# Package 'FSinR'

May 7, 2026

**Maintainer** Alfonso Jiménez-Vílchez <i52jivia@uco.es>

**Language** en-US

**Type** Package

**Title** Feature Selection in R

**Description**
Feature subset selection algorithms modularized in search algorithms and measure utilities.

**Version** 2.0.10

**Date** 2025-11-12

**Repository** CRAN

**License** GPL-3

**Imports** digest, caret, Rdpack, GA, dplyr, tidyr, prodlim, rlang, purrr, e1071

**RdMacros** Rdpack

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**Suggests** testthat, knitr, rmarkdown, RSNNS, randomForest

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Alfonso Jiménez-Vílchez [aut, cre] (ORCID: <https://orcid.org/0000-0002-9800-8160>), Francisco Aragón-Royón [aut] (ORCID: <https://orcid.org/0000-0003-2025-5898>), Adan M. Rodriguez [aut], Antonio Arauzo-Azofra [aut] (ORCID: <https://orcid.org/0000-0002-2486-5792>), José Manuel Benítez [aut] (ORCID: <https://orcid.org/0000-0002-2346-0793>)

**Date/Publication** 2025-11-12 00:40:02 UTC

# Contents

antColony . . . 3
binaryConsistency . . . 5
breadthFirst . . . 6
chiSquared . . . 7
cramer . . . 7
deepFirst . . . 8
determinationCoefficient . . . 9
directFeatureSelection . . . 10
directSearchAlgorithm . . . 12
featureSelection . . . 13
filterEvaluator . . . 16
fscore . . . 17
gainRatio . . . 18
geneticAlgorithm . . . 19
giniIndex . . . 20
hillClimbing . . . 21
hybridFeatureSelection . . . 23
hybridSearchAlgorithm . . . 25
IEConsistency . . . 26
IEPConsistency . . . 27
is.discrete . . . 28
isDataframeContinuous . . . 29
isDataframeDiscrete . . . 29
Jd . . . 30
LasVegas . . . 31
LCC . . . 32
MDLC . . . 33
mutualInformation . . . 34
normalizedRelief . . . 35
normalizedReliefFeatureSetMeasure . . . 36
relief . . . 37
ReliefFeatureSetMeasure . . . 38
roughsetConsistency . . . 39
searchAlgorithm . . . 40
selectDifference . . . 42
selectKBest . . . 43
selectPercentile . . . 44
selectSlope . . . 45
selectThreshold . . . 46
selectThresholdRange . . . 47
sequentialBackwardSelection . . . 48
sequentialFloatingBackwardSelection . . . 49
sequentialFloatingForwardSelection . . . 50
sequentialForwardSelection . . . 51
simulatedAnnealing . . . 52
symmetricalUncertain . . . 53

tabu . . . . . 54
whaleOptimization . . . . . 56
wrapperEvaluator . . . . . 57

**Index** **60**

---

`antColony` *Ant Colony Optimization (Advanced Binary Ant Colony Optimization)*

---

## Description

Generates a search function based on the ant colony optimization. This function is called internally within the `searchAlgorithm` function. The Ant Colony Optimization (Advanced Binary Ant Colony Optimization) (Kashef and Nezamabadi-pour 2015) algorithm consists of generating in each iteration a random population of individuals (ants) according to the values of a pheromone matrix (which is updated each iteration according to the paths most followed by the ants) and a heuristic (which determines how good is each path to follow by the ants). The evaluation measure is calculated for each individual. The algorithm ends once the established number of iterations has been reached

