Package ‘gaselect’
February 6, 2020

Type Package
Title Genetic Algorithm (GA) for Variable Selection from High-Dimensional Data
Version 1.0.9
Date 2020-02-04
Author David Kepplinger
Maintainer David Kepplinger <david.kepplinger@gmail.com>
Description Provides a genetic algorithm for finding variable subsets in high dimensional data with high prediction performance. The genetic algorithm can use ordinary least squares (OLS) regression models or partial least squares (PLS) regression models to evaluate the prediction power of variable subsets. By supporting different cross-validation schemes, the user can fine-tune the tradeoff between speed and quality of the solution.

URL https://github.com/dakep/gaselect
BugReports https://github.com/dakep/gaselect/issues
License GPL (>= 2)
NeedsCompilation yes
Encoding UTF-8
Biarch true
SystemRequirements C++11
Depends R (>= 3.0.2), methods (>= 2.10.0)
Imports Rcpp (>= 0.10.5)
LinkingTo Rcpp (>= 0.10.5), RcppArmadillo (>= 0.4.000)
Collate 'Evaluator.R' 'GenAlgControl.R' 'formatSegmentation.R'
'evaluate.R' 'fitness.R' 'genAlg.R' 'getEvalFun.R' 'subsets.R'
'toCControlList.R' 'validData.R'
Suggests chemometrics
RoxygenNote 7.0.2
Repository CRAN
Date/Publication 2020-02-06 01:00:10 UTC
**R topics documented:**

- evaluate .................................................. 2
- evaluatorFit ............................................. 3
- evaluatorLM ............................................... 4
- evaluatorPLS ............................................. 5
- evaluatorUserFunction ............................. 7
- fitness .................................................. 8
- fitnessEvolution ..................................... 9
- formatSegmentation ............................. 10
- genAlg ................................................... 11
- GenAlg-class .......................................... 12
- genAlgControl .......................................... 13
- GenAlgControl-class ............................. 15
- GenAlgEvaluator-class ......................... 16
- GenAlgFitEvaluator-class ................. 16
- GenAlgLMEvaluator-class ................. 16
- GenAlgPLSEvaluator-class ..................... 17
- GenAlgUserEvaluator-class ..................... 17
- getEvalFun .............................................. 18
- subsets ............................................... 18
- toCControlList ...................................... 19
- trueFitnessVal ....................................... 20
- validData .............................................. 21

**Index** 22

---

**evaluate**

*Evaluate the fitness of variable subsets*

**Description**

Evaluate the given variable subsets with the given Evaluator

**Usage**

```r
evaluate(object, X, y, subsets, seed, verbosity)
```

---

## S4 method for signature

```r
# 'GenAlgEvaluator,matrix,numeric,matrix,integer,integer'
evaluate(object, X, y, subsets, seed, verbosity)
```

## S4 method for signature

```r
# 'GenAlgEvaluator,matrix,numeric,logical,integer,integer'
evaluate(object, X, y, subsets, seed, verbosity)
```

## S4 method for signature 'GenAlgEvaluator,matrix,numeric,ANY,missing,integer'

```r
evaluate(object, X, y, subsets, seed, verbosity)
```
## S4 method for signature 'GenAlgEvaluator,matrix,numeric,ANY,integer,missing'
evaluate(object, X, y, subsets, seed, verbosity)

## S4 method for signature 'GenAlgEvaluator,matrix,numeric,ANY,missing,missing'
evaluate(object, X, y, subsets, seed, verbosity)

### Arguments

- **object**
  - The GenAlgEvaluator object that is used to evaluate the variables
- **X**
  - The data matrix used to fit the model
- **y**
  - The response vector
- **subsets**
  - The logical matrix where a column stands for one subset to evaluate
- **seed**
  - The value to seed the random number generator before evaluating
- **verbosity**
  - A value between 0 (no output at all) and 5 (maximum verbosity)

### Description

Creates the object that controls the evaluation step in the genetic algorithm

### Usage

```r
evaluatorFit(
  numSegments = 7L,
  statistic = c("BIC", "AIC", "adjusted.r.squared", "r.squared"),
  numThreads = NULL,
  maxNComp = NULL,
  sdfact = 1
)
```

### Arguments

- **numSegments**
  - The number of CV segments used to estimate the optimal number of PLS components (between 2 and $2^{16}$).
- **statistic**
  - The statistic used to evaluate the fitness (BIC, AIC, adjusted $R^2$, or $R^2$).
- **numThreads**
  - The maximum number of threads the algorithm is allowed to spawn (a value less than 1 or NULL means no threads).
- **maxNComp**
  - The maximum number of components the PLS models should consider (if not specified, the number of components is not constrained).
- **sdfact**
  - The factor to scale the standard deviation of the MSEP values when selecting the optimal number of components. For the "one standard error rule", `sdfact` is 1.
Details
The fitness of a variable subset is assessed by how well a PLS model fits the data. To estimate the
optimal number of components for the PLS model, cross-validation is used.

