Package 'galgo'

March 25, 2014

Type Package

Version 1.2

Date 2014-03-19

Title Genetic Algorithms for Multivariate Statistical Models from Large-scale Functional Genomics Data

Author@R c(person(``Victor", ``Trevino", role=``aut",email=``vtrevino@itesm.mx"), person(``Francesco", ``Falciani",role=``aut"))

Author Victor Trevino & Francesco Falciani

Maintainer Victor Trevino <vtrevino@itesm.mx>

Description Galgo attempt to build multivariate predictive models from large datases having far larger number of features than samples such as in functional genomics datasets

License GPL-2

LazyLoad no

Depends R.oo, MASS, class, e1071, rpart, nnet, randomForest

URL http://bioinformatica.mty.itesm.mx/?q=node/82

R topics documented:

`galgo-package` *Galgo perform feature selection from large scale data.*

Description

Represents a genetic algorithm (GA) itself. The basic GA uses at least one population of chromosomes, a "fitness" function, and a stopping rule (see references).

The Galgo object is not limited to a single population, it implements a list of populations where any element in the list can be either a `Niche` object or a `World` object. Nervertheless, any user-defined object that implements `evolve, progeny, best, max, bestFitness, and maxFitness` methods can be part of the `populations` list.

The "fitness" function is by far the most important part of a GA, it evaluates a `Chromosome` to determine how good the chromosome is respect to a given goal. The function can be sensitive to data stored in `.GlobalEnv` or any other object (see `*evaluate()` for further details). For this package and in the case of the microarray, we have included several fitness functions to classify samples using different methods. However, it is not limited for a classification problem for microarray data, because you can create any fitness function in any given context.

The stopping rule has three options. First, it is simply a desired fitness value implemented as a numeric `fitnessGoal`, and If the maximum fitness value of a population is equal or higher than `fitnessGoal` the GA ends. Second, `maxGenerations` determine the maximum number of generations a GA can evolve. The current generation is increased after evaluating the fitness function to the entire population list. Thus, if the current generation reach `maxGenerations` the GA stops. Third, if the result of the user-defined `callBackFunc` is `NA` the GA stops. In addition, you can always break any R program using `Ctrl-C` (or `Esc` in Windows).

When the GA ends many values are used for futher analysis. Examples are the best chromosome (`best` method), its fitness (`bestFitness` method), the final generation (`generation` variable), the evolution of the maximum fitness (`maxFitnesses` list variable), the maximum chromosome in each generation (`maxChromosome` list variable), and the elapsed time (`elapsedTime` variable). Moreover, flags like `goalScored`, `userCancelled`, and `running` are available.

Details

Package:	galgo
Type:	Package
Version:	1.0
Date:	2011-05-31
License:	What license is it under?
LazyLoad:	yes

See BigBang and Galgo Objects for usage.

Author(s)

Victor Trevino and Francesco Falciani

Maintainer: Victor Trevino <vtrevino@itesm.mx>

References

GALGO: An R Package For Multivariate Variable Selection Using Genetic Algorithms Victor Trevino and Francesco Falciani School of Biosciences, University of Birmingham, Edgbaston, UK Bioinformatics 2006

See Also

BigBang and Galgo Objects.

Examples

```
## Not run:
   bb <- configBB.VarSel(...) #not runs

## End(Not run)
```

`ALL` *Acute Lymphoblastic Leukemia data (Yeoh et. al., 2002) for GALGO package*

Description

Acute Lymphoblastic Leukemia for GALGO package data published by Yeoh et. al. The original 360 pediatric acute leukemia samples were filtered by class. 233 samples are included corresponding to 5 classes EMLLA, HYP+50, MLL, T, and TEL. The Affymetrix microarray HG_U95Av2 containing 12,600 probesets were filteres by range and standard deviation resulting in 2,435 probesets (genes).

Usage

```
data(ALL)
```

Format

The format is: 2,435 Rows : Genes 233 Columns : Samples Row Names : ProbeId Col Names : Samples Id 5 Classes : Lukemia Types: "EMLLA"=E2A-PBX1, "T"=T-ALL, "HYP+50"=Hyperdiploid > 50 Chromosomes, "MLL"=MLL rearragment, and "TEL"=TEL-AML1

Details

ALL data is complmented by ALL.classes which contain the classes for each column sample.

References

Eng-Juh Yeoh, Mary E. Ross, Sheila A. Shurtleff, W. Kent Williams, Divyen Patel, Rami Mahfouz, Fred G. Behm, Susana C. Raimondi, Mary V. Relling, Anami Patel, Cheng Cheng, Dario Campana,, Dawn Wilkins, Xiaodong Zhou, Jinyan Li, Huiqing Liu, Ching-Hon Pui, William E. Evans, Clayton Naeve, Limsoon Wong, and James R. Downing. *Classification, subtype discovery, and prediction of outcome in pediatric acute lymphoblastic leukemia by gene expression profiling*. Cancer Cell. March 2002.

Examples

```
data(ALL)
data(ALL.classes)
```

ALL.classes — *Acute Lymphoblastic Leukemia data (Yeoh et. al., 2002) for GALGO package*

Description

Acute Lymphoblastic Leukemia for GALGO package data published by Yeoh et. al. The original 360 pediatric acute leukemia samples were filtered by class. 233 samples are included corresponding to 5 classes EMLLA, HYP+50, MLL, T, and TEL. The Affymetrix microarray HG_U95Av2 containing 12,600 probesets were filteres by range and standard deviation resulting in 2,435 probesets (genes).

Usage

```
data(ALL.classes)
```

Format

5 Classes : Lukemia Types: "EMLLA"=E2A-PBX1, "T"=T-ALL, "HYP+50"=Hyperdiploid > 50 Chromosomes, "MLL"=MLL rearragment, and "TEL"=TEL-AML1

Details

ALL.classes is complementary for ALL data which contain the expression values for the genes in all samples.

References

Eng-Juh Yeoh, Mary E. Ross, Sheila A. Shurtleff, W. Kent Williams, Divyen Patel, Rami Mahfouz, Fred G. Behm, Susana C. Raimondi, Mary V. Relling, Anami Patel, Cheng Cheng, Dario Campana,, Dawn Wilkins, Xiaodong Zhou, Jinyan Li, Huiqing Liu, Ching-Hon Pui, William E. Evans, Clayton Naeve, Limsoon Wong, and James R. Downing. *Classification, subtype discovery, and prediction of outcome in pediatric acute lymphoblastic leukemia by gene expression profiling*. Cancer Cell. March 2002.

Examples

```
data(ALL)
data(ALL.classes)
```

`as.list.Object` *Convert a variable of class Object to a list*

Description

Object variables behave as lists, however they are really enviroments. Sometimes it is necesary to use the variable as a list instead of an Object. This function converts the Object to a list.

Usage

```
as.list(x, ...)
```

Arguments

`x`	Variable of class Object
`...`	Other object to include

Value

Returns a list with values equivalent to the Object.

Note

Values that contain functions will be assigned to .GlobalEnv enviroment.

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K.

Examples

```
xO <- Object()
xO$var = "hello"
class(xO)
xOL <- as.list(xO)
xOL
class(xOL)
```

Bag *A list-like Object*

Description

Create a list of values. Lists inside an `Object` behave as by value (if the list is modified in a method, the original list is not updated). Therefore, `Bag` replace this behaviour extending `Object` and allowing to save reference-lists inside objects.

Usage

```
Bag(...)
```

Arguments

`...`	Values to store in the `Bag` object.

Class

Package: galgo
Class Bag

```
Object
~~|
~~+--Bag
```

Directly known subclasses:

public static class **Bag**
extends Object

Fields and Methods

Methods:

`length`	Gets the length of the object as its list version.
`print`	Prints the representation of the Bag object.
`summary`	Prints the representation of the Bag object.

Methods inherited from Object:
as.list, unObject, $, $<-, [[, [[<-, as.character, attach, clone, detach, equals, extend, finalize, getFields, getInstanciationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, save

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K. http://www.bip.bham.ac.uk/bioinf

See Also

See also `list()`.

Examples

```
b <- Bag(a=1,b=2,c=3)
b
as.list(b)
unObject(b)
```

`BigBang` *Represents the ensemble of the results of evolving several Galgo objects*

Description

The `BigBang` object is an attempt to use more the information of a large collection of solutions instead of a unique solution. Perhaps we are studying the solution landscape or we would like to "ensemble" solutions from other "small" solutions. For complex problems (or even simple problems), the number of "solutions" may be very large and diverse. In the context of classification for microarray data, we have seen that models assembled from many solution could be used as "general models" and that the most frequent genes in solutions provide insights for biological phenomena.

Therefore, we designed the `BigBang` object, which implements methods to run a `Galgo` object several times recording relevant information from individual galgos for further analysis. Running a BigBang takes commonly several minutes, hours or perhaps days depending on the complexity of the fitness function, the data, the `goalFitness`, the stopping rules in `Galgo`, and the number of solutions to collect. Parallelism is not explicity implemented but some methods has been implemented to make this task easy and possible.

As in a `Galgo` object, there are three stopping methods: `maxBigBangs`, `maxSolutions` and `callBackFunc`. `maxBigBangs` controls the maximum number of galgo evolutions to run; when the current evolution-cycle reaches this value, the process ends. Sometimes evolutions do not end up with a `goalFitness` reached, this is not called a "solution". Therefore, `maxSolutions` controls the maximum number of solutions desired. If `onlySolutions==FALSE`, all galgo evolutions are saved and considered as "solution", nevertheless the `solution` variable save the real status in the `BigBang` object. `callBackFunc` may ends the process if it returns `NA`. It must be considered that any R-program can be broken typing `Ctrl-C` (`Esc` in Windows). If for some reason the process has been interrupt, the `BigBang` process can continue processing the same cycle just calling the method `blast` again. However the object integrity may be risked if the process is broken in critical parts (when the object is being updated at the end of each cycle). Thus, it is recommended to break the process in the galgo "evolution".