## Usage

```
antColony(
  population = 10,
  iter = 10,
  a = 1,
  b = 1,
  p = 0.2,
  q = 1,
  t0 = 0.2,
  tmin = 0,
  tmax = 1,
  mode = 1,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| `population` | The number of ants population |
| `iter` | The number of iterations |
| `a` | Parameter to control the influence of the pheromone (If a=0, no pheromone information is used) |
| `b` | Parameter to control the influence of the heuristic (If b=0, the attractiveness of the movements is not taken into account) |
| `p` | Rate of pheromone evaporation |

| | |
|---|---|
| q | Constant to determine the amount of pheromone deposited by the best ant. This amount is determined by the Q/F equation (for minimization) where F is the cost of the solution (F/Q for maximization) |
| t0 | Initial pheromone level |
| tmin | Minimum pheromone value |
| tmax | Maximum pheromone value |
| mode | Heuristic information measurement. 1 -> min redundancy (by default). 2-> max-relevance and min-redundancy. 3-> feature-feature. 4-> based on F-score |
| verbose | Print the partial results in each iteration |

## Value

Returns a search function that is used to guide the feature selection process.

## Author(s)

Francisco Aragón Royón

## References

Kashef S, Nezamabadi-pour H (2015). "An advanced ACO algorithm for feature subset selection." *Neurocomputing*, **147**, 271 - 279. ISSN 0925-2312, doi:10.1016/j.neucom.2014.06.067, Advances in Self-Organizing Maps Subtitle of the special issue: Selected Papers from the Workshop on Self-Organizing Maps 2012 (WSOM 2012), https://www.sciencedirect.com/science/article/abs/pii/S0925231214008601.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a search process in a feature space
## Classification problem

# Generates the filter evaluation function with ACO
filter_evaluator <- filterEvaluator('IEConsistency')

# Generates the search function
aco_search <- antColony()
# Performs the search process directly (parameters: dataset, target variable and evaluator)
aco_search(iris, 'Species', filter_evaluator)
```

`binaryConsistency` *Binary consistency measure*

## Description

Generates an evaluation function that calculates the binary consistency, also known as "Sufficiency test" from FOCUS (Almuallim and Dietterich 1991) (set measure). This function is called internally within the `filterEvaluator` function.

## Usage

```
binaryConsistency()
```

## Value

Returns a function that is used to generate an evaluation set measure using the binary consistency value for the selected features.

## Author(s)

Adan M. Rodriguez

## References

Almuallim H, Dietterich TG (1991). "Learning With Many Irrelevant Features." In *In Proceedings of the Ninth National Conference on Artificial Intelligence*, 547–552.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly to evaluate a set of features
## Classification problem

data("Titanic")
titanic <- as.data.frame(Titanic)

# A discrete dataset is used (in this case we use only several discrete columns)
titanic_subset <- titanic[, c("Class", "Sex", "Age", "Survived")]

# Generate the evaluation function with Binary Consistency
bc_evaluator <- binaryConsistency()
# Evaluate the features (parameters: dataset, target variable and features)
bc_evaluator(titanic_subset, "Survived", c("Class", "Sex", "Age"))
```

`breadthFirst` *Breadth First Search (exhaustive search)*

## Description

Generates a search function based on the breadth first search. This function is called internally within the `searchAlgorithm` function. Breadth First Search searches the whole features subset in breadth first order (Kozen 1992).

## Usage

```
breadthFirst()
```

## Value

Returns a search function that is used to guide the feature selection process.

## Author(s)

Adan M. Rodriguez

Francisco Aragón Royón

## References

Kozen DC (1992). *Depth-First and Breadth-First Search*. Springer New York, New York, NY. ISBN 978-1-4612-4400-4, doi:10.1007/9781461244004_4.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a search process in a feature space
## Classification problem

# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('IEConsistency')

# Generates the search function with Breadth first
bfs_search <- breadthFirst()
# Performs the search process directly (parameters: dataset, target variable and evaluator)
bfs_search(iris, 'Species', filter_evaluator)
```

## chiSquared *Chi squared measure*

### Description

Generates an evaluation function that calculates the Chi squared value (F.R.S. 1900), evaluating the selected features individually (individual measure). This function is called internally within the `filterEvaluator` function.

### Usage

```
chiSquared()
```

### Value

Returns a function that is used to generate an individual evaluation measure using chi squared.

### References

F.R.S. KP (1900). "X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling." *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, **50**(302), 157-175. doi:10.1080/14786440009463897, https://doi.org/10.1080/14786440009463897, https://doi.org/10.1080/14786440009463897.

### Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly to individually evaluate a set of features
## Classification problem

# Generate the evaluation function with Chi squared
chiSquared_evaluator <- chiSquared()
# Evaluate the features (parameters: dataset, target variable and features)
chiSquared_evaluator(iris,'Species',c('Sepal.Length'))
```

## cramer *Cramer V measure*

### Description

Generates an evaluation function that calculates Cramer's V value (Cramer 1946), evaluating features individually (individual measure). This function is called internally within the `filterEvaluator` function.

## Usage

```
cramer()
```

## Value

Returns a function that is used to generate an individual evaluation measure using Cramer V.

## References

Cramer H (1946). *Mathematical methods of statistics / by Harald Cramer*. Princeton University Press Princeton. ISBN ISBN 0-691-08004-6.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly to individually evaluate a set of features
## Classification problem

# Generate the evaluation function with Cramer
cramer_evaluator <- cramer()
# Evaluate the features (parameters: dataset, target variable and features)
cramer_evaluator(iris,'Species',c('Sepal.Length'))
```

---

`deepFirst` *Deep First Search (exhaustive search)*

---

## Description

Generates a search function based on the deep first search. This function is called internally within the `searchAlgorithm` function. Deep First Search searches the whole features subset in deep first order (Kozen 1992).

