Package 'GSgalgoR'

December 2, 2025

Type Package

Title An Evolutionary Framework for the Identification and Study of Prognostic Gene Expression Signatures in Cancer

Version 1.20.0

Description A multi-objective optimization algorithm for disease sub-type discovery based on a non-dominated sorting genetic algorithm.
The 'Galgo' framework combines the advantages of clustering algorithms for grouping heterogeneous 'omics' data and the searching properties of
genetic algorithms for feature selection. The algorithm search for the optimal number of clusters determination considering the features that
maximize the survival difference between sub-types while keeping cluster consistency high.

License MIT + file LICENSE

biocViews GeneExpression, Transcription, Clustering, Classification, Survival

Encoding UTF-8

LazyData true

Imports cluster, doParallel, foreach, matchingR, nsga2R, survival, proxy, stats, methods,

Suggests knitr, rmarkdown, ggplot2, BiocStyle, genefu, survcomp, Biobase, survminer, breastCancerTRANSBIG, breastCancerUPP, iC10TrainingData, pamr, testthat

URL https://github.com/harpomaxx/GSgalgoR

BugReports https://github.com/harpomaxx/GSgalgoR/issues

RoxygenNote 7.1.0

VignetteBuilder knitr

git_url https://git.bioconductor.org/packages/GSgalgoR

git_branch RELEASE_3_22

git_last_commit fc892d4

git_last_commit_date 2025-10-29

Repository Bioconductor 3.22

Date/Publication 2025-12-01

Author Martin Guerrero [aut], Carlos Catania [cre]

Maintainer Carlos Catania <harpomaxx@gmail.com>

Contents

`GSgalgoR-package` *GSgalgoR: A bi-objective evolutionary meta-heuristic to identify robust transcriptomic classifiers associated with patient outcome across multiple cancer types.*

Description

This package was developed to provide a simple to use set of functions to use the galgo algorithm. A multi-objective optimization algorithm for disease subtype discovery based on a non-dominated sorting genetic algorithm.

Different statistical and machine learning approaches have long been used to identify gene expression/molecular signatures with prognostic potential in different cancer types. Nonetheless, the molecular classification of tumors is a difficult task and the results obtained via the current statistical methods are highly dependent on the features analyzed, the number of possible tumor subtypes under consideration, and the underlying assumptions made about the data. In addition, some cancer types are still lacking prognostic signatures and/or of subtype-specific predictors which are continually needed to further dissect tumor biology. In order to identify specific molecular phenotypes to develop precision medicine strategies we present Galgo: A multi-objective optimization process based on a non-dominated sorting genetic algorithm that combines the advantages of clustering methods for grouping heterogeneous omics data and the exploratory properties of genetic algorithms (GA) in order to find features that maximize the survival difference between subtypes while keeping high cluster consistency.

Package:	GSgalgoR
Type:	Package
Version:	1.0.0
Date:	2020-05-06

License:	GPL-3
Copyright:	(c) 2020 Martin E. Guerrero-Gimenez.
URL:	https://www.github.com/harpomaxx/galgo
LazyLoad:	yes

Author(s)

Martin E. Guerrero-Gimenez <mguerrero@mendoza-conicet.gob.ar>

Maintainer: Carlos A. Catania <harpomaxx@gmail.com >

`calculate_distance` *Functions to calculate distance matrices using cpu computing*

Description

Functions to calculate distance matrices using cpu computing

Usage

```
calculate_distance_pearson_cpu(x)

calculate_distance_spearman_cpu(x)

calculate_distance_uncentered_cpu(x)

calculate_distance_euclidean_cpu(x)

select_distance(distancetype = "pearson")
```

Arguments

`x`	an expression matrix with features as rows and samples as columns
`distancetype`	a `character` that can be either `'pearson'`, `'uncentered'`, `'spearman'` or `'euclidean'`

Value

`select_distance(distancetype)` assigns global function calculate_distance according to the parameters specified

`calculate_distance_pearson_cpu(x)` returns columnwise pearson distance calculated using the CPU

`calculate_distance_uncentered_cpu(x)` returns columnwise uncentered pearson distance calculated using the CPU

`calculate_distance_spearman_cpu(x)` returns columnwise spearman distance calculated using the CPU