Value
Returns an S4 object of type GenAlgFitEvaluator to be used as argument to a call of genAlg.

See Also
Other GenAlg Evaluators: evaluatorLM(), evaluatorPLS(), evaluatorUserFunction()

Examples
ctrl <- genAlgControl(populationSize = 200, numGenerations = 30, minVariables = 5,
maxVariables = 12, verbosity = 1)
evaluator <- evaluatorFit(statistic = "BIC", numThreads = 1)

# Generate demo-data
set.seed(12345)
X <- matrix(rnorm(10000, sd = 1:5), ncol = 50, byrow = TRUE)
y <- drop(-1.2 + rowSums(X[, seq(1, 43, length = 8)]) + rnorm(nrow(X), 1.5));
result <- genAlg(y, X, control = ctrl, evaluator = evaluator, seed = 123)
subsets(result, 1:5)

---

**evaluatorLM**

**LM Evaluator**

Description
Create an evaluator that uses a linear model to evaluate the fitness.

Usage
evaluatorLM(
  statistic = c("BIC", "AIC", "adjusted.r.squared", "r.squared"),
  numThreads = NULL
)

Arguments
- statistic: The statistic used to evaluate the fitness
- numThreads: The maximum number of threads the algorithm is allowed to spawn (a value less
  than 1 or NULL means no threads)
Different statistics to evaluate the fitness of the variable subset can be given. If a maximum absolute correlation is given the algorithm will be very slow (as the C++ implementation can not be used anymore) and multithreading is not available.

Value

Returns an S4 object of type GenAlgMEvaluator

See Also

Other GenAlg Evaluators: evaluatorFit(), evaluatorPLS(), evaluatorUserFunction()

Examples

```r
ctrl <- genAlgControl(populationSize = 200, numGenerations = 30, minVariables = 5,
maxVariables = 12, verbosity = 1)
evaluator <- evaluatorLM(statistic = "BIC", numThreads = 1)

# Generate demo-data
set.seed(12345)
X <- matrix(rnorm(10000, sd = 1:5), ncol = 50, byrow = TRUE)
y <- drop(-1.2 + rowSums(X[, seq(1, 43, length = 8)]) + rnorm(nrow(X), 1.5));

result <- genAlg(y, X, control = ctrl, evaluator = evaluator, seed = 123)
subsets(result, 1:5)
```

---

evaluatorPLS  

**PLS Evaluator**

Description

Creates the object that controls the evaluation step in the genetic algorithm

Usage

evaluatorPLS(
  numReplications = 30L,
  innerSegments = 7L,
  outerSegments = 1L,
  testSetSize = NULL,
  numThreads = NULL,
  maxNComp = NULL,
  method = c("simpls"),
  sdfact = 1
)
Arguments

- **numReplications**: The number of replications used to evaluate a variable subset (must be between 1 and $2^{16}$)
- **innerSegments**: The number of CV segments used in one replication (must be between 2 and $2^{16}$)
- **outerSegments**: The number of outer CV segments used in one replication (between 0 and $2^{16}$). If this is greater than 1, repeated double cross-validation strategy (rdCV) will be used instead of simple repeated cross-validation (srCV) (see details)
- **testSetSize**: The relative size of the test set used for simple repeated CV (between 0 and 1). This parameter is ignored if outerSegments > 1 and a warning will be issued.
- **numThreads**: The maximum number of threads the algorithm is allowed to spawn (a value less than 1 or NULL means no threads)
- **maxNComp**: The maximum number of components the PLS models should consider (if not specified, the number of components is not constrained)
- **method**: The PLS method used to fit the PLS model (currently only SIMPLS is implemented)
- **sdfact**: The factor to scale the stand. dev. of the MSEP values when selecting the optimal number of components. For the "one standard error rule", sdfact is 1.