## Usage

```
deepFirst()
```

## Value

Returns a search function that is used to guide the feature selection process.

## Author(s)

Francisco Aragón Royón

## References

Kozen DC (1992). *Depth-First and Breadth-First Search*. Springer New York, New York, NY. ISBN 978-1-4612-4400-4, doi:10.1007/9781461244004_4.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a search process in a feature space
## Classification problem

# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('IEConsistency')

# Generates the search function with Deep first
dfs_search <- deepFirst()
# Performs the search process directly (parameters: dataset, target variable and evaluator)
dfs_search(iris, 'Species', filter_evaluator)
```

---

`determinationCoefficient`

*R Squared, to continous features*

---

## Description

Generates an evaluation function that calculates the determinantion coefficient (Dodge 2008) of continuous features (set measure). This function is called internally within the `filterEvaluator` function.

## Usage

```
determinationCoefficient()
```

## Value

Returns a function that is used to generate an evaluation set measure using the R squared value for the selected features.

## Author(s)

Adan M. Rodriguez

## References

Dodge Y (2008). *Coefficient of Determination*. Springer New York, New York, NY. ISBN 978-0-387-32833-1, doi:10.1007/9780387328331_62.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly to evaluate a set of features
## Classification problem

# Generate the evaluation function with Determination Coefficient
dc_evaluator <- determinationCoefficient()
# Evaluate the features (parameters: dataset, target variable and features)
dc_evaluator(longley, 'Employed', c('GNP', 'Population','Year'))
```

---

`directFeatureSelection` *Direct Feature Selection Proccess*

---

## Description

Performs the direct feature selection process. Given a direct search algorithm and an evaluation method, it uses the direct search algorithm in combination with the evaluation results to guide the feature selection process to an optimal subset.

## Usage

```
directFeatureSelection(data, class, directSearcher, evaluator)
```

## Arguments

| | |
|---|---|
| `data` | A data.frame with the input dataset where the examples are in the rows and the features and the target variable are in the columns. The dataset should be discrete (feature columns are expected to be factors) if the following filter methods are used as evaluation methods: Rough Set Consistency, Binary Consistency, IE Consistency, IEP Consistency, Mutual Information, Gain Ratio, Symmetrical Uncertain, Gini Index or MDLC. The Jd and F-Score filter methods only work on classification problems with 2 classes in the target variable. |
| `class` | The name of the dependent variable |
| `directSearcher` | The algorithm to conduct the direct feature search. See `directSearchAlgorithm`. |
| `evaluator` | The evaluation method to obtain a measure of the features. The evaluation method can be a filter (see `filterEvaluator`) or a wrapper method (see `wrapperEvaluator`). |

## Value

A list is returned with the results of the direct feature selection process:

**bestFeatures** A vector with all features. Selected features are marked with 1, unselected features are marked with 0.

**featuresSelected** The names of the returned features sorted according to the result of the evaluation measure

**valuePerFeature** The evaluation measures of the returned features

**evaluationType** Type of evaluation based on how the features have been evaluated.

**evaluationMethod** Evaluation method used.

**searchMethod** Search method used during the feature selection process.

**target** A character indicating if the objective of the process is to minimize or maximize the evaluation measure.

**numFeatures** Number of features in the problem.

**xNames** Name of the features.

**yNames** Name of the dependent variable.

**time** Value of class 'proc_time' containing the user time, system time, and total time of the feature selection process.

## Author(s)

Francisco Aragón Royón

## References

There are no references for Rd macro `\insertAllCites` on this help page.

