com)

See Also

`ga_parsimony`

parsimony_importance  Percentage of appearance of each feature in elitist population

Description

Shows the percentage of appearance of each feature in the whole GA-PARSIMONY process but only for the elitist-population.

Usage

`parsimony_importance(object, verbose=FALSE, ...)`

Arguments

- **object**: An object of class "ga_parsimony" resulting from a call to function `ga_parsimony` with `keep_history` parameter set to TRUE.
- **verbose**: If it is TRUE shows additional information.
- **...**: Further arguments passed to or from other methods.

Details

parsimony_importance extracts elitist population from all generations. Obtains the percentage of appearance of each feature in the all GA process. Return the features higher-ordered.
Value

Return a vector with the higher-ordered percentage of appearance of each feature in the elitist-population and in the whole GA process.

Author(s)

Francisco Javier Martinez de Pison. <fjmartin@unirioja.es>. EDMANS Group. http://www.mineriadatos.com

See Also

gparsimony

Description

Functions to print summary statistics of fitness values at each iteration of a GA search.

Usage

parsimony_monitor(object, digits = getOption("digits"), ...)

Arguments

object An object of class "ga_parsimony", usually resulting from a call to function gparsimony.
digits minimal number of significant digits.
... Further arguments passed to or from other methods.

Value

These functions print a summary of current GA-PARSIMONY step on the console.

Author(s)

Francisco Javier Martinez de Pison. <fjmartin@unirioja.es>. EDMANS Group. http://www.mineriadatos.com
parsimony_Mutation  

Mutation operators in GA-PARSIMONY

Description

Functions implementing mutation genetic operator for GA-PARSIMONY. Method mutes a `object@pmutation` percentage of them. If the value corresponds to a model parameter, algorithm uses uniform random mutation. For binary select features, method sets to one if the random value between [0,1] is lower or equal to `object@feat_mut_thres`.

Usage

```r
parsimony_mutation(object, ...)```

Arguments

- `object`: An object of class "ga_parsimony", usually resulting from a call to function `ga_parsimony`.
- `...`: Further arguments passed to or from other methods.

Value

Return `object` with the population muted.

Author(s)

Francisco Javier Martinez de Pison. <fjmartin@unirioja.es>. EDMANS Group. [http://www.mineriadatos.com](http://www.mineriadatos.com)

See Also

`ga_parsimony`

parsimony_Population  

Population initialization in GA-PARSIMONY with a combined chromosome of model parameters and selected features

Description

Functions for creating an initial population to be used in the GA-PARSIMONY process.

Usage

```r
parsimony_population(object, type_ini_pop="randomLHS", ...)```
Arguments

- **object**: An object of class "ga_parsimony", usually resulting from a call to function `ga_parsimony`.
- **type_ini_pop**: How to create the initial population. 'random' option initializes a random population between the predefined ranges. Values 'randomLHS', 'geneticLHS', 'improvedLHS', 'maximinLHS' & 'optimumLHS' correspond with several methods of the Latin Hypercube Sampling (see 'lhs' package for more details).
- **...**: Further arguments passed to or from other methods.

Details

`parsimony_population` generates a random population of `object@popsize` individuals. For each individual a random chromosome is generated with `object@nparams` real values in the range `[object@min_param, object@max_param]` plus `object@nfeatures` random binary values for feature selection. 'random' or Latin Hypercube Sampling can be used to create an efficient spread initial population.

Value

Return a matrix of dimension `object@popsize` rows and `object@nparams+object@nfeatures` columns.

Author(s)

Francisco Javier Martinez de Pison. <fjmartin@unirioja.es>. EDMANS Group. [http://www.mineriadatos.com](http://www.mineriadatos.com)

See Also

`ga_parsimony`

---

`parsimony_rerank` *Function for reranking by complexity in parsimonious model selection process*

Description

Promotes models with similar fitness but lower complexity to top positions.