In the case of variable selection for microarray data, some methods has been proposed that use several independent solutions to design a final solution (or set of better solutions, see XXX references *** MISSING ***).

There is configBB.VarSel and configBB.VarSelMisc functions that configure a BigBang object together with all sub-objects for common variable selection problems (e.g. classification, regression, etc.)

Usage

```
BigBang(id=0,
galgo=NULL,
maxBigBangs=10,
maxSolutions=1,
collectMode=c("bigbang", "galgos", "chromosomes"),
onlySolutions=TRUE,
verbose=1,
callPreFunc=function(bigbang, galgo) TRUE,
callBackFunc=function(bigbang, galgo) TRUE,
callEnhancerFunc=function(chr, parent) NULL,
data=NULL,
saveFile=NULL,
saveFrequency=100,
saveVariableName=collectMode,
saveMode=c("unObject+compress", "unObject", "object", "object+compress"),
saveGeneBreaks=NULL,
geneNames=NULL,
sampleNames=NULL,
classes=NULL,
gcFrequency=123,
gcCalls=5,
call=NULL,
...)
```

Arguments

`id`	A way to identify the object.
`galgo`	The prototype `Galgo` object that will be used to run and collect solutions.
`maxBigBangs`	The maximum number of BigBangs. A bigbang is the evolution of a `Galgo` object using the method `evolve`. When the current number of bigbangs has reached `maxBigBangs` value, the process ends.
`maxSolutions`	The maximum number of solutions. If the total number of solutions collected achieve `maxSolutions` value the process ends. A solution is defined when the `goalFitness` has been reach. When the `Galgo` object ends and `goalFitness` has not been reached, The `best` chromosome is NOT saved unless `onlySolutions` is `FALSE`, in this case `maxSolutions` and `maxBigBangs` are equivalent.
`collectMode`	The type of result to collect for further analysis. `"galgos"` saves every evolved galgo object, thus it consumes a lot of memory; more than 100 is perhaps not recommendable. `"chromosomes"` and `"bigbangs"` save the best chromosome, its fitness, and fitness evolution in the `BigBang` object. `"bigbang"` saves the `BigBang` object to disk whereas `"chromosome"` saves only the list of chromosomes.
`onlySolutions`	If `TRUE` only solutions that has been reach the `goalFitness` are saved. Otherwise, all solutions are saved and counted as "solution" and `$solutions` variable contains the real status.
`verbose`	Instruct the BigBang to display the general information about the process. When `verbose==1` this information is printed every evolution. In general every `verbose` number of generation would produce a line of output. Of course if `verbose==0` would not display a thing at all.

`callPreFunc` A user-function to be called before every evolution. It should receive the `BigBang` and `Galgo` objects. If the result is `NA`, the process ends.

`callBackFunc` A user-function to be called after every evolution. It should receive the `BigBang` and `Galgo` objects. If the result is `NA`, the process ends. When `callBackFunc` is for instance `plot` the trace of the evolution is nicely viewed in a plot; however, in long runs it can consume time and memory.

`callEnhancerFunc`
A user-function to be called after every evolution to improve the solution. It should receive a `Chromosome` and the `BigBang` objects as parameters, and must return a new `Chromosome` object. If the result is `NULL` nothing is saved. The result replace the original evolved chromosomes, which is saved in evolvedChromosomes list variable in the `BigBang` object. For functional genomics data, we have included two general routines called `geneBackwardElimination` and `robustGeneBackwardElimination` to generate "enhanced" chromosomes.

`data` Any user-data can be stored in this variable (but it is not limited to `data`, the user can insert any other like `myData`, `mama.mia` or `whatever` in the `...` argument).

`saveFile` The file name where the objects would be saved (see `collectMode`).

`saveFrequency`
How often the operation of saving would occur. Saving is a time-consuming operation, low values may degradate the performance.

`saveVariableName`
The prefereable variable name used for saving (this will be needed when loading).

`saveMode` Any combinations of the two options `compress` and `unObject`. It can be character vector length 1 or larger. For example, `saveMode=="compress+unObject"` would call `unObject` and save the file using `compress=TRUE`. The vector `c("object","compress")` (or shorter `c("compress")`) would save the `BigBang` object and compressed. It is not recommended to save the crude object because the functions varibles are stuck to environments and R will try to save those environments together, the result can be a waste of disk space and saving time. We strongly recommend `saveMode="unObject+compress"`.

`geneNames` Gene names (if they are discrete and finite).

`sampleNames` Sample names (if any).

`classes` Class of the original samples (useful for classification problems only).

`saveGeneBreaks`
In the case of variable selection for microarray data (and other problems with discrete and finite genes), a summary on the genes selected is computed and saved in each evolution. It is used to facilitate the computation for some plots and others methods. For no-finite gene applications, it may be useful interpreting `saveGeneBreaks` as the breaks needed to create an histogram based on the genes included in the "best".

`gcFrequency` How often the garbage collector would be called. Useful if memory needs to be collected during the process.

`gcCalls` How many calls to garbage collector (we have seen that many consecutive calls to `gc()` is better [R < 2.0]).

`call` Internal use.

`...` Other user named values to include in the object.

Class

Package: galgo
Class BigBang

```
Object
~~|
~~+--BigBang
```

Directly known subclasses:

public static class **BigBang**
extends Object

Fields and Methods

Methods:

activeChromosomeSet	Focus the analysis to different sets of chromosomes.
addCount	Add a chromosome to rank and frequency stability counting.
addRandomSolutions	Adds random pre-existed solutions.
as.matrix	Prints the representation of the BigBang object.
assignParallelFile	Assigns a different saveFile value for parallelization.
blast	Evolves Galgo objects saving the results for further analysis.
buildCount	Builds the rank and frequency stability counting.
classPredictionMatrix	Predicts class for samples from chromosomes.
computeCount	Compute the counts for every gene from a set of chromosomes..
confusionMatrix	Computes the class confusion matrix from a class prediction matrix.
distanceImportanceNetwork	Converts geneImportanceNetwork matrix to distance matrix.
filterSolution	Filters solutions.
fitnessSplits	Computes the fitness function from chromosomes for different splits.
formatChromosome	Converts chromosome for storage in BigBang object.
forwardSelectionModels	Gets the "best" models using top-ranked genes and a forward-selection strategy.
geneCoverage	Computes the fraction of genes present in the top-rank from the total genes present
geneFrequency	Computes the frequency of genes based on chromosomes.
geneImportanceNetwork	Computes the number of times a couple of top-ranked-genes are present in models.
geneRankStability	Computes the rank history for top-ranked genes.
getFrequencies	Computes gene freqencies.
heatmapModels	Plots models using heatmap plot.
loadParallelFiles	Load all files saved during the parallelization.
meanFitness	Computes the "mean" fitness from several solutions.
meanGeneration	Computes the mean number of generations requiered to reach a given fitness value.
mergeBangs	Merges the information from other BigBang objects.
pcaModels	Plots models in principal components space.
plot	Plots about the collected information in a BigBang object.
predict	Predicts the class or fitting of new set of samples.
print	Prints the representation of a BigBang object.
saveObject	Saves the BigBang object into a file in a suitable format.
sensitivityClass	Computes the sensitivity of class prediction.
specificityClass	Computes the specificity of class prediction.

summary	Prints the representation of the BigBang object.

Methods inherited from Object:
as.list, unObject, $, $<-, [[, [[<-, as.character, attach, clone, detach, equals, extend, finalize, getFields, getInstanciationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, save

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K. http://www.bip.bham.ac.uk/bioinf

References

Goldberg, David E. 1989 *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Pub. Co. ISBN: 0201157675

See Also

`Gene`, `Chromosome`, `Niche`, `World`, `Galgo`, `configBB.VarSel()`, `configBB.VarSelMisc()`.

Examples

```
 ## Not run:
   cr <- Chromosome(genes=newCollection(Gene(shape1=1, shape2=100),5))
   ni <- Niche(chromosomes=newRandomCollection(cr, 10))
   wo <- World(niches=newRandomCollection(ni,2))
   ga <- Galgo(populations=newRandomCollection(wo,1), goalFitness = 0.75,
callBackFunc=plot,
               fitnessFunc=function(chr, parent) 5/sd(as.numeric(chr)))

   #evolve(ga) ## not needed here

   bb <- BigBang(galgo=ga, maxSolutions=10, maxBigBangs=10, saveGeneBreaks=1:100)
   blast(bb)
   ## it performs 10 times evolve() onto ga object
   ## every time, it reinitilize and randomize
   ## finally, the results are saved.
   plot(bb)

   #it is missing a microarray classification example

## End(Not run)
```

`Chromosome` *The representation of a set of genes for genetic algorithms*

Description

Represents a set of genes for the genetic algorithm. The chromosome contains all current values of each gene and will be evaluated using a "fitness" function similar to those defined by Goldberg. The fitness function normally depends on the `Galgo` object.

See references for Genetic Algorithms.

Usage

```
Chromosome(id=0,
genes=list(),
getValues=function(x, ...) unlist(lapply(x, ...)),
decode=function(x) genes(x),
values=list(),
...)
```

Arguments

id	A way to identify the object.
genes	A list of defined Gene objects composing the chromosome.
getValues	A function to be evaluated for every gene to obtain a value. In general, the result could be any object in a list. In particular, the default is a vector of current gene values.
decode	A function that converts the chromosome representation in real values. It is used mainly for output purposes and for frequency counting. It has no effect for variable selection in microarray data since the default decode is directly the gene value.
values	The specific initial values. If value is not specified, getValues function is ran to obtain initial values.
...	Other user named values to include in the object.