Details

With this method the genetic algorithm uses PLS regression models to assess the prediction power of variable subsets. By default, simple repeated cross-validation (srCV) is used. The optimal number of PLS components is estimated using cross-validation (with innerSegments segments) on a training set. The prediction power is then evaluated by fitting a PLS regression model with this optimal number of components to the training set and predicting the values of a test set (of either testSetSize size or $1 / \text{innerSegments}$, if testSetSize is not specified).

If the parameter outerSegments is given, repeated double cross-validation is used instead. There, the data set is first split into outerSegments segments and one segment is used as prediction set and the other segments as test set. This is repeated for each outer segment.

The whole procedure is repeated numReplications times to get a more reliable estimate of the prediction power.

Value

Returns an S4 object of type **GenAlgPLSEvaluator** to be used as argument to a call of **genAlg**.

See Also

Other GenAlg Evaluators: **evaluatorFit()**, **evaluatorLM()**, **evaluatorUserFunction()**
**Examples**

```r
ctrl <- genAlgControl(populationSize = 100, numGenerations = 15, minVariables = 5, maxVariables = 12, verbosity = 1)
evaluatorSRCV <- evaluatorPLS(numReplications = 2, innerSegments = 7, testSetSize = 0.4, numThreads = 1)
evaluatorRDCV <- evaluatorPLS(numReplications = 2, innerSegments = 5, outerSegments = 3, numThreads = 1)

# Generate demo-data
set.seed(12345)
X <- matrix(rnorm(10000, sd = 1:5), ncol = 50, byrow = TRUE)
y <- drop(-1.2 + rowSums(X[, seq(1, 43, length = 8)]) + rnorm(nrow(X), 1.5));

resultSRCV <- genAlg(y, X, control = ctrl, evaluator = evaluatorSRCV, seed = 123)
resultRDCV <- genAlg(y, X, control = ctrl, evaluator = evaluatorRDCV, seed = 123)

subsets(resultSRCV, 1:5)
subsets(resultRDCV, 1:5)
```

---

**evaluatorUserFunction**  
*User Defined Evaluator*

**Description**

Create an evaluator that uses a user defined function to evaluate the fitness

**Usage**

```r
evaluatorUserFunction(FUN, sepFUN = NULL, ...)
```

**Arguments**

- **FUN**
  - Function used to evaluate the fitness

- **sepFUN**
  - Function to calculate the SEP of the variable subsets

- **...**
  - Additional arguments passed to FUN and sepFUN

**Details**

The user specified function must take a the response vector as first and the covariates matrix as second argument. The function must return a number representing the fitness of the variable subset (the higher the value the fitter the subset). Additionally the user can specify a function that takes a `GenAlg` object and returns the standard error of prediction of the found variable subsets.

**Value**

Returns an S4 object of type `GenAlgUserEvaluator`
### See Also

Other GenAlg Evaluators: `evaluatorFit()`, `evaluatorLM()`, `evaluatorPLS()`

### Examples

```r
ctrl <- genAlgControl(populationSize = 100, numGenerations = 10, minVariables = 5,
                      maxVariables = 12, verbosity = 1)

# Use the BIC of a linear model to evaluate the fitness of a variable subset
evalFUN <- function(y, X) {
  return(BIC(lm(y ~ X)));}

# Dummy function that returns the residuals standard deviation and not the SEP
sepFUN <- function(genAlg) {
  return(apply(genAlg@subsets, 2, function(subset) {
    m <- lm(genAlg@response ~ genAlg@covariates[, subset]);
    return(sd(m$residuals));
  }));
}

evaluator <- evaluatorUserFunction(FUN = evalFUN, sepFUN = sepFUN)

# Generate demo-data
set.seed(12345)
X <- matrix(rnorm(10000, sd = 1:5), ncol = 50, byrow = TRUE)
y <- drop(-1.2 + rowSums(X[, seq(1, 43, length = 8)]) + rnorm(nrow(X), 1.5));
result <- genAlg(y, X, control = ctrl, evaluator = evaluator, seed = 123)
subsets(result, 1:5)
```

---

#### fitness

*Get the fitness of a variable subset*

### Description

Get the internal fitness for all variable subsets

### Usage

```r
fitness(object)
```

### Arguments

- **object**
  - The `GenAlg` object returned by `genAlg`

### Details

This method is used to get the fitness of all variable subsets found by the genetic algorithm.
fitnesEvolution