Usage

`parsimony_rerank(object, verbose=FALSE, ...)`
parsimony_rerank

Arguments

object  An object of class "ga_parsimony" resulting from a call to function `ga_parsimony` with `keep_history` parameter set to TRUE.

verbose  If it is TRUE shows additional information.

...  Further arguments passed to or from other methods.

Details

This method corresponds with the second step of parsimonious model selection (PMS) procedure. PMS works in the following way: in each GA generation, best solutions are first sorted by their cost, J. Then, in a second step, individuals with less complexity are moved to the top positions when the absolute difference of their J is lower than a `object@rerank_error` threshold value. Therefore, the selection of less complex solutions among those with similar accuracy promotes the evolution of robust solutions with better generalization capabilities.

Value

Return a vector with the new position of the individuals.

Author(s)

Francisco Javier Martinez de Pison. <fjmartin@unirioja.es>. EDMANS Group. http://www.mineriadatos.com

See Also

ga_parsimony

Examples

```r
library(GAparsimony)
object <- new("ga_parsimony",
                   rerank_error=0.2,
                   popSize = 4,
                   fitnessval = c(2.0, 1.9, 1.1, 1.0),
                   complexity=c(2,1,2,1))

pop_ini <- data.frame(fitnessval=object@fitnessval,
                       complexity=object@complexity)
print("INITIAL POPULATION:")
print(pop_ini)

print("POPULATION ORDERED BY COMPLEXITY")
print(paste0("WHEN abs(diff(fitnessval)) \< ",
            object@rerank_error,"\)\)\)
nop_ini[parsimony_rerank(object),]
```
parsimony_Selection  
Selection operators in GA-PARSIMONY

Description

Functions implementing selection genetic operator in GA-PARSIMONY after parsimony_rerank process. Linear-rank or Nonlinear-rank selection (Michalewicz (1996)).

Usage

parsimony_lrSelection(object, r = 2/(object@popSize*(object@popSize-1)),
q = 2/object@popSize, ...)
parsimony_nlrSelection(object, q = 0.25, ...)

Arguments

object  An object of class "ga_parsimony", usually resulting from a call to function ga_parsimony.
r  A tuning parameter for the specific selection operator.
q  A tuning parameter for the specific selection operator.
...  Further arguments passed to or from other methods.

Value

Return a list with four elements:

population  a matrix of dimension object@popSize times the number of decision variables containing the selected individuals or strings;
fitnessval  a vector of length object@popSize containing the fitness validation values for the selected individuals;
fitnesstst  a vector of length object@popSize containing the fitness with the test database (if it was supplied), for the selected individuals;
complexity  a vector of length object@popSize containing the model complexity for the selected individuals.

Author(s)

Francisco Javier Martinez de Pison. <fjmartin@unirioja.es>. EDMANS Group. http://www.mineriadatos.com

See Also

ga_parsimony
The plot method for `ga_parsimony-class` objects gives a evolution plot of the validation and testing errors, and the number of model features selected of elitists.

## Usage

```r
## S4 method for signature 'ga_parsimony'
plot(x, general_cex = 0.7, min_ylim=NULL, max_ylim=NULL,
     min_iter=NULL, max_iter=NULL,
     main_label="Boxplot cost evolution",
     iter_auto_ylim=3, steps=5, pos_cost_num=-3.1,
     pos_feat_num=-1.7, digits_plot=4, width_plot=12,
     height_plot=6, window=TRUE, ...)
```

## Arguments

- `x`: An object of class "ga_parsimony".
- `general_cex`: Main text scale.
- `min_ylim`: Min limit on the y-axis.
- `max_ylim`: Max limit on the y-axis.
- `min_iter`: Min GA iteration to visualize.
- `max_iter`: Max GA iteration to visualize.
- `main_label`: Main plot title.
- `iter_auto_ylim`: If it is not NULL, GA iteration to choose the min limit of y-axis.
- `steps`: Number of divisions in y-axis.
- `pos_cost_num`: Relative position of numbers in cost axis.
- `pos_feat_num`: Relative position of numbers in feature axis.
- `digits_plot`: Number of digits to visualize.
- `width_plot`: Figure width in inches.
- `height_plot`: Figure height in inches.
- `window`: If TRUE shows a new window.
- `...`: Further arguments, currently not used.