`calculate_distance_euclidean_cpu(x)` returns columnwise euclidean distance calculated using the CPU

Author(s)

Martin E Guerrero-Gimenez, <mguerrero@mendoza-conicet.gob.ar>

Examples

```
# load example dataset
require(iC10TrainingData)
require(pamr)

data(train.Exp)

calculate_distance <- select_distance(distancetype = "pearson")
Dist <- calculate_distance(train.Exp)
k <- 4
Pam <- cluster_algorithm(Dist, k)
table(Pam$cluster)
```

---

callback_base_report *Print basic info per generation*

Description

Print basic info per generation

Usage

```
callback_base_report (userdir, generation, pop_pool,
pareto, prob_matrix, current_time)
```

Arguments

userdir	the default directory used by 'galgo()' to store files
generation	a number indicating the number of iterations of the galgo algorithm
pop_pool	a data.frame with the solution vectors, number of clusters and their ranking.
pareto	the solutions found by Galgo across all generations in the solution space
prob_matrix	a matrix or data.frame. Must be an expression matrix with features in rows and samples in columns
current_time	an POSIXct object

Value

Nothing.

Examples

```
# load example dataset

library(breastCancerTRANSBIG)
data(transbig)
Train <- transbig
rm(transbig)
```

```
expression <- Biobase::exprs(Train)
clinical <- Biobase::pData(Train)
OS <- survival::Surv(time = clinical$t.rfs, event = clinical$e.rfs)
# We will use a reduced dataset for the example
expression <- expression[sample(1:nrow(expression), 100), ]

# Now we scale the expression matrix
expression <- t(scale(t(expression)))

# Run galgo with base_report_callback assigned to the report_callback
# hook-point
GSgalgoR::galgo(generations = 5,
population = 15,
prob_matrix = expression,
OS = OS,
report_callback = callback_base_report
)
```

`callback_base_return_pop` *A base callback function that returns a galgo.Obj*

Description

A base callback function that returns a galgo.Obj

Usage

```
callback_base_return_pop (userdir = "",generation, pop_pool,
pareto, prob_matrix, current_time)
```

Arguments

`userdir`	the default directory used by ‘galgo()‘ to store files
`generation`	a number indicating the number of iterations of the galgo algorithm
`pop_pool`	a `data.frame` with the solution vectors, number of clusters and their ranking.
`pareto`	the solutions found by Galgo across all generations in the solution space
`prob_matrix`	a `matrix` or `data.frame`. Must be an expression matrix with features in rows and samples in columns
`current_time`	an `POSIXct` object

Value

an object of class galgo

Examples

```
# load example dataset

library(breastCancerTRANSBIG)
data(transbig)
Train <- transbig
rm(transbig)
expression <- Biobase::exprs(Train)
clinical <- Biobase::pData(Train)
OS <- survival::Surv(time = clinical$t.rfs, event = clinical$e.rfs)
# We will use a reduced dataset for the example
expression <- expression[sample(1:nrow(expression), 100), ]

# Now we scale the expression matrix
expression <- t(scale(t(expression)))

# Run galgo with base_return_pop_callback assigned to the end_galgo_callback
# hook-point
# By using this callback galgo() return a `galgo,Obj` object.
output <- GSgalgoR::galgo(generations = 5,
population = 15,
prob_matrix = expression,
OS = OS,
end_galgo_callback = callback_base_return_pop
)
```

---

callback_default *A default call_back function that does nothing.*

Description

A default call_back function that does nothing.

Usage

```
callback_default (userdir = "",generation, pop_pool,
pareto, prob_matrix, current_time)
```

Arguments

userdir	the default directory used by `galgo()` to store files
generation	a number indicating the number of iterations of the galgo algorithm
pop_pool	a `data.frame` with the solution vectors, number of clusters and their ranking.
pareto	the solutions found by Galgo across all generations in the solution space
prob_matrix	a `matrix` or `data.frame`. Must be an expression matrix with features in rows and samples in columns
current_time	an `POSIXct` object