Class

Package: galgo
Class Chromosome

```
Object
~~|
~~+--Chromosome
```

Directly known subclasses:

public static class **Chromosome**
extends Object

Fields and Methods

Methods:

as.double	Converts the chromosome values (genes) to its numerical representation.
clone	Clones itself and its genes.
decode	Converts the gene values to user-readable values.
generateRandom	Generates random values for all genes in the chromosome.
genes	Converts the genes values to a numeric vector.
length	Gets the number of genes defined in the chromosome.

mutate	Mutates a chromosome in specific positions.
newCollection	Generates a list of chromosomes cloning the original chromosome object.
newRandomCollection	Creates a list of cloned chromosomes object with its internal values generated by random.
print	Prints the representation of the chromosome object.
reInit	Erases all internal values in order to re-use the object.
summary	Prints the representation of the chromosome object and all its genes.

Methods inherited from Object:
as.list, unObject, $, $<-, [[, [[<-, as.character, attach, clone, detach, equals, extend, finalize, getFields, getInstanciationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, save

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K. http://www.bip.bham.ac.uk/bioinf

References

Goldberg, David E. 1989 *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Pub. Co. ISBN: 0201157675

See Also

`Gene`. `Niche`. `World`. `Galgo`. `BigBang`.

Examples

```
cr <- Chromosome(genes=newCollection(Gene(shape1=1, shape2=100),5))
cr
```

---

classPrediction *Function used to predict class evaluating a fitness function in many train-test sets*

Description

Function used to predict class evaluating a fitness function in many train-test sets.

Usage

```
classPrediction(chr, parent, splits = 1:length(parent$data$splitTrain), set = parer
```

Arguments

chr	Chromosome or vector object.
parent	Parent object, commonly BigBang object.
splits	Which sets of splits will be used to compute the fitness function. Default to all splits defined in `parent$data$splitTrain`.
set	Weigths used in training and test sets. Vector of two values. The first is the weight of train error. The second is the weight of test error. The default value is taken from parent$data$testErrorWeights whose default is c(0,1) (considering only test error).

mode — The type of value to return. "sum" returns a matrix with the number of times a sample (rows) has been predicted as any class (columns). The values are proportional to train and test weights. "probability" convertion of "sum" to probabilities dividing each row by its sum. "class" returns a vector of the predicted class given by majority vote.

Value

A matrix or vector depending on mode paramater.

Note

This function is designed to be used in forwardSelectionModels

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K.

See Also

forwardSelectionModels.BigBang, modelSelection, fitness, configBB.VarSel

---

configBB.VarSel *Creates and configure all objects needed for a "variable selection for classificacion" problem*

Description

Creates and configure all objects needed for a "variable selection for classificacion" problem. It configures Gene, Chromosome, Niche, World, Galgo and BigBang objects.

Usage

```
configBB.VarSel(
file=NULL,
data=NULL,
classes=NULL,
train=rep(2/3,333),
test=1-train,
force.train=c(),
force.test=c(),
train.cases=FALSE,
main="project",
classification.method=c("knn","mlhd","svm","nearcent","rpart","nnet","ranforest","u
classification.test.error=c(0,1),
classification.train.error=c("kfolds","splits","loocv","resubstitution"),
classification.train.Ksets=-1,
classification.train.splitFactor=2/3,
classification.rutines=c("C","R"),
classification.userFitnessFunc=NULL,
scale=(classification.method[1] %in% c("knn","nearcent","mlhd","svm")),
knn.k=3,
```

```
knn.l=1,
knn.distance=c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowsk
nearcent.method=c("mean","median"),
svm.kernel=c("radial","polynomial","linear","sigmoid"),
svm.type=c("C-classification", "nu-classification", "one-classification"),
svm.nu=0.5,
svm.degree=4,
svm.cost=1,
nnet.size=2,
nnet.decay=5e-4,
nnet.skip=TRUE,
nnet.rang=0.1,
geneFunc=runifInt,
chromosomeSize=5,
populationSize=-1,
niches=1,
worlds=1,
immigration=c(rep(0,18),.5,1),
mutationsFunc=function(ni) length(ni),
crossoverFunc=function(ni) round(length(ni)/2,0),
crossoverPoints=round(chromosomeSize/2,0),
offspringScaleFactor=1,
offspringMeanFactor=0.85,
offspringPowerFactor=2,
elitism=c(rep(1,9),.5),
goalFitness=0.90,
galgoVerbose=20,
maxGenerations=200,
minGenerations=10,
galgoUserData=NULL,
maxBigBangs=1000,
maxSolutions=1000,
onlySolutions=FALSE,
collectMode="bigbang",
bigbangVerbose=1,
saveFile="?.Rdata",
saveFrequency=50,
saveVariable="bigbang",
callBackFuncGALGO=function(...) 1,
callBackFuncBB=plot,
callEnhancerFunc=function(chr, parent) NULL,
saveGeneBreaks=NULL,
geneNames=NULL,
sampleNames=NULL,
bigbangUserData=NULL
)
```

Arguments

`file`	The file containing the data. First row should be sample names. First column should be variable names (genes). Second row must be the class for every sample if `classes` is not provided.

`data` If a file is not provided, `data` is the a data matrix or data frame with samples in columns and genes in rows (with its respective colnames and rownames set). If `data` is provided, `class` must be specified.

`classes` if a file is not provided, specifies the classes for the data. If the `file` is provided and classes is specified, the second row of the file is considered as data.

`train` A vector of the proportion of random samples to be used as training sets. The number of sets is determined by the length of `train`. The `train+test` should never be greather than 1. All sets are randomly chosen with the same proportion of samples per class than the original sample set.

`test` A vector of the proportion of random samples to be used as testing sets. The number of sets is determined by the length of `train`. All sets are randomly chosen with the same proportion of samples per class than the original sample set.

`force.train` A vector with sample indexes forced to be part of all training sets.

`force.test` A vector with sample indexes forced to be part of all test sets.

`train.cases` If TRUE, the same number of cases for each class. If numeric vector, then it is interpreted as the number of samples in training per class

`main` A string or ID related to your project that will be used in all plots and would help you to distinguish results from different studies.

`classification.method`
The method to be used for classification. The current available methods (in this package) are `"knn", "mlhd", "svm", "nearcent" (nearest centroid), "rpart"`

`classification.test.error`
Vector of two weights specifing how the fitness function is evaluated to compute the test error. The first value is the weight of training and the second the weight of test. The default is c(0,1) which consider only test error. The sum of this values should be 1.

`classification.train.error`
Specify how the training set is divided to compute the error in the training set (in `evolve` method for `Galgo` object). The fitness function really compute `1-error` where `error` is always computed from the proportion of samples that has been incorrectly classified. `"kfolds"` (k-fold-cross-validation) compute `K` non overlapping sets (`classification.train.Ksets`) attempting to conserve class proportions. `"splits"` compute `K` (`classification.train.Ksets`) random splits. `"loocv"` (leave-one-out-cross-validation) compute `K=training samples`. `"resubstitution"` no folding at all; it is faster and provided for quick overviews.

`classification.train.Ksets`
The number of training set folds/splits. Negative means automatic detection (n=samples, max(min(round(13-n/11),n),3)).

`classification.train.splitFactor`
When `classification.train.error=="splits"`, specifies the proportion of samples used in spliting the training set.

`classification.rutines`
For most of the methods, `R` and `C` code has been provided. `C` code is preferred for performance reason, however finding mistakes is easier in R. Besides, the example code could be used as a guide for new user fitness functions. `"rpart"` has not `C` code. `"svm"` has only some improvments removing redundancy checks.

`classification.userFitnessFunc`
: For `classification.method == "user"`, specify the function that would be used to compute the accuracy and class prediction. The required prototype is `function(chr, parent, tr, te, result)` where `chr` is the chromosome to be evaluated, a convertion using `as.numeric` is commonly needed to extract the exact values from the chromosome. `parent` would be the `BigBang` object where all their variables are exposed. The fitness function commonly use `parent$data$data`, which has been trasposed. `tr` is the vector of samples (rows) that MUST be used as training and `te` the samples that must be used as test. They can correspond to training and test in the evolution or in any other context (as the computation of the confusion matrix or the forward selection). The fitness function should return the result in two different formats, which is specified in the `result` parameter. `result` is `0` (zero) when the predicted class for the test is required (as an integer, not as a factor) otherwise the it is expected the number of correctly classified samples from the test vector.

`scale`
: `TRUE` instruct to scale all rows for zero mean and unitary variance. By default, scale is `TRUE` when `classification.method` is "knn","nearcent","mlhd", or "svm".

`knn.k`
: For KNN method, `knn.k` is the number of nearest neighbours to consider.

`knn.l`
: For KNN method, `knn.l` is the number of minimum neighbours needed to predict a class.

`knn.distance`
: The distance to be used in KNN method. Possible values are `"euclidean", "maximum", "manha` (see `dist` method).

`nearcent.method`
: For nearest centroid method, `nearcent.method` specify the method for computing the centroid (`"mean", "median"`).

`nnet.size`
: Parameter passed to nnet.

`nnet.decay`
: Parameter passed to nnet.

`nnet.skip`
: Parameter passed to nnet.

`nnet.rang`
: Parameter passed to nnet.

`svm.kernel`
: For SVM (support vector machines) method, specify the kernel method `"radial","polynomial","` (see `svm` method in `e1071` package).

`svm.type`
: For SVM method, specify the type of classificacion.

`svm.nu`
: For SVM method and `nu-classification` specify the `nu` value.

`svm.degree`
: For SVM method and `polynomial` kernel, specify the degreee value.

`svm.cost`
: For SVM method, specify the `C` value (cost).

`nnet.`
: Parameters for neural networks classification. See `nnet` package.

`geneFunc`
: The function that provides random values for genes. The default is runifInt, which generates a random integer value with a uniform distribution.

`chromosomeSize`
: Specify the chromosome size (the number of variables/genes to be included in a model). Defaults to 5. See `Gene` and `Chromosome` objects.

`populationSize`
: Specify the number of chromosomes per niche. Defaults is min(20,20+(2000-nrow(data))/400). See `Chromosome` and `Niche` objects.