## Examples

```
## Examples of the direct feature selection process
## Classification problem with filter

# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('ReliefFeatureSetMeasure')
# Generates the direct search function
direct_search_method <- directSearchAlgorithm('selectKBest')
# Runs the direct feature selection process
res <- directFeatureSelection(iris, 'Species', direct_search_method, filter_evaluator)


## Classification problem with wrapper

# Generates the wraper evaluation function
wrapper_evaluator <- wrapperEvaluator('knn')
# Generates the direct search function
direct_search_method <- directSearchAlgorithm('selectKBest')
# Runs the direct feature selection process
res <- directFeatureSelection(iris, 'Species', direct_search_method, wrapper_evaluator)


## Examples of the direct feature selection process (with parameters)
## Regression problem with filter
```

```
# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('relief', list(neighbours.count = 4))
# Generates the direct search function
direct_search_method <- directSearchAlgorithm('selectKBest', list(k=2))
# Runs the direct feature selection process
res <- directFeatureSelection(mtcars, 'mpg', direct_search_method, filter_evaluator)


## Regression problem with wrapper

# Values for the caret trainControl function (resampling parameters)
resamplingParams <- list(method = "cv", repeats = 5)
# Values for the caret train function (fitting parameters)
fittingParams <- list(preProc = c("center", "scale"), metric="RMSE",
                      tuneGrid = expand.grid(k = c(1:12)))
# Generates the wraper evaluation function
wrapper_evaluator <- wrapperEvaluator('knn', resamplingParams, fittingParams)
# Generates the direct search function
direct_search_method <- directSearchAlgorithm('selectKBest',list(k=2))
# Runs the direct feature selection process
res <- directFeatureSelection(mtcars, 'mpg', direct_search_method, wrapper_evaluator)
```

---

`directSearchAlgorithm` *Direct search algorithm generator*

---

## Description

Generates a direct search function. This function in combination with the evaluator composes the feature selection process. Specifically, the result of calling this function is another function that is passed on as a parameter to the `directFeatureSelection` function. However, you can run this function directly to perform a direct search process.

## Usage

```
directSearchAlgorithm(directSearcher, params = list())
```

## Arguments

| | |
|---|---|
| `directSearcher` | Name of the direct search algorithm. The available direct search algorithms are:<br>**selectKBest** See `selectKBest`<br>**selectPercentile** See `selectPercentile`<br>**selectThreshold** See `selectThreshold`<br>**selectThresholdRange** See `selectThresholdRange`<br>**selectDifference** See `selectDifference`<br>**selectSlope** See `selectSlope` |
| `params` | List with the parameters of each direct search method. For more details see each method. Default: empty list. |

## Value

Returns a direct search function that is used in the feature selection process.

## Author(s)

Francisco Aragón Royón

## References

There are no references for Rd macro \insertAllCites on this help page.

## Examples

```
## Examples of a direct search algorithm generation

direct_search_method_1 <- directSearchAlgorithm('selectKBest')
direct_search_method_2 <- directSearchAlgorithm('selectPercentile')
direct_search_method_3 <- directSearchAlgorithm('selectThreshold')


## Examples of a direct search algorithm generation (with parameters)

direct_search_method_1 <- directSearchAlgorithm('selectKBest', list(k=2))
direct_search_method_2 <- directSearchAlgorithm('selectPercentile', list(percentile=25))
direct_search_method_3 <- directSearchAlgorithm('selectThreshold', list(threshold=0.55))


## The direct application of this function is an advanced use that consists of using this
# function directly to perform a direct search process
## Classification problem


# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('IEConsistency')

# Generates the direct search function
direct_search_method <- directSearchAlgorithm('selectKBest')
# Performs the diret search process directly (parameters: dataset, target variable and evaluator)
direct_search_method(iris, 'Species', filter_evaluator)
```

---

# featureSelection *Feature Selection Proccess*

---

## Description

Performs the feature selection process. Given a search algorithm and an evaluation method, it uses the search algorithm in combination with the evaluation results to guide the feature selection process to an optimal subset.

## Usage

```
featureSelection(data, class, searcher, evaluator)
```

## Arguments

| | |
|---|---|
| data | A data.frame with the input dataset where the examples are in the rows and the features and the target variable are in the columns. The dataset should be discrete (feature columns are expected to be factors) if the following filter methods are used as evaluation methods: Rough Set Consistency, Binary Consistency, IE Consistency, IEP Consistency, Mutual Information, Gain Ratio, Symmetrical Uncertain, Gini Index or MDLC. If Ant Colony Optimization is used as a search strategy, the dataset must be numerical since heuristics only work with continuous values. The Jd and F-Score filter methods only work on classification problems with 2 classes in the target variable. |
| class | The name of the dependent variable |
| searcher | The algorithm to guide the search in the feature space. See searchAlgorithm. |
| evaluator | The evaluation method to obtain a measure of the features. The evaluation method can be a filter (see filterEvaluator) or a wrapper method (see wrapperEvaluator). |