Value

Nothing

Examples

```
# load example dataset

library(breastCancerTRANSBIG)
data(transbig)
Train <- transbig
rm(transbig)
expression <- Biobase::exprs(Train)
clinical <- Biobase::pData(Train)
OS <- survival::Surv(time = clinical$t.rfs, event = clinical$e.rfs)
# We will use a reduced dataset for the example
expression <- expression[sample(1:nrow(expression), 100), ]

# Now we scale the expression matrix
expression <- t(scale(t(expression)))

# Run galgo with default_callback assigned to all the hook-points

GSgalgoR::galgo(generations = 5,
population = 15,
prob_matrix = expression,
OS = OS,
start_galgo_callback = callback_default,# When Galgo is about to start.
end_galgo_callback = callback_default,  # When Galgo is about to finish.
start_gen_callback = callback_default,  # At the beginning of each iteration.
end_gen_callback = callback_default,    # At the end of each iteration.
)
```

`callback_no_report` *Print minimal information to the user about galgo execution.*

Description

The main idea behind this callback function is to provide some feedback to the user about galgo execution. No other relevant information is shown

Usage

```
callback_no_report (userdir = "",generation, pop_pool,
pareto, prob_matrix, current_time)
```

Arguments

`userdir`	the default directory used by 'galgo()' to store files
`generation`	a number indicating the number of iterations of the galgo algorithm
`pop_pool`	a `data.frame` with the solution vectors, number of clusters and their ranking.

pareto	the solutions found by Galgo across all generations in the solution space
prob_matrix	a matrix or data.frame. Must be an expression matrix with features in rows and samples in columns
current_time	an POSIXct object

Value

Nothing.

Examples

```
# load example dataset

library(breastCancerTRANSBIG)
data(transbig)
Train <- transbig
rm(transbig)
expression <- Biobase::exprs(Train)
clinical <- Biobase::pData(Train)
OS <- survival::Surv(time = clinical$t.rfs, event = clinical$e.rfs)
# We will use a reduced dataset for the example
expression <- expression[sample(1:nrow(expression), 100), ]

# Now we scale the expression matrix
expression <- t(scale(t(expression)))

# Run galgo with no_report_callback assigned to the report_callback
# hook-point
GSgalgoR::galgo(generations = 5,
population = 15,
prob_matrix = expression,
OS = OS,
report_callback = callback_no_report
)
```

---

classify_multiple *Classify samples from multiple centroids*

Description

Classify samples from multiple centroids

Usage

```
classify_multiple(prob_matrix, centroid_list, distancetype = "pearson")
```

Arguments

prob_matrix	a matrix or data.frame. Must be an expression matrix with features in rows and samples in columns
centroid_list	alist with the centroid matrix for each of the signatures to evaluate, where each column represents the prototypic centroid of a subtype and each row the constituents features of the solution signature. The output of create_centroids can be used.

distancetype a character that can be either 'pearson' (default), 'spearman' or 'kendall'.

Value

Returns a data.frame with the classes assigned to each sample in each signature, were samples are a rows and signatures in columns

Examples

```
# load example dataset
library(breastCancerTRANSBIG)
data(transbig)
Train <- transbig
rm(transbig)

expression <- Biobase::exprs(Train)
clinical <- Biobase::pData(Train)
OS <- survival::Surv(time = clinical$t.rfs, event = clinical$e.rfs)

# We will use a reduced dataset for the example
expression <- expression[sample(1:nrow(expression), 100), ]

# Now we scale the expression matrix
expression <- t(scale(t(expression)))

# Run galgo
output <- GSgalgoR::galgo(generations = 5, population = 15,
prob_matrix = expression, OS = OS)
outputDF <- to_dataframe(output)
outputList <- to_list(output)

RESULTS <- non_dominated_summary(
    output = output, OS = OS,
    prob_matrix = expression,
    distancetype = "pearson"
)
CentroidsList <- create_centroids(output, RESULTS$solution,
trainset = expression)
classes <- classify_multiple(prob_matrix = expression,
centroid_list = CentroidsList)
```

---

cluster_algorithm *Wrapper function to perform partition around medioids (PAM) for GalgoR*

Description

In GSgalgoR, the partition around medioids (PAM) algorithm is the default clustering process used under the evolutionary process.