`niches`
: Specify the number of niches. Defaults to 2. See `Niche`, `World` and `Galgo` objects.

`worlds` Specify the number of worlds. Defaults to 1. See `World` and `Galgo` objects.

`immigration` Specify the migration criteria.

`mutationsFunc`
: Specify the function that returns the number of mutations to perform in the population.

`crossoverFunc`
: Specify the function that returns the number of crossover to perform. The default is the length of the niche divided by 2.

`crossoverPoints`
: Specify the active positions for crossover operator. Defaults to a single point in the middle of the chromosome. See `Niche` object.

`offspringScaleFactor`
: Scale factor for offspring generation. Defaults 1. See `Niche` object.

`offspringMeanFactor`
: Mean factor for offspring generation. Defaults to 0.85. See `Niche` object.

`offspringPowerFactor`
: Power factor for offspring generation. Defaults to 2. See `Niche` object.

`elitism` Elitism probability/flag/vector. Defaults to c(1,1,1,1,1,1,1,1,1,0.5) (elitism present for 9 generations followed by a 50% chance, then repeated). See `Niche` object.

`goalFitness` Specify the desired fitness value (fraction of correct classification). Defaults to 0.90. See `Galgo` object.

`galgoVerbose` `verbose` parameter for `Galgo` object.

`maxGenerations`
: Maximum number of generations. Defaults to 200. See `Galgo` object.

`minGenerations`
: Minimum number of generations. Defaults to 10. See `Galgo` object.

`galgoUserData`
: Additional user data for the `Galgo` object. See `Galgo` object.

`maxBigBangs` Maximum number of bigbang cycles. Defaults to 1000. See `BigBang` object.

`maxSolutions` Maximum number of solutions collected. Defaults to 1000. See `BigBang` object.

`onlySolutions`
: Save only when a solution is reach. Defaults to FALSE (to use all the information, then a filter can be used afterwards). See `BigBang` object.

`collectMode` information to collect. Defaults to `"bigbang"`. See `BigBang` object.

`bigbangVerbose`
: Verbose flag for `BigBang` object. Defaults to 1. See `BigBang` object.

`saveFile` File name where the data is saved. Defaults to `NULL` which implies the name is a concatenation of `classification.method`, method specific parameters, `file` and `".Rdata"`. See `BigBang` object.

`saveFrequency`
: How often the "current" solutions are saved. Defaults to 50. See `BigBang` object.

`saveVariable` Internal `R` variable name of the saved file. Defaults to "bigbang". See `BigBang` object.

`callBackFuncGALGO`
: `callBackFunc` for `Galgo` object. See `Galgo` object.

`callBackFuncBB`
: `callBackFunc` for `BigBang` object. See `BigBang` object.

`callEnhancerFunc`
: `callEnhancerFunc` for `BigBang` object. See `BigBang` object.

`saveGeneBreaks`
: `saveGeneBreaks` vector for `BigBang` object. Defaults to `NULL` which means to be computed automatically (recommended). See `BigBang` object.

`geneNames`
: The gene (variable) names if they differ from the first column in `file` or `rownames(data)`.

`sampleNames`
: The sample names if they differ from first row in `file` or `colnames(data)`.

`bigbangUserData`
: Additional user data for `BigBang` object (stored in `data` variable in `BigBang` object returned).

Details

Wrapper function. Configure all objects from parameters.

Value

A ready to use bigbang object.

*** TO DO: EXPLAIN THE STRUCTURE OF "DATA" ***

Author(s)

Victor Trevino

See Also

`BigBang`.

Examples

```
## Not run:
bb <- configBB.VarSel(...)
bb
blast(bb)

## End(Not run)
```

---

configBB.VarSelMisc

Creates and configure all objects needed for a "variable selection" problem

---

Description

Creates and configure all objects needed for a "variable selection" problem. It configures Gene, Chromosome, Niche, World, Galgo and BigBang objects.

Usage

```
configBB.VarSelMisc(
file=NULL,
data=NULL,
strata=NULL,
train=rep(2/3,333),
test=1-train,
force.train=c(),
force.test=c(),

main="project",

test.error=c(0,1),
train.error=c("kfolds","splits","loocv","resubstitution"),
train.Ksets=-1, # -1 : automatic detection : max(min(round(13-n/11),n),3) n=samples
train.splitFactor=2/3,
fitnessFunc=NULL,

scale=FALSE,

geneFunc=runifInt,
chromosomeSize=5,
populationSize=-1,
niches=1,
worlds=1,
immigration=c(rep(0,18),.5,1),
  mutationsFunc=function(ni) length(ni),
crossoverFunc=function(ni) round(length(ni)/2,0),
crossoverPoints=round(chromosomeSize/2,0),
offspringScaleFactor=1,
offspringMeanFactor=0.85,
offspringPowerFactor=2,
elitism=c(rep(1,9),.5),
goalFitness=0.90,
galgoVerbose=20,
maxGenerations=200,
minGenerations=10,
galgoUserData=NULL, # additional user data for galgo

maxBigBangs=1000,
maxSolutions=1000,
onlySolutions=FALSE,
collectMode="bigbang",
bigbangVerbose=1,
saveFile="?.Rdata",
saveFrequency=50,
saveVariable="bigbang",
callBackFuncGALGO=function(...) 1,
callBackFuncBB=plot,
callEnhancerFunc=function(chr, parent) NULL,
saveGeneBreaks=NULL,
geneNames=NULL,
```

```
sampleNames=NULL,
bigbangUserData=NULL # additional user data for bigbang
)
```

Arguments

file	The file containing the data. First row should be sample names. First column should be variable names (genes). Second row must be the class or strata for every sample if strata is not provided. The strata is used to balance the train-test sets relative to different strata. If there are only one strata, use the same value for all samples.
data	If a file is not provided, data is the a data matrix or data frame with samples in columns and genes in rows (with its respective colnames and rownames set). If data is provided, strata must be specified.
strata	if a file is not provided, specifies the classes or strata of the data. If the file is provided and strata is specified, the second row of the file is considered as data. The strata is used to balance the train-test sets relative to different strata. If there are only one strata, use the same value for all samples.
train	A vector of the proportion of random samples to be used as training sets. The number of sets is determined by the length of train. The train+test should never be greather than 1. All sets are randomly chosen with the same proportion of samples per class than the original sample set.
test	A vector of the proportion of random samples to be used as testing sets. The number of sets is determined by the length of train. All sets are randomly chosen with the same proportion of samples per class than the original sample set.
force.train	A vector with sample indexes forced to be part of all training sets.
force.test	A vector with sample indexes forced to be part of all test sets.
main	A string or ID related to your project that will be used in all plots and would help you to distinguish results from different studies.
test.error	Vector of two weights specifing how the fitness function is evaluated to compute the test error. The first value is the weight of training and the second the weight of test. The default is c(0,1) which consider only test error. The sum of this values should be 1.
train.error	Specify how the training set is divided to compute the error in the training set (in evolve method for Galgo object). "splits" compute K (train.Ksets) random splits. "loocv" (leave-one-out-cross-validation) compute K=training samples. "resubstitution" no folding at all; it is faster and provided for quick overviews.
train.Ksets	The number of training set folds/splits. Negative means automatic detection (n=samples, max(min(round(13-n/11),n),3)).
train.splitFactor	When train.error=="splits", specifies the proportion of samples used in spliting the training set.
fitnessFunc	Specify the function that would be used to compute the accuracy. The required prototype is function(chr, parent, tr, te, result) where chr is the chromosome to be evaluated. parent would be the BigBang object where all their variables are exposed. The fitness function commonly use parent$data$data, which has been trasposed. tr is the vector of samples

(rows) that MUST be used as training and `te` the samples that must be used as test.

`scale` `TRUE` instruct to scale all rows for zero mean and unitary variance. By default, this value is FALSE.

`geneFunc` Specify the function that mutate genes. The default is using an integer uniform distribution function (runifInt).

`chromosomeSize`
Specify the chromosome size (the number of variables/genes to be included in a model). Defaults to 5. See `Gene` and `Chromosome` objects.

`populationSize`
Specify the number of chromosomes per niche. Defaults is min(20,20+(2000-nrow(data))/400). See `Chromosome` and `Niche` objects.

`niches` Specify the number of niches. Defaults to 2. See `Niche`, `World` and `Galgo` objects.

`worlds` Specify the number of worlds. Defaults to 1. See `World` and `Galgo` objects.

`immigration` Specify the migration criteria.

`mutationsFunc`
Specify the function that returns the number of mutations to perform in the population.

`crossoverFunc`
Specify the function that returns the number of crossover to perform. The default is the length of the niche divided by 2.

`crossoverPoints`
Specify the active positions for crossover operator. Defaults to a single point in the middle of the chromosome. See `Niche` object.

`offspringScaleFactor`
Scale factor for offspring generation. Defaults 1. See `Niche` object.

`offspringMeanFactor`
Mean factor for offspring generation. Defaults to 0.85. See `Niche` object.

`offspringPowerFactor`
Power factor for offspring generation. Defaults to 2. See `Niche` object.

`elitism` Elitism probability/flag/vector. Defaults to c(1,1,1,1,1,1,1,1,1,0.5) (elitism present for 9 generations followed by a 50% chance, then repeated). See `Niche` object.

`goalFitness` Specify the desired fitness value (fraction of correct classification). Defaults to 0.90. See `Galgo` object.

`galgoVerbose` `verbose` parameter for `Galgo` object.

`maxGenerations`
Maximum number of generations. Defaults to 200. See `Galgo` object.

`minGenerations`
Minimum number of generations. Defaults to 10. See `Galgo` object.

`galgoUserData`
Additional user data for the `Galgo` object. See `Galgo` object.

`maxBigBangs` Maximum number of bigbang cycles. Defaults to 1000. See `BigBang` object.

`maxSolutions` Maximum number of solutions collected. Defaults to 1000. See `BigBang` object.