Value

A vector with the estimated fitness for each solution

Examples

ctrl <- genAlgControl(populationSize = 100, numGenerations = 15, minVariables = 5,
                      maxVariables = 12, verbosity = 1)

evaluator <- evaluatorPLS(numReplications = 2, innerSegments = 7, testSetSize = 0.4,
                         numThreads = 1)

# Generate demo-data
set.seed(12345)
X <- matrix(rnorm(10000, sd = 1:5), ncol = 50, byrow = TRUE)
y <- drop(-1.2 + rowSums(X[, seq(1, 43, length = 8)]) + rnorm(nrow(X), 1.5));

result <- genAlg(y, X, control = ctrl, evaluator = evaluator, seed = 123)

fitness(result) # Get fitness of the found subsets

h <- fitnessEvolution(result) # Get average fitness as well as the fitness of the
    # best chromosome for each generation (at raw scale!)

plot(h[, "mean"], type = "l", col = 1, ylim = c(-7, -1))
lines(h[, "mean"] - h[, "std.dev"], type = "l", col = "gray30", lty = 2)
lines(h[, "mean"] + h[, "std.dev"], type = "l", col = "gray30", lty = 2)
lines(h[, "best"], type = "l", col = 2)

fitnessEvolution Get the evolution of the fitness

Description

Get the fitness of the best / average chromosomes after each generation

Usage

fitnessEvolution(
  object,
  what = c("mean", "best", "std.dev"),
  type = c("true", "raw")
)

Arguments

object The GenAlg object returned by genAlg
what can be one or more of "best" (to return the fitness of the best chromosome for each generation), "mean" (to return the arithmetic mean fitness during each generation), and "std.dev" (for the standard deviation of the fitness values in each generation).

type one of "true" or "raw". raw means the raw fitness value used within the GA, while true tries to convert it to the standard error of prediction (like fitness). If the standard deviation (what = "std.dev") is requested, the type will always be raw.

Details

Returns the progress of the fitness of the best or average chromosome.

Value

A vector with the best or average fitness value after each generation

Examples

ctrl <- genAlgControl(populationSize = 100, numGenerations = 15, minVariables = 5, maxVariables = 12, verbosity = 1)
evaluator <- evaluatorPLS(numReplications = 2, innerSegments = 7, testSetSize = 0.4, numThreads = 1)

# Generate demo-data
set.seed(12345)
X <- matrix(rnorm(10000, sd = 1:5), ncol = 50, byrow = TRUE)
y <- drop(-1.2 + rowSums(X[, seq(1, 43, length = 8)]) + rnorm(nrow(X), 1.5));
result <- genAlg(y, X, control = ctrl, evaluator = evaluator, seed = 123)

fitness(result) # Get fitness of the found subsets
h <- fitnessEvolution(result) # Get average fitness as well as the fitness of the # best chromosome for each generation (at raw scale!)

plot(h[,, "mean"], type = "l", col = 1, ylim = c(-7, -1))
lines(h[, "mean"], type = "l", col = "gray30", lty = 2)
lines(h[, "mean"] + h[, "std.dev"], type = "l", col = "gray30", lty = 2)
lines(h[, "best"], type = "l", col = 2)

formatSegmentation

Format the raw segmentation list returned from the C++ code into a usable list

Description

Format the raw segmentation list returned from the C++ code into a usable list
genAlg

**Usage**

```r
formatSegmentation(object, segments)
## S4 method for signature 'GenAlgPLSEvaluator,list'
formatSegmentation(object, segments)
## S4 method for signature 'GenAlgUserEvaluator,list'
formatSegmentation(object, segments)
## S4 method for signature 'GenAlgLMEvaluator,list'
formatSegmentation(object, segments)
## S4 method for signature 'GenAlgFitEvaluator,list'
```

**Arguments**

- `object` The Evaluator object.
- `segments` The raw segmentation list.

**Value**

A list of the form `replication -> outerSegment -> (calibration, validation, inner -> (test, train))`

---

**genAlg**

*Genetic algorithm for variable subset selection*

**Description**

A genetic algorithm to find "good" variable subsets based on internal PLS evaluation or a user specified evaluation function