Usage

```
cluster_algorithm(c, k)
```

Arguments

c	a dissimilarity matrix object of type 'dist'
k	positive integer specifying the number of clusters, less than the number of observations

Details

The function runs the pam function of the 'cluster' package with options cluster.only =TRUE, diss = TRUE, do.swap=TRUE, keep.diss=FALSE, keep.data = FALSE, pamonce= 2

Value

Returns a 'list' with the value '$cluster' which contains the cluster assignment of each of the samples evaluated

References

- Reynolds, A., Richards, G., de la Iglesia, B. and Rayward-Smith, V. (1992) Clustering rules: A comparison of partitioning and hierarchical clustering algorithms; Journal of Mathematical Modelling and Algorithms 5, 475–504. 10.1007/s10852-005-9022-1.
- Erich Schubert and Peter J. Rousseeuw (2019) Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms; Preprint, (https://arxiv.org/abs/1810.05691).

Examples

```
# load example dataset
require(iC10TrainingData)
require(pamr)
data(train.Exp)

calculate_distance <- select_distance(distancetype = "pearson")
Dist <- calculate_distance(train.Exp)
k <- 4
Pam <- cluster_algorithm(Dist, k)
table(Pam$cluster)
```

---

cluster_classify *Distance to centroid classifier function*

Description

Given an nxm matrix of centroids, where m are the prototypic centroids with n features, classify new samples according to the distance to the centroids.

Usage

```
cluster_classify(data, centroid, method = "pearson")
```

Arguments

data	a data.frame of dimensions nxp with the samples to classify, were n are the same set of features as in the centroids
centroid	a data.frame of dimensions nxm, where each column is a prototypic centroid to classify the samples
method	Character string indicating which method to use to calculate distance to centroid. Options are "pearson" (default), "kendall", or "spearman"

Value

Returns a numeric vector of length p with the class assigned to each sample according to the shortest distance to centroid

Examples

```
# load example dataset
require(iC10TrainingData)
require(pamr)

data(train.Exp)
data(IntClustMemb)
TrainData <- list(x = train.Exp, y = IntClustMemb)

# Create prototypic centroids
pam <- pamr.train(TrainData)
centroids <- pam$centroids

Class <- cluster_classify(train.Exp, centroids)
table(Class, IntClustMemb)
```

---

cosine_similarity *Function for calculating the cosine similarity*

Description

Cosine similarity is a metric of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. Two vectors with the same orientation have a cosine similarity of 1, if they are perpendicular they have a similarity of 0, and if they have opposing directions the cosine similarity is -1, independent of their magnitude. One advantage of cosine similarity is its low-complexity, especially for sparse vectors where only the non-zero dimensions need to be considered, which is a common case in GSgalgoR. Other names of cosine similarity are Otuska-Orchini similarity when it is applied to binary data, which is the case for GSgalgoR, where individual solutions represented as strings of 0 and 1 are compared with t his metric.

Usage

```
cosine_similarity(a, b)
```

Arguments

a, b	A string of numbers with equal length. It can also be two binary strings of 0's and 1's

Value

In practice, the function can return numeric values from -1 to 1 according the vector orientations, where a cosine similarity of 1 implies same orientation of the vectors while -1 imply vector of opposing directions. In the binary application, values range from 0 to 1, where 0 are totally discordant vectors while 1 are identical binary vectors.

Examples

```
solution1 <- c(1, 0, 0, 1, 0, 0, 1)
solution2 <- solution1
r <- cosine_similarity(solution1, solution2)
# the cosine similarity (r) equals 1
solution2 <- abs(solution1 - 1)
r2 <- cosine_similarity(solution1, solution2)
# the cosine similarity (r2) equals 0
```

`create_centroids` *Create Centroids*

Description

This functions create the signature centroids estimated from the GalgoR output and the expression matrix of the training sets.