`onlySolutions`
Save only when a solution is reach. Defaults to FALSE (to use all the information, then a filter can be used afterwards). See `BigBang` object.

collectMode information to collect. Defaults to "bigbang". See BigBang object.

bigbangVerbose
Verbose flag for BigBang object. Defaults to 1. See BigBang object.

saveFile File name where the data is saved. Defaults to NULL which implies the name is a concatenation of classification.method, method specific parameters, file and ".Rdata". See BigBang object.

saveFrequency
How often the "current" solutions are saved. Defaults to 50. See BigBang object.

saveVariable Internal R variable name of the saved file. Defaults to "bigbang". See BigBang object.

callBackFuncGALGO
callBackFunc for Galgo object. See Galgo object.

callBackFuncBB
callBackFunc for BigBang object. See BigBang object.

callEnhancerFunc
callEnhancerFunc for BigBang object. See BigBang object.

saveGeneBreaks
saveGeneBreaks vector for BigBang object. Defaults to NULL which means to be computed automatically (recommended). See BigBang object.

geneNames The gene (variable) names if they differ from the first column in file or rownames(data).

sampleNames The sample names if they differ from first row in file or colnames(data).

bigbangUserData
Additional user data for BigBang object (stored in $data variable in BigBang object returned).

Details

Wrapper function. Configure all objects from parameters.

Value

A ready to use bigbang object.

*** TO DO: EXPLAIN THE STRUCTURE OF "DATA" ***

Author(s)

Victor Trevino

See Also

BigBang.

Examples

```
## Not run:
bb <- configBB.VarSelMisc(...)
bb
blast(bb)

## End(Not run)
```

fitness *Function used to evaluate a chromosome*

Description

Function used to evaluate a chromosome using training and validation sets (second-level training-test sets) and the selected split.

Usage

```
fitness(chr, parent)
```

Arguments

chr	Chromosome or vector object.
parent	Parent object, commonly BigBang object.

Value

A numeric value with the fitness computed for the chromosome.

Note

This function is designed to be used under configBB.VarSel configuration and depend on splitTrainKFold and splitValidKFold variables.

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K.

See Also

forwardSelectionModels.BigBang, modelSelection, classPrediction, configBB.VarSel

Galgo *The representation of a Genetic Algorithm*

Description

Represents a genetic algorithm (GA) itself. The basic GA uses at least one population of chromosomes, a "fitness" function, and a stopping rule (see references).

The Galgo object is not limited to a single population, it implements a list of populations where any element in the list can be either a Niche object or a World object. Nervertheless, any user-defined object that implements evolve, progeny, best, max, bestFitness, and maxFitness methods can be part of the populations list.

The "fitness" function is by far the most important part of a GA, it evaluates a Chromosome to determine how good the chromosome is respect to a given goal. The function can be sensitive to data stored in .GlobalEnv or any other object (see *evaluate() for further details). For this package and in the case of the microarray, we have included several fitness functions to classify samples

using different methods. However, it is not limited for a classification problem for microarray data, because you can create any fitness function in any given context.

The stopping rule has three options. First, it is simply a desired fitness value implemented as a numeric `fitnessGoal`, and If the maximum fitness value of a population is equal or higher than `fitnessGoal` the GA ends. Second, `maxGenerations` determine the maximum number of generations a GA can evolve. The current generation is increased after evaluating the fitness function to the entire population list. Thus, if the current generation reach `maxGenerations` the GA stops. Third, if the result of the user-defined `callBackFunc` is `NA` the GA stops. In addition, you can always break any R program using `Ctrl-C` (or `Esc` in Windows).

When the GA ends many values are used for futher analysis. Examples are the best chromosome (`best` method), its fitness (`bestFitness` method), the final generation (`generation` variable), the evolution of the maximum fitness (`maxFitnesses` list variable), the maximum chromosome in each generation (`maxChromosome` list variable), and the elapsed time (`elapsedTime` variable). Moreover, flags like `goalScored`, `userCancelled`, and `running` are available.

Usage

```
Galgo(id=0,
populations=list(),
fitnessFunc=function(...) 1,
goalFitness=0.9,
minGenerations=1,
maxGenerations=100,
addGenerations=0,
verbose=20,
callBackFunc=function(...) 1,
data=NULL,
gcCall=0,
savePopulations=FALSE,
maxFitnesses=c(),
maxFitness=0,
maxChromosomes=list(),
maxChromosome=NULL,
bestFitness=0,
bestChromosome=NULL,
savedPopulations=list(),
generation=0,
elapsedTime=0,
initialTime=0,
userCancelled=FALSE,
goalScored=FALSE,
running=FALSE,
...)
```

Arguments

`id`	A way to identify the object.
`populations`	A list of populations of any class `World`, `Niche`, or user-defined population.
`fitnessFunc`	The function that will be evaluate any chromosome in the populations. This function should receive two parameteres, the `Chromosome` object and the `parent` object (defined as a parameter as well). The `parent` object is commonly a object of class `BigBang` when used combined. Theoretically, the fitness function

may return a numeric non-negative finite value, but commonly in practice these values are limited from 0 to 1. The offspring factors in class Niche where established using the 0−1 range assumption.

goalFitness The desired fitness. The GA will evolve until it reach this value or any other stopping rule is met. See description section.

minGenerations
The minimum number of generations. A GA evolution will not ends before this generation number even that fitnessGoal has been reach.

maxGenerations
The maximum number of generations that the GA could evolve.

addGenerations
The number of generations to over-evolve once that goalFitness has been met. Some solutions reach the goal from a large "jump" (or quasi-random mutation) and some other from "plateau". addGenerations helps to ensure the solutions has been "matured" at least that number of generations.

verbose Instruct the GA to display the general information about the evolution. When verbose==1 this information is printed every generation. In general every verbose number of generation would produce a line of output. Of course if verbose==0 would not display a thing at all.

callBackFunc A user-function to be called after every generation. It should receive the Galgo object itself. If the result is NA the GA ends. For instance, if callBackFunc is plot the trace of all generations is nicely viewed in a plot; however, in long runs it can consume time and memory.

data Any user-data can be stored in this variable (but it is not limited to data, the user can insert any other like myData, mama.mia or whatever in the ... argument).

gcCall How often 10 calls to garbage collection function gc(). This sometimes helps for memory issues.

savePopulations
If TRUE, it save the population array in a savedPopulations variable of the galgo object.

maxFitnesses Internal object included for generality not inteded for final users.

maxFitness Internal object included for generality not inteded for final users.

maxChromosomes
Internal object included for generality not inteded for final users.

maxChromosome
Internal object included for generality not inteded for final users.

bestFitness Internal object included for generality not inteded for final users.

bestChromosome
Internal object included for generality not inteded for final users.

savedPopulations
Internal object included for generality not inteded for final users.

generation Internal object included for generality not inteded for final users.

elapsedTime Internal object included for generality not inteded for final users.

initialTime Internal object included for generality not inteded for final users.

userCancelled
Internal object included for generality not inteded for final users.

goalScored	Internal object included for generality not inteded for final users.
running	Internal object included for generality not inteded for final users.
...	Other user named values to include in the object (like pMutation, pCrossover or any other).

Class

Package: galgo
Class Galgo

```
Object
~~|
~~+--Galgo
```

Directly known subclasses:

public static class **Galgo**
extends Object

Fields and Methods

Methods:

best	Returns the best chromosome.
bestFitness	Returns the fitness of the best chromosome.
clone	Clones itself and all its objects.
evaluate	Evaluates all chromosomes with a fitness function.
evolve	Evolves the chromosomes populations of a Galgo (Genetic Algorithm).
generateRandom	Generates random values for all populations in the Galgo object.
length	Gets the number of populations defined in the Galgo object.
max	Returns the chromosome whose current fitness is maximum.
maxFitness	Returns the fitness of the maximum chromosome.
plot	Plots information about the Galgo object.
print	Prints the representation of a Galgo object.
refreshStats	Updates the internal values from the current populations.
reInit	Erases all internal values in order to re-use the object.
summary	Prints the representation and statistics of the galgo object.

Methods inherited from Object:
as.list, unObject, $, $<-, [[, [[<-, as.character, attach, clone, detach, equals, extend, finalize, getFields, getInstanciationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, save

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K. http://www.bip.bham.ac.uk/bioinf

References

Goldberg, David E. 1989 *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Pub. Co. ISBN: 0201157675

See Also

`Gene`, `Chromosome`, `Niche`, `World`, `BigBang`, `configBB.VarSel()`, `configBB.VarSelMisc()`.

Examples

```
cr <- Chromosome(genes=newCollection(Gene(shape1=1, shape2=100),5))
ni <- Niche(chromosomes = newRandomCollection(cr, 10))
wo <- World(niches=newRandomCollection(ni,2))
ga <- Galgo(populations=list(wo), goalFitness = 0.75, callBackFunc=plot,
            fitnessFunc=function(chr, parent) 5/sd(as.numeric(chr)))
ga
evolve(ga)

# missing a classification example
```

`galgo.dist` *Computes the distance in GALGO for KNN based methods*

Description

KNN function does not include other common distances. This function includes more distances computations.

Usage

```
galgo.dist(x, method, p = 2)
```

Arguments

`x`	Matrix to compute distnaces
`method`	Any of `"euclidean", "maximum", "manhattan", "canberra", "binary", "minko` See `dist` function.
`p`	Minkowski power.

Value

A vector class dist.

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K.

See Also

`dist`

Gene *The representation of a gene in a chromosome for genetic algorithms*

Description

Represents the behaviour of a gene in a chromosome for the genetic algorithm. The default properties are supposed to be used in the variable selection problem for microarray data. However, they can be used for any other problem. In addition, any other wanted variable can be added.

See references for Genetic Algorithms.