**Usage**

```r
genAlg(y, X, control, evaluator = evaluatorPLS(), seed)
```

**Arguments**

- `y` The numeric response vector of length n
- `X` A n x p numeric matrix with all p covariates
- `control` Options for controlling the genetic algorithm. See `genAlgControl` for details.
- `evaluator` The evaluator used to evaluate the fitness of a variable subset. See `evaluatorPLS`, `evaluatorLM` or `evaluatorUserFunction` for details.
- `seed` Integer with the seed for the random number generator or NULL to automatically seed the RNG
Details
The GA generates an initial "population" of \( \text{populationSize} \) chromosomes where each initial chromosome has a random number of randomly selected variables. The fitness of every chromosome is evaluated by the specified evaluator. The default built-in PLS evaluator (see \texttt{evaluatorPLS}) is the preferred evaluator. Chromosomes with higher fitness have higher probability of mating with another chromosome. \( \text{populationSize} / 2 \) couples each create 2 children. The children are created by randomly mixing the parents’ variables. These children make up the new generation and are again selected for mating based on their fitness. A total of \( \text{numGenerations} \) generations are built this way. The algorithm returns the last generation as well as the best elitism chromosomes from all generations.

Value
An object of type \texttt{GenAlg}

Examples
\begin{verbatim}
ctrl <- genAlgControl(populationSize = 100, numGenerations = 15, minVariables = 5, maxVariables = 12, verbosity = 1)
evaluatorSRCV <- evaluatorPLS(numReplications = 2, innerSegments = 7, testSetSize = 0.4, numThreads = 1)
evaluatorRDCV <- evaluatorPLS(numReplications = 2, innerSegments = 5, outerSegments = 3, numThreads = 1)

# Generate demo-data set.seed(12345)
X <- matrix(rnorm(10000, sd = 1:5), ncol = 50, byrow = TRUE)
y <- drop(-1.2 + rowSums(X[, seq(1, 43, length = 8)]) + rnorm(nrow(X), 1.5));

resultSRCV <- genAlg(y, X, control = ctrl, evaluator = evaluatorSRCV, seed = 123)
resultRDCV <- genAlg(y, X, control = ctrl, evaluator = evaluatorRDCV, seed = 123)

subsets(resultSRCV, 1:5)
subsets(resultRDCV, 1:5)
\end{verbatim}
rawFitness Numeric vector with the raw fitness of the corresponding variable subset returned by the evaluator.

response The original response vector.

covariates The original covariates matrix.

evaluator The evaluator used in the genetic algorithm.

control The control object.

segmentation The segments used by the evaluator. Empty list if the evaluator doesn’t use segmentation.

seed The seed the algorithm is started with.

---

genAlgControl Set control arguments for the genetic algorithm

**Description**

The population must be large enough to allow the algorithm to explore the whole solution space. If the initial population is not diverse enough, the chance to find the global optimum is very small. Thus the more variables to choose from, the larger the population has to be.

**Usage**

```r
genAlgControl(
  populationSize,  # The number of "chromosomes" in the population (between 1 and 2^16)
  numGenerations,  # The number of generations to produce (between 1 and 2^16)
  minVariables,    # The minimum number of variables in the variable subset (between 0 and p - 1
  maxVariables,    # where p is the total number of variables)
  elitism = 10L,   #
  mutationProbability = 0.01,
  crossover = c("single", "random"),
  maxDuplicateEliminationTries = 0L,
  verbosity = 0L,
  badSolutionThreshold = 2,
  fitnessScaling = c("none", "exp")
)
```

**Arguments**

- `populationSize`: The number of "chromosomes" in the population (between 1 and 2^16).
- `numGenerations`: The number of generations to produce (between 1 and 2^16).
- `minVariables`: The minimum number of variables in the variable subset (between 0 and p - 1 where p is the total number of variables).
- `maxVariables`: The maximum number of variables in the variable subset (between 1 and p, and greater than `minVariables`).
elitism
The number of absolute best chromosomes to keep across all generations (between 1 and min(populationSize * numGenerations, 2^16))

mutationProbability
The probability of mutation (between 0 and 1)

crossover
The crossover type to use during mating (see details). Partial matching is performed.

maxDuplicateEliminationTries
The maximum number of tries to eliminate duplicates (a value of 0 or NULL means that no checks for duplicates are done.

verbosity
The level of verbosity. 0 means no output at all, 2 is very verbose.

badSolutionThreshold
The worst child must not be more than badSolutionThreshold times worse than the worse parent. If less than 0, the child must be even better than the worst parent. If the algorithm can’t find a better child in a long time it issues a warning and uses the last found child to continue.

fitnessScaling
How the fitness values are internally scaled before the selection probabilities are assigned to the chromosomes. See the details for possible values and their meaning.

Details
The initial population is generated randomly. Every chromosome uses between minVariables and maxVariables (uniformly distributed).