Usage

```
create_centroids (output, solution_names, trainset,
distancetype = "pearson")
```

Arguments

`output`	@param output An object of class `galgo.Obj`
`solution_names`	A `character` vector with the names of the solutions for which the centroids are to be calculated
`trainset`	a `matrix` or `data.frame`. Must be an expression matrix with features in rows and samples in columns
`distancetype`	a `character` that can be either `'pearson'`, `'uncentered'`, `'spearman'` or `'euclidean'`

Value

Returns a list with the centroid matrix for each of the solutions in `solution_names`, where each column represents the prototypic centroid of a subtype and each row the constituents features of the solution signature

Examples

```
# load example dataset
library(breastCancerTRANSBIG)
data(transbig)
Train <- transbig
rm(transbig)

expression <- Biobase::exprs(Train)
clinical <- Biobase::pData(Train)
OS <- survival::Surv(time = clinical$t.rfs, event = clinical$e.rfs)

# We will use a reduced dataset for the example
expression <- expression[sample(1:nrow(expression), 100), ]

# Now we scale the expression matrix
expression <- t(scale(t(expression)))

# Run galgo
output <- GSgalgoR::galgo(generations = 5, population = 15,
prob_matrix = expression, OS = OS)
outputDF <- to_dataframe(output)
outputList <- to_list(output)

RESULTS <- non_dominated_summary(
    output = output, OS = OS,
    prob_matrix = expression,
    distancetype = "pearson"
)
CentroidsList <- create_centroids(output, RESULTS$solution,
trainset = expression)
```

---

galgo *GSgalgoR main function*

Description

galgo accepts an expression matrix and a survival object to find robust gene expression signatures related to a given outcome

Usage

```
galgo (population = 30, generations = 2, nCV = 5,
distancetype = "pearson", TournamentSize = 2, period = 1825,
OS, prob_matrix, res_dir = "", start_galgo_callback = callback_default,
end_galgo_callback = callback_base_return_pop,
report_callback = callback_base_report,
start_gen_callback = callback_default,
end_gen_callback = callback_default,
verbose = 2)
```

Arguments

population	a number indicating the number of solutions in the population of solutions that will be evolved
generations	a number indicating the number of iterations of the galgo algorithm
nCV	number of cross-validation sets
distancetype	character, it can be 'pearson' (centered pearson), 'uncentered' (uncentered pearson), 'spearman' or 'euclidean'
TournamentSize	a number indicating the size of the tournaments for the selection procedure
period	a number indicating the outcome period to evaluate the RMST
OS	a survival object (see Surv function from the survival package)
prob_matrix	a matrix or data.frame. Must be an expression matrix with features in rows and samples in columns
res_dir	a character string indicating where to save the intermediate and final output of the algorithm
start_galgo_callback	optional callback function for the start of the galgo execution
end_galgo_callback	optional callback function for the end of the galgo execution
report_callback	optional callback function
start_gen_callback	optional callback function for the beginning of the run
end_gen_callback	optional callback function for the end of the run
verbose	select the level of information printed during galgo execution

Value

an object of type 'galgo.Obj' that corresponds to a list with the elements $Solutions and $ParetoFront. $Solutions is a $lx(n+5)$ matrix where n is the number of features evaluated and l is the number of solutions obtained. The submatrix lxn is a binary matrix where each row represents the chromosome of an evolved solution from the solution population, where each feature can be present (1) or absent (0) in the solution. Column $n+1$ represent the k number of clusters for each solutions. Column $n+2$ to $n+5$ shows the SC Fitness and Survival Fitness values, the solution rank, and the crowding distance of the solution in the final pareto front respectively. For easier interpretation of the 'galgo.Obj', the output can be reshaped using the to_list and to_dataframe functions

Author(s)

Martin E Guerrero-Gimenez, <mguerrero@mendoza-conicet.gob.ar>

Examples

```
# load example dataset
library(breastCancerTRANSBIG)
data(transbig)
Train <- transbig
rm(transbig)
```

```
expression <- Biobase::exprs(Train)
clinical <- Biobase::pData(Train)
OS <- survival::Surv(time = clinical$t.rfs, event = clinical$e.rfs)

# We will use a reduced dataset for the example
expression <- expression[sample(seq_len(nrow(expression)), 100), ]

# Now we scale the expression matrix
expression <- t(scale(t(expression)))

# Run galgo
output <- GSgalgoR::galgo(generations = 5, population = 15,
prob_matrix = expression, OS = OS)
outputDF <- to_dataframe(output)
outputList <- to_list(output)
```

`galgo.Obj-class` *Galgo Object class*

Description

Galgo Object class

Slots

`Solutions` matrix.

`ParetoFront` list.