## Value

A list is returned with the results of the feature selection process:

**bestFeatures** A vector with all features. Selected features are marked with 1, unselected features are marked with 0.

**bestValue** Evaluation measure obtained with the feature selection.

**evaluationType** Type of evaluation based on how the features have been evaluated.

**evaluationMethod** Evaluation method used.

**measureType** Type of evaluation measure.

**searchMethod** Search method used during the feature selection process.

**target** A character indicating if the objective of the process is to minimize or maximize the evaluation measure.

**numFeatures** Number of features in the problem.

**xNames** Name of the features.

**yNames** Name of the dependent variable.

**time** Value of class 'proc_time' containing the user time, system time, and total time of the feature selection process.

## Author(s)

Francisco Aragón Royón

## References

There are no references for Rd macro \insertAllCites on this help page.

## Examples

```
## Examples of the feature selection process
## Classification problem with filter

# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('ReliefFeatureSetMeasure')
# Generates the search function
search_method <- searchAlgorithm('hillClimbing')
# Runs the feature selection process
res <- featureSelection(iris, 'Species', search_method, filter_evaluator)


## Classification problem with wrapper

# Generates the wraper evaluation function
wrapper_evaluator <- wrapperEvaluator('knn')
# Generates the search function
search_method <- searchAlgorithm('hillClimbing')
# Runs the feature selection process
res <- featureSelection(iris, 'Species', search_method, wrapper_evaluator)


## Examples of the feature selection process (with parameters)
## Regression problem with filter

# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('ReliefFeatureSetMeasure', list(iterations = 10))
# Generates the search function
search_method <- searchAlgorithm('hillClimbing', list(repeats=2))
# Runs the feature selection process
res <- featureSelection(mtcars, 'mpg', search_method, filter_evaluator)


## Regression problem with wrapper

# Values for the caret trainControl function (resampling parameters)
resamplingParams <- list(method = "cv", repeats = 5)
# Values for the caret train function (fitting parameters)
fittingParams <- list(preProc = c("center", "scale"), metric="RMSE",
                      tuneGrid = expand.grid(k = c(1:12)))
# Generates the wraper evaluation function
wrapper_evaluator <- wrapperEvaluator('knn', resamplingParams, fittingParams)
# Generates the search function
search_method <- searchAlgorithm('geneticAlgorithm',list(popSize=10, maxiter=25, verbose=TRUE))
# Runs the feature selection process
res <- featureSelection(mtcars, 'mpg', search_method, wrapper_evaluator)
```

`filterEvaluator` *Filter measure generator*

## Description

Generates a filter function to be used as an evaluator in the feature selection proccess. More specifically, the result of calling this function is another function that is passed on as a parameter to the `featureSelection` function. However, you can also run this function directly to generate an evaluation measure.

## Usage

```
filterEvaluator(filter, params = list())
```

## Arguments

| | |
|---|---|
| `filter` | Name of the filter method. The available filter methods are:<br>**binaryConsistency** Binary consistency measure. See `binaryConsistency`<br>**chiSquared** Chi squared measure. See `chiSquared`<br>**cramer** Cramer V measure. See `cramer`<br>**determinationCoefficient** R Squared, to continous features. See `determinationCoefficient`<br>**fscore** F-score measure. See `fscore`<br>**gainRatio** The gain ratio measure. See `gainRatio`<br>**giniIndex** Gini index measure. See `giniIndex`<br>**IEConsistency** Inconsistent Examples consistency measure. See `IEConsistency`<br>**IEPConsistency** Inconsistent Examples Pairs consistency measure. See `chiSquared`<br>**Jd** Jd evaluation measure. See `Jd`<br>**MDLC** MDLC evaluation measure. See `MDLC`<br>**mutualInformation** The mutual information measure. See `mutualInformation`<br>**roughsetConsistency** Rough Set consistency measure. See `roughsetConsistency`<br>**relief** Relief. See `relief`<br>**ReliefFeatureSetMeasure** Relief Feature Set Measure evaluation measure. See `ReliefFeatureSetMeasure`<br>**symmetricalUncertain** Symmetrical uncertain measure. See `symmetricalUncertain` |
| `params` | List with the parameters of each filter method. For more details see each method. Default: empty list. |

## Value

Returns a filter method that is used to generate an evaluation measure.