If the mutation probability (mutationProbability is greater than 0, a random number of variables is added/removed according to a truncated geometric distribution to each offspring-chromosome. The resulting distribution of the total number of variables in the subset is not uniform anymore, but almost (the smaller the mutation probability, the more "uniform" the distribution). This should not be a problem for most applications.

The user can choose between single and random crossover for the mating process. If single crossover is used, a single position is randomly chosen that marks the position to split both parent chromosomes. The child chromosomes are then the concatenated chromosomes from the 1st part of the 1st parent and the 2nd part of the 2nd parent resp. the 2nd part of the 1st parent and the 1st part of the 2nd parent. Random crossover is that a random number of random positions are drawn and these positions are transferred from one parent to the other in order to generate the children.

Elitism is a method of enhancing the GA by keeping track of very good solutions. The parameter elitism specifies how many "very good" solutions should be kept.
Before the selection probabilities are determined, the fitness values f of the chromosomes are standardized to the z-scores \( z = (f - mu) / sd \). Scaling the fitness values afterwards with the exponential function can help the algorithm to faster find good solutions. When setting fitnessScaling to "exp", the (standardized) fitness z will be scaled by exp(z). This promotes good solutions to get an even higher selection probability, while bad solutions will get an even lower selection probability.

Value
An object of type GenAlgControl
Examples
ctrl <- genAlgControl(populationSize = 100, numGenerations = 15, minVariables = 5,
maxVariables = 12, verbosity = 1)
evaluatorSRCV <- evaluatorPLS(numReplications = 2, innerSegments = 7, testSetSize = 0.4,
numThreads = 1)
evaluatorRDCV <- evaluatorPLS(numReplications = 2, innerSegments = 5, outerSegments = 3,
numThreads = 1)

# Generate demo-data
set.seed(12345)
X <- matrix(rnorm(10000, sd = 1:5), ncol = 50, byrow = TRUE)
y <- drop(-1.2 + rowSums(X[, seq(1, 43, length = 8)]) + rnorm(nrow(X), 1.5));

resultSRCV <- genAlg(y, X, control = ctrl, evaluator = evaluatorSRCV, seed = 123)
resultRDCV <- genAlg(y, X, control = ctrl, evaluator = evaluatorRDCV, seed = 123)
subsets(resultSRCV, 1:5)
subsets(resultRDCV, 1:5)

---

GenAlgControl-class  Control class for the genetic algorithm

Description
This class controls the general setup of the genetic algorithm

Slots
populationSize The number of "chromosomes" in the population (between 1 and 2^16).
numGenerations The number of generations to produce (between 1 and 2^16).
minVariables The minimum number of variables in the variable subset (between 0 and p - 1 where p is the total number of variables).
maxVariables The maximum number of variables in the variable subset (between 1 and p, and greater than minVariables).
elitism The number of absolute best chromosomes to keep across all generations (between 1 and min(populationSize * numGenerations, 2^16)).
multiplicationProbability The probability of mutation (between 0 and 1).
badSolutionThreshold The child must not be more than badSolutionThreshold percent worse than the worse parent. If less than 0, the child must be even better than the worst parent.
crossover The crossover method to use
crossoverId The numeric ID of the crossover method to use
maxDuplicateEliminationTries The maximum number of tries to eliminate duplicates
verbosity The level of verbosity. 0 means no output at all, 2 is very verbose.
GenAlgEvaluator-class  
*Evaluator Base Class*

**Description**

Virtual base class of all available evaluators

---

GenAlgFitEvaluator-class  
*Fit Evaluator*

**Description**

Fit Evaluator

**Slots**

- **numSegments**  The number of CV segments used in one replication.
- **numThreads**  The maximum number of threads the algorithm is allowed to spawn (a value less than 1 or NULL means no threads).
- **maxNComp**  The maximum number of components to consider in the PLS model.
- **sdfact**  The factor to scale the stand. dev. of the MSEP values when selecting the optimal number of components. For the "one standard error rule", sdfact is 1.
- **statistic**  The statistic used to evaluate the fitness.
- **statisticId**  The (internal) numeric ID of the statistic.