Usage

```
Gene(id=0, shape1=0, shape2=0, generateFunc=runifInt, ...)
```

Arguments

`id`	To identify the object.
`shape1`	Parameter for a distribution. Used to generate a random value for a gene (mean, minimum, alfa, etc).
`shape2`	Parameter for a distribution. Used to generate a random value for a gene (sd, maximum, beta, etc).
`generateFunc`	Function that generate a random value for a gene using the above shape parameters. This function would be used to get an initial value and to mutate a gene. The default is a random uniform integer with shape1 as minimum and shape2 as maximum (either inclusive). The parameters used in the call are object, n, shape1, and shape2. The random value generated is not saved. If future values depends on the previous, you must save it explicitly in the object.
`...`	Other user named values to include in the object.

Class

Package: galgo
Class Gene

```
Object
~~|
~~+--Gene
```

Directly known subclasses:

public static class **Gene**
extends Object

Fields and Methods

Methods:

as.double	Converts the gene parameters (shape1, shape2) to its numerical representation.
as.matrix	Converts the gene parameters (shape1, shape2) to matrix.
generateRandom	Generates a random value from the defined function.
mutate	Mutates a gene.
newCollection	Generates a list of cloned objects.
newRandomCollection	Generates a list of cloned objects and random values.
print	Prints the representation of a gene object.
reInit	Erases all internal values in order to re-use the object.
summary	Prints the representation of a gene object.

Methods inherited from Object:
as.list, unObject, $, $<-, [[, [[<-, as.character, attach, clone, detach, equals, extend, finalize, getFields, getInstanciationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, save

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K. http://www.bip.bham.ac.uk/bioinf

References

Goldberg, David E. 1989 *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Pub. Co. ISBN: 0201157675

See Also

Chromosome. Niche. World. Galgo. BigBang. runifInt.

Examples

```
ge <- Gene(shape1=1, shape2=100)
ge
```

---

geneBackwardElimination

Searches for shorter or better models using backward elimination strategy

---

Description

Searches for shorter or better models using backward elimination strategy. Recursively eliminates variables/genes from a chromosome one by one computing the fitness function. This function is specially designed to be used in the BigBang object and for variable selection problems.

Usage

```
geneBackwardElimination(chr, bigbang, result=c("highest","shortest", "selected", "v
```

Arguments

chr	Original chromosome object (or numeric vector).
bigbang	The BigBang object to be used to call the fitness function.
result	The type of result needed. "highest" returns the visited chromosome whose fitness was highest. Ties are resolved using the shortest chromosome and finally by random. "shortest" returns the visited chromosome whose length was minimum and fitness greather than or equal to the original. Ties are resolved by highest fitness and finally by random. "visited" returns a list of all visited chromosomes. "selected" only the chromosomes with fitness greather than or equal to original fitness.
minChromosomeSize	The minimum possible size of a chromosome. The default is 2.
fitnessFunc	The fitness function used to evaluate the chromosomes. The default is the usage of bigbang$galgo$fitnessFunc.
fitnessAid	To avoid local minima, fitnessAid is an amount to be reduced to original fitness in order to try search for better fitness. When it is negative, it is interpreted as percentage value to reduce from the original fitness. If fitnessAid is positive, it is substracted from original fitness.
verbose	Display internal steps for debugging purposes.
...	Additional arguments to fitnessFunc.

Details

Removes one gene/variable at the time and compute the fitness. If the fitness is greather than or equal to original "reduced" fitness, another attempt to remove other variable will be performed. The result might be a reduced chromosome with same or better fitness.

Value

A chromosome when result=="highest" or result=="smallest" and a data frame otherwise.

Author(s)

Victor Trevino

See Also

BigBang, robustGeneBackwardElimination.

Examples

```
## Not run:
rchr <- lapply(bb$bestChromosomes[1:100],geneBackwardElimination, bb, result="shortest")
barplot(table(unlist(lapply(rchr,length))),main="Length of Shortened Chromosomes (evaluated

rchr <- lapply(bb$bestChromosomes[1:100],robustGeneBackwardElimination, bb, result="shortest
barplot(table(unlist(lapply(rchr,length))),main="Length of Shortened Chromosomes")

## End(Not run)
```

generateRandomModels *Generates Random shorter models*

Description

Evaluate random models using the specified gene indexes.

Usage

```
generateRandomModels(genes,  bigbang,
size=trunc(length(genes)/2), n=100,
fitnessFunc=bigbang$data$modelSelectionFunc,
models=FALSE, ...)
```

Arguments

genes	Original chromosome object (or numeric vector).
bigbang	The BigBang object to be used to call the fitness function.
size	Size of new random models.
n	Number of models
fitnessFunc	The fitness function used to evaluate the chromosomes. The default is the usage of bigbang$galgo$fitnessFunc.
models	Logical value.
...	Other parameters passed to fitnessFunc.

Value

If models==TRUE, a vector of resulted fitness for random models, otherwise a list with models (matrix, cols=models) and their fitness (vector) is returned.

Author(s)

Victor Trevino

See Also

BigBang.

Examples

```
## Not run:
rm <- generateRandomModels(geneFrequency(bb,value="index")[1:50],bb,size=5,n=100,models=T)
rm

## End(Not run)
```

knn_C_predict *Class prediction using KNN method calling the C code*

Description

C code for knn. There is a knn_R_predict but R code is slower, which is included for debugging and educational purposes.

Usage

```
knn_C_predict(chr, parent, tr, te, result)
```

Arguments

chr	Chromosome. Must be integer, use as.integer().
parent	Bigbang object.
tr	Sample indexes for training vector. Must be integer, use as.integer().
te	Sample indexes for test vector. Must be integer, use as.integer().
result	0 indicates to return class prediction, non-zero returns the proportion of samples with same class prediction. Must be integer, use as.integer().

Value

Vector of classes (integer) or numeric value. Depends on result argument.

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K.

See Also

knn

knn_R_predict *Class prediction using KNN method calling the R code*

Description

R code is slower than C code. This function is included for debugging and educational purposes.

Usage

```
knn_R_predict(chr, parent, tr, te, result)
```

Arguments

chr	Chromosome. Must be integer, use as.integer().
parent	Bigbang object.
tr	Sample indexes for training vector. Must be integer, use as.integer().
te	Sample indexes for test vector. Must be integer, use as.integer().
result	0 indicates to return class prediction, non-zero returns the proportion of samples with same class prediction. Must be integer, use as.integer().

Value

Vector of classes (integer) or numeric value. Depends on result argument.

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K.

See Also

nnet

---

loadObject *Load saved data of class Object and use reObject as necessary*

Description

Load the data from a file into the .GlobalEnv (or any other enviroment). If variables were converted to a list using unObject, this variables are converted to original object using reObject method.

Usage

```
loadObject(file=NULL, envir=.GlobalEnv, verbose=T, reobjectize=T, compatibilize=TRU
```

Arguments

file	The file to load.
envir	The environment to load the data. The default is .GlobalEnv.
verbose	Displays progress.
reobjectize	Specify if reObject method should be called. Defaults to TRUE.
compatibilize	Compatibilze chromosomes built on previous versions.
...	Additional arguments to reObject

Details

Load the data from a file into the .GlobalEnv (or any other enviroment). If variables were converted to a list using unObject, this variables are converted to original object using reObject method.

Value

A data frame with variable names and class of loaded objects.

Warning

It could take some seconds for large and/or complex objects/files.

Author(s)

Victor Trevino

See Also

`unObject`, `reObject`.

Examples

```
library(R.oo)  # needed library
o <- Object()
o$x = 1
o$y = 2
o$x
o$y
o
class(o)
names(o)
uo <- unObject(o)
uo
class(uo)
save(uo, file="uo.Rdata")


### perhaps other session here
library(R.oo)
loadObject("uo.Rdata")
uo
class(uo)
# the class is the original from the original object (o in this case)

### equivalent to:
library(R.oo)
load("uo.Rdata")
uo <- reObject(uo)
uo
class(uo)
```

---

mlhd_C_predict *Class prediction using Maximum Likelihood Discriminant Functions method calling the C code*

Description

C code for mlhd. There is a mlhd_R_predict but R code is slower, which is included for debugging and educational purposes.

Usage

```
mlhd_C_predict(chr, parent, tr, te, result)
```

Arguments

chr	Chromosome. Must be integer, use as.integer().
parent	Bigbang object.
tr	Sample indexes for training vector. Must be integer, use as.integer().
te	Sample indexes for test vector. Must be integer, use as.integer().
result	0 indicates to return class prediction, non-zero returns the proportion of samples with same class prediction. Must be integer, use as.integer().

Value

Vector of classes (integer) or numeric value. Depends on `result` argument.

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K.

---

mlhd_R_predict *Class prediction using Maximum Likelihood Discriminant Functions method calling the R code*

Description

R code is slower than C code. This function is included for debugging and educational purposes.

Usage

```
mlhd_R_predict(chr, parent, tr, te, result)
```

Arguments

chr	Chromosome. Must be integer, use as.integer().
parent	Bigbang object.
tr	Sample indexes for training vector. Must be integer, use as.integer().
te	Sample indexes for test vector. Must be integer, use as.integer().
result	0 indicates to return class prediction, non-zero returns the proportion of samples with same class prediction. Must be integer, use as.integer().

Value

Vector of classes (integer) or numeric value. Depends on `result` argument.

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K.

modelSelection *Function used to evaluate a fitness function in many train-test sets*

Description

Function used to evaluate a fitness function in many train-test sets

Usage

```
modelSelection(chr, parent, splits=1:length(parent$data$splitTrain), set=parent$dat
```

Arguments

chr	Chromosome or vector object.
parent	Parent object, commonly BigBang object.
splits	Which sets of splits will be used to compute the fitness function. Default to all splits defined in `parent$data$splitTrain`.
set	Weigths used in training and test sets. Vector of two values. The first is the weight of train error. The second is the weight of test error. The default value is taken from parent$data$testErrorWeights whose default is c(0,1) (considering only test error).

Value

A vector with the fitness computed for each split weighted according to `set` parameter.