`k_centroids` *Function to calculate the centroids of different groups (classes)*

Description

This function calculates the mean value for each feature of each class to calculate the prototypic centroids of the different groups

Usage

```
k_centroids(data, class)
```

Arguments

`data`	a scaled gene expression `matrix` or `data.frame` with samples as columns and features as rows
`class`	a vector with the samples classes

Value

returns a `data.frame` with the estimated prototypic centroids for each class with the features names as rownames

Examples

```
# load example dataset
require(iC10TrainingData)
require(pamr)

data(train.Exp)

calculate_distance <- select_distance(distancetype = "pearson")
Dist <- calculate_distance(train.Exp)
k <- 4
Pam <- cluster_algorithm(Dist, k)
table(Pam$cluster)
centroids <- k_centroids(train.Exp, Pam)
```

---

non_dominated_summary *Summary of the non dominated solutions*

Description

The function uses a 'galgo.Obj' as input an the training dataset to evaluate the non-dominated solutions found by GalgoR

Usage

```
non_dominated_summary (output, prob_matrix, OS,
distancetype = "pearson")
```

Arguments

output	An object of class galgo.Obj
prob_matrix	a matrix or data.frame. Must be an expression matrix with features in rows and samples in columns
OS	a survival object (see Surv function from the survival package)
distancetype	a character that can be either 'pearson', 'uncentered', 'spearman' or 'euclidean'

Value

Returns a data.frame with 5 columns and a number of rows equals to the non-dominated solutions found by GalgoR. The first column has the name of the non-dominated solutions, the second the number of partitions found for each solution (k), the third, the number of genes, the fourth the mean silhouette coefficient of the solution and the last columns has the estimated C.Index for each one.

Examples

```
# load example dataset
library(breastCancerTRANSBIG)
data(transbig)
Train <- transbig
rm(transbig)
```

```
expression <- Biobase::exprs(Train)
clinical <- Biobase::pData(Train)
OS <- survival::Surv(time = clinical$t.rfs, event = clinical$e.rfs)

# We will use a reduced dataset for the example
expression <- expression[sample(1:nrow(expression), 100), ]

# Now we scale the expression matrix
expression <- t(scale(t(expression)))

# Run galgo
output <- GSgalgoR::galgo(generations = 5, population = 15,
prob_matrix = expression, OS = OS)
non_dominated_summary(
    output = output,
    OS = OS,
    prob_matrix = expression,
    distancetype = "pearson"
)
```

---

plot_pareto *Plot pareto front from an galgo.Obj*

Description

Plot pareto front from an galgo.Obj

Usage

```
plot_pareto(output)
```

Arguments

output	An object of class galgo.Obj

Value

This function returns a scatterplot showing the solutions found by Galgo accross all generations in the solution space, where the Silhouette Fitness is in the x-axis and the survival fitness in the y-axis. A line is drawn over all non-dominated solutions showing the estimated Pareto front

Examples

```
# load example dataset
library(breastCancerTRANSBIG)
data(transbig)
Train <- transbig
rm(transbig)

expression <- Biobase::exprs(Train)
clinical <- Biobase::pData(Train)
OS <- survival::Surv(time = clinical$t.rfs, event = clinical$e.rfs)

# We will use a reduced dataset for the example
```

```
expression <- expression[sample(1:nrow(expression), 100), ]

# Now we scale the expression matrix
expression <- t(scale(t(expression)))

# Run galgo
output <- GSgalgoR::galgo(generations = 5, population = 15,
prob_matrix = expression, OS = OS)
plot_pareto(output)
```

`surv_fitness` *Survival fitness function using the Restricted Mean Survival Time (RMST) of each group*

Description

Survival fitness function using the Restricted Mean Survival Time (RMST) of each group as proposed by *Dehbi & Royston et al. (2017)*.

Usage

```
surv_fitness(OS, clustclass, period)
```

Arguments

`OS`	a `survival` object with survival data of the patients evaluated
`clustclass`	a numeric vector with the group label for each patient
`period`	a number representing the period of time to evaluate in the RMST calculation

Value

The function computes the Harmonic mean of the differences between Restricted Mean Survival Time (RMST) of consecutive survival curves multiplied by the number of comparisons.