---

GenAlgLMEvaluator-class  
*LM Evaluator*

**Description**

LM Evaluator

**Slots**

- **statistic**  The statistic used to evaluate the fitness.
- **statisticId**  The (internal) numeric ID of the statistic.
- **numThreads**  The maximum number of threads the algorithm is allowed to spawn (a value less than 1 or NULL means no threads).
GenAlgPLSEvaluator-class

PLS Evaluator

Description

PLS Evaluator

Slots

numReplications  The number of replications used to evaluate a variable subset.
innerSegments  The number of inner RDCV segments used in one replication.
outerSegments  The number of outer RDCV segments used in one replication.
testSetSize  The relative size of the test set (between 0 and 1).
sdfact  The factor to scale the stand. dev. of the MSEP values when selecting the optimal number of components. For the "one standard error rule", sdfact is 1.
numThreads  The maximum number of threads the algorithm is allowed to spawn (a value less than 1 or NULL means no threads).
maxNComp  The maximum number of components to consider in the PLS model.
method  The PLS method used to fit the PLS model (currently only SIMPLS is implemented).
methodId  The ID of the PLS method used to fit the PLS model (see C++ code for allowed values).

GenAlgUserEvaluator-class

User Function Evaluator

Description

User Function Evaluator

Slots

evalFunction  The function that is called to evaluate the variable subset.
sepFunction  The function that calculates the standard error of prediction for the found subsets.
getEvalFun

Get the evaluation function from a GenAlgUserEvaluator

Description

This method returns the correct evaluation function from a GenAlgUserEvaluator that can be used by the C++-code as callback or NULL for any other evaluator.

Usage

getEvalFun(object, genAlg)

## S4 method for signature 'GenAlgUserEvaluator,GenAlg'
getEvalFun(object, genAlg)

## S4 method for signature 'GenAlgUserEvaluator, matrix'
getEvalFun(object, genAlg)

## S4 method for signature 'GenAlgEvaluator,GenAlg'
getEvalFun(object, genAlg)

## S4 method for signature 'GenAlgEvaluator, matrix'
getEvalFun(object, genAlg)

Arguments

object The evaluator (an object of type GenAlgEvaluator)
genAlg The GenAlg object

subsets

Get the found variable subset(s)

Description

Get a list of variable indices/names of the found variable subsets.

Usage

subsets(object, indices, names = TRUE)

Arguments

object The GenAlg object returned by genAlg.
indices The indices of the subsets or empty if all subsets should be returned.
names Should the names or the column numbers of the variables be returned.
Details

This method is used to get the names or indices of the variables used in specified variable subsets.

Value

A logical matrix where each column represents a variable subset

Examples

```r
ctrl <- genAlgControl(populationSize = 200, numGenerations = 15, minVariables = 5,
                       maxVariables = 12, verbosity = 1)

evaluator <- evaluatorPLS(numReplications = 2, innerSegments = 7, testSetSize = 0.4,
                           numThreads = 1)

# Generate demo-data
set.seed(12345)
X <- matrix(rnorm(10000, sd = 1:5), ncol = 50, byrow = TRUE)
y <- drop(-1.2 + rowSums(X[, seq(1, 43, length = 8)]) + rnorm(nrow(X), 1.5))

result <- genAlg(y, X, control = ctrl, evaluator = evaluator, seed = 123)

subsets(result, names = TRUE, indices = 1:5) # best 5 variable subsets as a list of names
result@subsets[, 1:5] # best 5 variable subsets as a logical matrix with the subsets in the columns
```

Description

Get the control list for the C++ procedure genAlgPLS from the object

Usage

```r
toCControlList(object)
```

## S4 method for signature 'GenAlgPLSEvaluator'
toCControlList(object)

## S4 method for signature 'GenAlgFitEvaluator'
toCControlList(object)

## S4 method for signature 'GenAlgUserEvaluator'
toCControlList(object)

## S4 method for signature 'GenAlgLMEvaluator'
toCControlList(object)

## S4 method for signature 'GenAlgPostProc'
toCControlList(object)

## S4 method for signature 'GenAlgResid'
toCControlList(object)

## S4 method for signature 'GenAlgResidEvaluator'
toCControlList(object)

## S4 method for signature 'GenAlgUserResidEvaluator'
toCControlList(object)

## S4 method for signature 'GenAlgUnique'
toCControlList(object)

## S4 method for signature 'GenAlgUserUnique'
toCControlList(object)

## S4 method for signature 'GenAlgUserUniqueEvaluator'
toCControlList(object)
trueFitnessVal

## S4 method for signature 'GenAlgControl'

toCControlList(object)

### Arguments

- **object**: The object

### Value

A list with all items expected by the C++ code

---

traceFitness:

*Get the transformed fitness values*

### Description

Transform the given fitness values according to the GenAlgEvaluator class

#### Usage

trueFitnessVal(object, fitness)

---

### Arguments

- **object**: The used evaluator, an object with type or with a subtype of `GenAlgEvaluator`
- **fitness**: A numeric vector of fitnesses

### Details

This method is used to calculate the true fitness given the GenAlgEvaluator class (as they use different internal fitness measures)

### Value

A vector with the true fitness values
validData

Check if the data is valid for the evaluator

Description
This method checks if the covariates matrix is valid for the evaluator

Usage
validData(object, genAlg)

## S4 method for signature 'GenAlgPLSEvaluator,GenAlg'
validData(object, genAlg)

## S4 method for signature 'GenAlgFitEvaluator,GenAlg'
validData(object, genAlg)

## S4 method for signature 'GenAlgLMEvaluator,GenAlg'
validData(object, genAlg)

## S4 method for signature 'GenAlgEvaluator,GenAlg'
validData(object, genAlg)

Arguments

object The evaluator
genAlg The GenAlg object the evaluator is used in
Index

evaluate, 2
evaluate, GenAlgEvaluator, matrix, numeric, ANY, integer, numeric, missing-method
  (evaluate), 2
evaluate, GenAlgEvaluator, matrix, numeric, ANY, integer-method
  (evaluate), 2
evaluate, GenAlgEvaluator, matrix, numeric, ANY, missing-method
  (evaluate), 2
evaluate, GenAlgEvaluator, matrix, numeric, logical, integer, integer-method
  (evaluate), 2
evaluate, GenAlgEvaluator, matrix, numeric, matrix, integer, integer-method
  (evaluate), 2
evaluatorFit, 3, 5, 6, 8
evaluatorLM, 4, 4, 6, 8, 11
evaluatorPLS, 4, 5, 5, 8, 11, 12
evaluatorUserFunction, 4–6, 7, 11

fitness, 8, 10
fitnessEvolution, 9
formatSegmentation, 10
formatSegmentation, GenAlgFitEvaluator, list-method
  (formatSegmentation), 10
formatSegmentation, GenAlgLMEvaluator, list-method
  (formatSegmentation), 10
formatSegmentation, GenAlgPLSEvaluator, list-method
  (formatSegmentation), 10
formatSegmentation, GenAlgUserEvaluator, list-method
  (formatSegmentation), 10

GenAlg, 7–9, 12, 18
GenAlg (GenAlg-class), 12
genAlg, 4, 6, 8, 9, 11, 18
GenAlg-class, 12
GenAlgControl, 14
GenAlgControl (GenAlgControl-class), 15
genAlgControl, 11, 13
GenAlgControl-class, 15
GenAlgEvaluator, 18, 20
GenAlgEvaluator
  (GenAlgEvaluator-class), 16
GenAlgEvaluator-class, 16
GenAlgFitEvaluator, 4
GenAlgFitEvaluator-class, 16
GenAlgLMEvaluator, 5
GenAlgLMEvaluator-class, 16
GenAlgPLSEvaluator, 6
GenAlgPLSEvaluator-class, 17
GenAlgUserEvaluator, 7
GenAlgUserEvaluator-class, 17
getEvalFun, 18
getEvalFun, GenAlgEvaluator, GenAlg-method
  (getEvalFun), 18
toCControlList, 19
toCControlList, GenAlgControl-method
  (toCControlList), 19
toCControlList, GenAlgFitEvaluator-method
  (toCControlList), 19
toCControlList, GenAlgLMEvaluator-method
  (toCControlList), 19
toCControlList, GenAlgPLSEvaluator-method
  (toCControlList), 19
toCControlList, GenAlgUserEvaluator-method
  (toCControlList), 19
trueFitnessVal, 20
trueFitnessVal, GenAlgFitEvaluator, numeric-method (trueFitnessVal), 20
trueFitnessVal, GenAlgLMEvaluator, numeric-method (trueFitnessVal), 20
trueFitnessVal, GenAlgPLSEvaluator, numeric-method (trueFitnessVal), 20
trueFitnessVal, GenAlgUserEvaluator, numeric-method (trueFitnessVal), 20
validData, 21
validData, GenAlgEvaluator, GenAlg-method (validData), 21
validData, GenAlgFitEvaluator, GenAlg-method (validData), 21
validData, GenAlgLMEvaluator, GenAlg-method (validData), 21
validData, GenAlgPLSEvaluator, GenAlg-method (validData), 21