Note

This function is designed to be used in forwardSelectionModels

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K.

See Also

`forwardSelectionModels.BigBang`, `classPrediction`, `fitness`, `configBB.VarSel`

nearcent_C_predict *Class prediction using the nearest centroid method calling the C code*

Description

C code for nearest centroid. There is a nearcent_R_predict but R code is slower, which is included for debugging and educational purposes.

Usage

```
nearcent_C_predict(chr, parent, tr, te, result)
```

Arguments

chr	Chromosome. Must be integer, use as.integer().
parent	Bigbang object.
tr	Sample indexes for training vector. Must be integer, use as.integer().
te	Sample indexes for test vector. Must be integer, use as.integer().
result	0 indicates to return class prediction, non-zero returns the proportion of samples with same class prediction. Must be integer, use as.integer().

Value

Vector of classes (integer) or numeric value. Depends on `result` argument.

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K.

---

nearcent_R_predict *Class prediction using the nearest centroid method calling the R code*

Description

R code is slower than C code. This function is included for debugging and educational purposes.

Usage

```
nearcent_R_predict(chr, parent, tr, te, result)
```

Arguments

chr	Chromosome. Must be integer, use as.integer().
parent	Bigbang object.
tr	Sample indexes for training vector. Must be integer, use as.integer().
te	Sample indexes for test vector. Must be integer, use as.integer().
result	0 indicates to return class prediction, non-zero returns the proportion of samples with same class prediction. Must be integer, use as.integer().

Value

Vector of classes (integer) or numeric value. Depends on `result` argument.

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K.

Niche *The representation of a set of chromosomes for genetic algorithms*

Description

Niche represents a set of chromosomes for the genetic algorithm. The niche can generate a progeny that may be more adapted to certains tasks (or enviroment, see Goldberg). To decide which chromosomes are more suitable to be chosen as "parents", every chromosome in the niche is evaluated using a "fitness" function. The selected chromosomes are mated using crossover to produce diversity. Finally the chromosomes are mutated and the new progeny is ready for next generation.

The basic idea to generate a progeny is a random selection biased toward the best chromosomes (see Goldberg). We implented this idea as a weighted probability for a chromosome to be selected using the formula:

p = scale * max(0,fitness - mean * mean(fitness))\^\ power

where scale, mean and power are the properties of the niche (`offspringScaleFactor, offspringMeanFactor a` respectively). The default values were selected to be reasonably bias when the variance in the fitness are both high (at early generations) and low (in late generatios).

The crossover mechanism needs to know the positions whose chromosomes can actually mate (`crossoverPoints`). The number of crossovers can be customized with `crossoverFunc` (`*crossover()`).

The elitism mechanism (`elitism` variable) are implemented replacing a random chromosome from the niche at the end of the progeny process (`*progeny()`).

The Niche object keeps a record of the number of generations, the maximum chromosome in the niche, and the best chromosome ever known (see `*best()` for an example).

The length of the niche is static. Nevertheless this behaviour (and any other) can be customised overwriting original methods (like progeny or crossover) methods. However, this is intend to be used only for experienced users.

The niche is considered a "closed population", this means mating with chromosomes within the same niche. Migration mechanism uses niches to exchange chromosomes between them, which is implemented in `World` object (see `World`).

Usage

```
Niche(id=0,
chromosomes=list(),
offspringScaleFactor=1,
offspringMeanFactor=0.85,
offspringPowerFactor=2,
crossoverPoints=0,
mutationsFunc=function(.O) length(.O),
crossoverFunc=function(.O) length(.O)/2,
elitism=1,
generation=0,
fitness=0,
maxFitness=0,
bestFitness=0,
maxChromosome=NULL,
bestChromosome=NULL,
...)
```

Arguments

id	A way to identify the object.
chromosomes	A list of defined chromosomes composing the niche.
offspringScaleFactor	The offspringScaleFactor parameter. See description.
offspringMeanFactor	The offspringMeanFactor parameter. See description.
offspringPowerFactor	The offspringPowerFactor parameter. See description.
crossoverPoints	Specific positions at which the chromosomes can be mated. Should be from 2 to *minimum* possible length of any chromosome in the niche.
mutationsFunc	A function returning the final number of mutations in the niche. It receives the Niche object as parameter. To implement "probability of mutation" instead, add a variable like pMutation in the constructor and multiply by the length of the niche and the length of the chromosome in the function (function(niche) niche$pMutation
crossoverFunc	A function returning the final number of crossovers in the niche. It receives the Niche object as parameter. To implement "probability of crossover" instead, add a variable like pCrossOver in the constructor and multiply by the length of the niche in the function. (function(niche) niche$pCrossOver * length(niche)).
elitism	Controls the elitism mechanism. Elitism is desired to find solutions quicker, but it may be a nuisance when it is trapped in strong attractors. Therefore, in general, it may be a probability. Furthermore, it can be a vector of probabilities where the index is controlled by generation. If the current generation is greather than the length of this vector, a cycled version is used (starting from the first value).
fitness	The current fitness. It should be 0 initially, but it is included for generalization.
bestFitness	The best fitness ever visited. It should be 0 initially. Included for generalization.
maxFitness	The maximum fitness from the current chromosomes. It should be 0 initially, but it is included for generalization.
maxChromosome	The chromosome whose fitness is maximum from the current chromosomes. It should be NULL initially, but it is included for generalization.
bestChromosome	The chromosome whose fitness is maximum visited ever. It should be NULL initially, but it is included for generalization.
generation	For internal uses only.
...	Other user named values to include in the object (like pMutation, pCrossover or any other).

Class

Package: galgo
Class Niche

Object
~~|

~~+--Niche

Directly known subclasses:

public static class **Niche**
extends Object

Fields and Methods

Methods:

as.double	Converts the chromosome values (genes) to a vector.
as.matrix	Converts the chromosome values (genes) to a matrix.
best	Returns the best chromosome of the niche.
bestFitness	Returns the fitness of the best chromosome in the niche.
clone	Clones itself and its chromosomes.
crossover	Performs crossover between chromosomes of the niche.
evaluate	Evaluates the chromosome using a fitness function.
generateRandom	Generates random values for all genes contained in all chromosomes in the niche.
getFitness	Returns the fitness vector related to chromosomes.
length	Gets the number of chromosomes defined in the niche.
max	Returns the chromosome in the niche whose current fitness is maximum.
maxFitness	Returns the fitness of the maximum chromosome in the niche.
mutate	Mutates a niche calling mutate method for all chromosomes.
newCollection	Generates a list of cloned niches.
newRandomCollection	Creates a list of cloned niches with its internal values generated by random.
offspring	Overwrites the new niche selecting a new population from the best chromosomes.
plot	Plots information about niche object.
print	Prints the representation of a niche object.
progeny	Performs offspring, crossover, mutation, and elitism mechanism to generate the "evolved" n
refreshStats	Updates the internal values from the current population.
reInit	Erases all internal values in order to re-use the object.
scaling	Assigns a weight for every chromosome to be selected for the next generation.
summary	Prints the representation and statistics of the niche object.

Methods inherited from Object:
as.list, unObject, $, $<-, [[, [[<-, as.character, attach, clone, detach, equals, extend, finalize, getFields, getInstanciationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, save

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K. http://www.bip.bham.ac.uk/bioinf

References

Goldberg, David E. 1989 *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Pub. Co. ISBN: 0201157675

See Also

Gene, Chromosome, World, Galgo, BigBang.

Examples

```
cr <- Chromosome(genes=newCollection(Gene(shape1=1, shape2=100),5))
ni <- Niche(chromosomes=newRandomCollection(cr, 10))
ni

## in average, one of 10 genes can be mutated
mf <- function(niche) niche$pMutations * length(niche) * length(niche$chromosomes[[1]])
ni2 <- Niche(chromosomes=newRandomCollection(cr, 10),
       mutationsFunc=mf,
pMutations=1/10)
ni2    # random initial niche
mutate(ni2) # returns the chromosomes indexes that were mutated
ni2    # mutated niche
```

---

nnet_R_predict *Class prediction using the neural networks method calling the R code*

Description

neural networks R code for prediction.

Usage

```
nnet_R_predict(x, parent, tr, te, result, ...)
```

Arguments

x	Chromosome. Must be integer, use as.integer().
parent	Bigbang object.
tr	Sample indexes for training vector. Must be integer, use as.integer().
te	Sample indexes for test vector. Must be integer, use as.integer().
result	0 indicates to return class prediction, non-zero returns the proportion of samples with same class prediction. Must be integer, use as.integer().
...	Not used. Included for package compatbility documentation purposes.

Value

Vector of classes (integer) or numeric value. Depends on result argument.

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K.

See Also

nnet

randomforest_R_predict

Class prediction using RandomForest method calling the R code

Description

Prediction using random forest from randomForest package.

Usage

```
randomforest_R_predict(chr, parent, tr, te, result)
```

Arguments

chr	Chromosome. Must be integer, use as.integer().
parent	Bigbang object.
tr	Sample indexes for training vector. Must be integer, use as.integer().
te	Sample indexes for test vector. Must be integer, use as.integer().
result	0 indicates to return class prediction, non-zero returns the proportion of samples with same class prediction. Must be integer, use as.integer().

Value

Vector of classes (integer) or numeric value. Depends on `result` argument.

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K.

See Also

`knn`

reObject

Creates proper extended Object from a list obtained by unObject

Description

Rebuilds an object to its original class from a list which was usually obtained using unObject. The original class is deduced using the `Class.` value and its S3 constructor will be called using all other values as properties.

Usage

```
reObject(o, showStructure = 0)
```

Arguments

o	The list attempted to convert to its original Object.
showStructure	Flag to show/debug the conversion course. It can be 1, 2 or 0.

Details

The original class (x$Class. value) is called without any parameter, then all properties (names) in the list are set using assign. The procedure is recursive called if an object of class list is found inside x. If the original object was extended from Object, this object have to be already defined using S3 methodology, otherwise an error would occur.