## Details

Plot method shows the evolution of validation and testing errors, and the number of model features selected of elitists. White and grey box-plots represent validation and testing errors of elitists evolution, respectively. Continuous and dashed-dotted lines show the validation and testing error of the best individual for each generation, respectively. Finally, the shaded area delimits the maximum and minimum number of features, and the dashed line, the number fo features of the best individual.
Value

The method invisibly return a list with the elistists validation error, testing error and model complexity in the whole GA process.

Author(s)

Francisco Javier Martinez de Pison. <fjmartin@unirioja.es>. EDMANS Group. http://www.mineriadatos.com

See Also

ga_parsimony, ga_parsimony-class.

Description

Summary method for class ga_parsimony-class.

Usage

```r
## S4 method for signature 'ga_parsimony'
summary(object, ...)
## S3 method for class 'summary.ga_parsimony'
print(x, digits = getOption("digits"), ...)
```

Arguments

- `object`: an object of class `ga_parsimony-class`.
- `x`: an object of class `summary.ga_parsimony`.
- `digits`: number of significant digits.
- `...`: further arguments passed to or from other methods.

Value

The summary function returns an object of class `ga_parsimony-class` which can be printed by the corresponding print method. The function also returns invisibly a list with the information from the genetic algorithm search.

Author(s)

Francisco Javier Martinez de Pison. <fjmartin@unirioja.es>. EDMANS Group. http://www.mineriadatos.com
See Also

ga_parsimony
Index

*Topic **classes**
  * ga_parsimony-class, 13
  * numericOrNA-class, 15
*Topic **hplot**
  * plot.ga_parsimony-method, 22
*Topic **methods**
  * plot.ga_parsimony-method, 22
*Topic **optimize**
  * ga_parsimony, 3
  * ga_parsimony-class, 13
  * summary.ga_parsimony-method, 23
*Topic **package**
  * GAParsimony-package, 2

detectCores, 5

**ga_parsimony, 3, 13, 15–21, 23, 24**

**ga_parsimony-class, 6, 13**

**GAparsimony (GAparsimony-package), 2**

**GAparsimony-package, 2**

**numericOrNA (ga_parsimony), 3**

**numericOrNA-class, 15**

**parsimony_Crossover, 6**

**parsimony_Crossover**
  *(parsimony_crossover), 15*

**parsimony_crossover, 5, 15**

**parsimony_importance, 6, 16**

**parsimony_lrSelection**
  *(parsimony_Selection), 21*

**parsimony_monitor, 5, 17**

**parsimony_Mutation, 6, 18**

**parsimony_mutation, 5**

**parsimony_mutation**
  *(parsimony_Mutation), 18*

**parsimony_nlrSelection, 5**

**parsimony_nlrSelection**
  *(parsimony_Selection), 21*

**parsimony_Population, 6, 18**

**parsimony_population, 5**

**parsimony_population**
  *(parsimony_Population), 18*

**parsimony_rerank, 4, 6, 19, 21**

**parsimony_Selection, 6, 21**

**plot,ga_parsimony-method**
  *(plot.ga_parsimony-method), 22*

**plot.ga_parsimony, 6**

**plot.ga_parsimony**
  *(plot.ga_parsimony-method), 22*

**plot.ga_parsimony-method**
  *(ga_parsimony), 3*

**print,ga_parsimony-method**
  *(ga_parsimony), 3*

**print.summary.ga_parsimony**
  *(summary.ga_parsimony-method), 23*

**setClassUnion, 15**

**show,ga_parsimony-method**
  *(ga_parsimony), 3*

**summary,ga_parsimony-method**
  *(summary.ga_parsimony-method), 23*

**summary.ga_parsimony, 6**

**summary.ga_parsimony**
  *(summary.ga_parsimony-method), 23*

**summary.ga_parsimony-method, 23**