Author(s)

Martin E Guerrero-Gimenez, <mguerrero@mendoza-conicet.gob.ar>

References

Dehbi Hakim-Moulay, Royston Patrick, Hackshaw Allan. Life expectancy difference and life expectancy ratio: two measures of treatment effects in randomized trials with non-proportional hazards BMJ 2017; 357 :j2250 https://www.bmj.com/content/357/bmj.j2250

Examples

```
# load example dataset
library(breastCancerTRANSBIG)
library(Biobase)
data(transbig)
Train <- transbig
rm(transbig)
```

```
clinical <- pData(Train)
OS <- survival::Surv(time = clinical$t.rfs, event = clinical$e.rfs)

surv_fitness(OS, clustclass = clinical$grade, period = 3650)
```

`to_dataframe` *Convert galgo.Obj to data.frame*

Description

The current function transforms a `galgo.Obj` to a `data.frame`

Usage

```
to_dataframe(output)
```

Arguments

`output`	An object of class `galgo.Obj`

Value

The current function restructurates a `galgo.Obj` to a more easy to understand an use `data.frame`. The output `data.frame` has mxn dimensions, were the rownames (m) are the solutions obtained by the `galgo` algorithm. The columns has the following structure:

1. **Genes**: The features included in each solution in form of a `list`
2. **k**: The number of partitions found in that solution
3. **SC.Fit**: The average silhouette coefficient of the partitions found
4. **Surv.Fit**: The survival fitness value
5. **Rank**: The solution rank
6. **CrowD**: The solution crowding distance related to the rest of the solutions

Author(s)

Martin E Guerrero-Gimenez, <mguerrero@mendoza-conicet.gob.ar>

Examples

```
# load example dataset
library(breastCancerTRANSBIG)
data(transbig)
Train <- transbig
rm(transbig)

expression <- Biobase::exprs(Train)
clinical <- Biobase::pData(Train)
OS <- survival::Surv(time = clinical$t.rfs, event = clinical$e.rfs)

# We will use a reduced dataset for the example
expression <- expression[sample(1:nrow(expression), 100), ]
```

```
# Now we scale the expression matrix
expression <- t(scale(t(expression)))

# Run galgo
output <- GSgalgoR::galgo(generations = 5, population = 15,
prob_matrix = expression, OS = OS)
outputDF <- to_dataframe(output)
outputList <- to_list(output)
```

`to_list` *Convert galgo.Obj to list*

Description

The current function transforms a `galgo.Obj` to a `list`

Usage

```
to_list(output)
```

Arguments

`output`	An object of class `galgo.Obj`

Value

The current function restructurates a `galgo.Obj` to a more easy to understand an use `list`. This output is particularly useful if one wants to select a given solution and use its outputs in a new classifier. The output of type `list` has a length equals to the number of solutions obtained by the `galgo` algorithm.

Basically this output is a list of lists, where each element of the output is named after the solution's name (`solution.n`, where `n` is the number assigned to that solution), and inside of it, it has all the constituents for that given solution with the following structure:

1. **output$solution.n$Genes**: A vector of the features included in the solution
2. **output$solution.n$k**: The number of partitions found in that solution
3. **output$solution.n$SC.Fit**: The average silhouette coefficient of the partitions found
4. **output$solution.n$Surv.Fit**: The survival fitness value
5. **output$solution.n$Rank**: The solution rank
6. **CrowD**: The solution crowding distance related to the rest of the solutions

Author(s)

Martin E Guerrero-Gimenez, <mguerrero@mendoza-conicet.gob.ar>

Examples

```
# load example dataset
library(breastCancerTRANSBIG)
data(transbig)
Train <- transbig
rm(transbig)

expression <- Biobase::exprs(Train)
clinical <- Biobase::pData(Train)
OS <- survival::Surv(time = clinical$t.rfs, event = clinical$e.rfs)

# We will use a reduced dataset for the example
expression <- expression[sample(1:nrow(expression), 100), ]

# Now we scale the expression matrix
expression <- t(scale(t(expression)))

# Run galgo
output <- GSgalgoR::galgo(generations = 5, population = 15,
prob_matrix = expression, OS = OS)
outputDF <- to_dataframe(output)
outputList <- to_list(output)
```

Index