Value

Object of original class given by x$Class.

Warning

It could take some seconds for large and/or complex objects.

Note

It is very important that if the original class was extendend from Object, this class and its methods are already defined, otherwise unexpected behaviour and/or errors would occur.

Author(s)

Victor Trevino

See Also

unObject.

Examples

```
library(R.oo)  # needed library
o <- Object()
o$x = 1
o$y = 2
o$x
o$y
o
class(o)
names(o)
uo <- unObject(o)
uo
x <- reObject(uo)
class(x)
names(x)
x$x
x$y



### saving/retriving
```

```
library(R.oo)
o <- Object()
o$x = 1
o$y = 2
uo <- unObject(o)
save(uo, file="uo.Rdata")
### perhaps other session here
library(R.oo)
#if your object requiere other sub-class (extend Object) and/or method definition,
#load it here before using reObject otherwise an error would occur.
load("uo.Rdata")
class(uo)             ## uo now is a list
uo
x <- reObject(uo)
class(x)                ### now x is Object
names(x)
x$x
x$y
x
```

`robustGeneBackwardElimination` *Searches for shorter or better models using backward elimination strategy*

Description

Searches for shorter or better models using backward elimination strategy. Recursively eliminates variables/genes from a chromosome one by one computing the fitness function. This function is specially designed to be used in the `BigBang` object and for variable selection problems.

Usage

```
robustGeneBackwardElimination(chr, bigbang, fitnessFunc=bigbang$data$modelSelectior
```

Arguments

`chr`	Original chromosome object (or numeric vector).
`bigbang`	The `BigBang` object to be used to call the fitness function.
`fitnessFunc`	The fitness function used to evaluate the chromosomes. The default is the usage of `bigbang$data$modelSelectionFunc`.
`...`	Additional Arguments passed to geneBackwardElimination.

Details

Removes one gene/variable at the time and compute the fitness. If the fitness is greather than or equal to original "reduced" fitness, another attempt to remove other variable will be performed. The result might be a reduced chromosome with same or better fitness.

Value

A chromosome when `result=="highest" or result=="smallest"` and a data frame otherwise.

Author(s)

Victor Trevino

See Also

`BigBang`, `geneBackwardElimination`.

Examples

```
## Not run:
rchr <- lapply(bb$bestChromosomes[1:100],robustGeneBackwardElimination, bb, result="shortest
barplot(table(unlist(lapply(rchr,length))),main="Length of Shortened Chromosomes")

## End(Not run)
```

---

rpart_R_predict *Class prediction using the recursive tree partitions method calling the R code*

Description

Recursive tree partition code in R.

Usage

```
rpart_R_predict(chr, parent, tr, te, result)
```

Arguments

`chr`	Chromosome. Must be integer, use as.integer().
`parent`	Bigbang object.
`tr`	Sample indexes for training vector. Must be integer, use as.integer().
`te`	Sample indexes for test vector. Must be integer, use as.integer().
`result`	0 indicates to return class prediction, non-zero returns the proportion of samples with same class prediction. Must be integer, use as.integer().

Value

Vector of classes (integer) or numeric value. Depends on `result` argument.

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K.

`runifInt` *Generation of random uniform integer values*

Description

random number of uniform distribution

Usage

```
runifInt(.O, n, mn, mx)
```

Arguments

`.O`	Gene objct
`n`	Number of random values to generate
`mn`	Minimum value
`mx`	Maximum value

Value

A vector with random values drawn from a uniform distribution.

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K.

See Also

`runif`

`svm_C_predict` *Class prediction using support vector machines method calling the C/R code*

Description

This function really calls the C function that is provided in svm package. The only difference with svm function is that many checks are removed in order to speed up the process. However, it is responsability of the user use valid values.

Usage

```
svm_C_predict(x, parent, tr, te, result, ...)
```

Arguments

x	Chromosome. Must be integer, use as.integer().
parent	Bigbang object.
tr	Sample indexes for training vector. Must be integer, use as.integer().
te	Sample indexes for test vector. Must be integer, use as.integer().
result	0 indicates to return class prediction, non-zero returns the proportion of samples with same class prediction. Must be integer, use as.integer().
...	Not used. Included for package compatbility documentation purposes.

Value

Vector of classes (integer) or numeric value. Depends on `result` argument.

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K.

See Also

`svm`

---

svm_R_predict *Class prediction using support vector machines method calling the R code*

Description

This function just call svm R code.

Usage

```
svm_R_predict(x, parent, tr, te, result, ...)
```

Arguments

x	Chromosome. Must be integer, use as.integer().
parent	Bigbang object.
tr	Sample indexes for training vector. Must be integer, use as.integer().
te	Sample indexes for test vector. Must be integer, use as.integer().
result	0 indicates to return class prediction, non-zero returns the proportion of samples with same class prediction. Must be integer, use as.integer().
...	Not used. Included for package compatbility documentation purposes.

Value

Vector of classes (integer) or numeric value. Depends on `result` argument.

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K.

See Also

svm

---

unObject *Converts variables from class Object (and derived classes) to list*

Description

Converts objects derived from class Object to a list object that preserve the properties (data) and can be accessed using the same R syntax. It is primarly used to explore the data or to save the object as an R object independent of the original methods.

Usage

```
unObject(...)
```

Arguments

...	Variables of class Object (or list containing some Object).

Details

In R.oo package, all objects are internally represented as enviroment objects (see R.oo package) to give the "by value" functionality. However, this representation is not suitable to save, retrive or explore the data as easy as common objects in R. This method converts an object derived from class Object to a common list object preserving all data except the original methods. It is very useful when an object of class Object contains other objects derived from same class Object.

Value

Return a list containg all values of the object. If x contains a list or other Object, these are represented also as a list.The original class of the object is stored in "Class." value.

Warning

The CPU time consumed by this method depends on the complexity of x. It is commonly very fast but can be a nuisance when x contains many nested objects of class Object (or many lists containing Objects).

Note

If properties (values) inside an object Object contains a function object, the enviroment is set to .GlobalEnv for convenience. This method is also implemented for list object because it could contain another Object.

Author(s)

Victor Trevino

See Also

Object, reObject, unObject.list.

Examples

```
library(R.oo)  # needed library
o <- Object()
o$x = 1
o$y = 2
o$x
o$y
o
class(o)
names(o)
uo <- unObject(o)
uo
```

World *The representation of a set of niches with migration for genetic algorithms*

Description

Represents a set of nices for the genetic algorithm. Because the niches are "closed populations", it is sometimes needed exchange information bewteen niches (or "islands"). The World object implements the exchange of chromosomes between niches, and to be compatible, it also implements the needed methods than an usual niche but considering the immigration property. Thus, the Galgo object can receive a list of Niches, a list of Worlds, or a list of any mixture of them.

Usage

```
World(id=0,
niches=list(),
immigration=0,
maxFitness=0,
bestFitness=0,
maxChromosome=NULL,
bestChromosome=NULL,
generation=0,
...)
```

Arguments

id	A way to identify the object.
niches	A list of defined niches composing the world. However, it can be a list containing even World objects.
immigration	It can be NULL, a function, or a vector. When it is NULL immigration is disabled. When it is a function it is evaluated using the same World object as parameter, the result should be a numeric value. When the length of immigration is greather than 1 a cycled version is used depending on the generation. If the resulted or selected numeric value is greather than 1 it is

interpreted as the number of chromosomes to migrate, otherwise it is assumed to be a probability to migrate one chromosome. The final `I` best chromosomes to migrate apply to all niches.

`bestFitness` The best fitness ever visited.

`maxFitness` The maximum fitness from the current chromosomes. It should be 0 initially, but it is included for generalization.

`maxChromosome`
The chromosome whose fitness is maximum from the current chromosomes. It should be NULL initially, but it is included for generalization.

`bestChromosome`
The chromosome whose fitness is maximum visited ever. It should be NULL initially, but it is included for generalization.

`generation` For internal use only.

`...` Other user named values to include in the object (like pMutation, pCrossover or any other).

Class

Package: galgo
Class World

```
Object
~~|
~~+--World
```

Directly known subclasses:

public static class **World**
extends Object

Fields and Methods

Methods:

`best`	Returns the best chromosome.
`bestFitness`	Returns the fitness of the best chromosome.
`clone`	Clones itself and its niches.
`evaluate`	Evaluate all niches with a fitness function.
`generateRandom`	Generates random values for all niches in the world.
`length`	Gets the number of niches defined in the world.
`max`	Returns the chromosome whose current fitness is maximum.
`maxFitness`	Returns the fitness of the maximum chromosome.
`newCollection`	Generates a list cloning an object.
`newRandomCollection`	Creates a list of cloned object with its internal values generated by random.
`plot`	Plots information about world object.
`print`	Prints the representation of a world object.
`progeny`	Calls progeny method to all niches in the world object.
`refreshStats`	Updates the internal statistics from the current population.

`reInit`	Erases all internal values in order to re-use the world object.
`summary`	Prints the representation and statistics of the world object.

Methods inherited from Object:
as.list, unObject, $, $<-, [[, [[<-, as.character, attach, clone, detach, equals, extend, finalize, getFields, getInstanciationTime, getStaticInstance, hasField, hashCode, ll, load, objectSize, print, save

Author(s)

Victor Trevino. Francesco Falciani Group. University of Birmingham, U.K. http://www.bip.bham.ac.uk/bioinf

References

Goldberg, David E. 1989 *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Pub. Co. ISBN: 0201157675

See Also

`Gene`, `Chromosome`, `Niche`, `Galgo`, `BigBang`.

Examples

```
cr <- Chromosome(genes=newCollection(Gene(shape1=1, shape2=100),5))
ni <- Niche(chromosomes=newRandomCollection(cr, 10))
wo <- World(niches=newRandomCollection(ni,2))
wo

progeny(wo) # returns the chromosomes indexes that were mutated
```

Index