## Author(s)

Francisco Aragón Royón

## References

There are no references for Rd macro \insertAllCites on this help page.

## Examples

```
## Examples of a filter evaluator generation

filter_evaluator_1 <- filterEvaluator('cramer')
filter_evaluator_2 <- filterEvaluator('gainRatio')
filter_evaluator_3 <- filterEvaluator('MDLC')


## Examples of a filter evaluator generation (with parameters)

filter_evaluator_1 <- filterEvaluator('relief', list(neighbours.count=4, sample.size=15))
filter_evaluator_2 <- filterEvaluator('ReliefFeatureSetMeasure', list(iterations = 10))


## The direct application of this function is an advanced use that consists of using this
# function directly to evaluate a set of features
## Classification problem

# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('ReliefFeatureSetMeasure')
# Evaluates features directly (parameters: dataset, target variable and features)
filter_evaluator(iris,'Species',c('Sepal.Length','Sepal.Width','Petal.Length','Petal.Width'))
```

---

# fscore *F-score measure*

---

## Description

Generates an evaluation function that calculates the F-score approach defined in (Wang et al. 2018) (individual measure). This function is called internally within the filterEvaluator function.

## Usage

```
fscore()
```

## Value

Returns a function that is used to generate an individual evaluation measure using the F-score.

## References

Wang D, Zhang Z, Bai R, Mao Y (2018). "A hybrid system with filter approach and multiple population genetic algorithm for feature selection in credit scoring." *Journal of Computational and Applied Mathematics*, **329**, 307–321. ISSN 0377-0427, doi:10.1016/j.cam.2017.04.036, The International Conference on Information and Computational Science, 2–6 August 2016, Dalian, China, https://www.sciencedirect.com/science/article/abs/pii/S0377042717302078.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly to individually evaluate a set of features
## Classification problem

# Generate the evaluation function with F-Score
fscore_evaluator <- fscore()
# Evaluate the features (parameters: dataset, target variable and features)
fscore_evaluator(ToothGrowth, 'supp', c('len'))
```

---

# gainRatio *The gain ratio measure*

---

## Description

Generates an evaluation function that calculates the gain ratio value (Quinlan 1986), using the information theory (set measure). This function is called internally within the `filterEvaluator` function.

## Usage

```
gainRatio()
```

## Value

Returns a function that is used to generate an evaluation set measure using the gain ratio value for the selected features.

## Author(s)

Adan M. Rodriguez

## References

Quinlan JR (1986). "Induction of decision trees." *Machine Learning*, **1**(1), 81–106. ISSN 1573-0565, doi:10.1007/BF00116251.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly to evaluate a set of features
## Classification problem
data("Titanic")
titanic <- as.data.frame(Titanic)

# A discrete dataset is used (in this case we use only several discrete columns)
titanic_subset <- titanic[, c("Class", "Sex", "Age", "Survived")]

# Generate the evaluation function with Gain Ratio
gr_evaluator <- gainRatio()
# Evaluate the features (parameters: dataset, target variable and features)
gr_evaluator(titanic_subset, "Survived", c("Class", "Sex", "Age"))
```

---

# geneticAlgorithm *Genetic Algorithm*

---

## Description

Generates a search function based on a genetic algorithm. This function is called internally within the searchAlgorithm function. The geneticAlgorithm method (Yang and Honavar 1998) starts with an initial population of solutions and at each step applies a series of operators to the individuals in order to obtain new and better population of individuals. These operators are selection, crossing and mutation methods. This method uses the GA package implementation (Scrucca 2013) (Scrucca 2017).

## Usage

```
geneticAlgorithm(
  popSize = 20,
  pcrossover = 0.8,
  pmutation = 0.1,
  maxiter = 100,
  run = 100,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| popSize | The popuplation size |
| pcrossover | The probability of crossover between individuals |
| pmutation | The probability of mutation between individuals |
| maxiter | The number of iterations |

| | |
|---|---|
| run | Number of consecutive iterations without fitness improvement to stop the algorithm |
| verbose | Print the partial results in each iteration. This functionality is not available if the objective of the evaluation method is to minimize the target value (e.g. regression methods) |

## Value

Returns a search function that is used to guide the feature selection process.

## Author(s)

Francisco Aragón Royón

## References

Scrucca L (2013). "GA: A Package for Genetic Algorithms in R." *Journal of Statistical Software*, **53**(4), 1–37. https://www.jstatsoft.org/article/view/v053i04.

Scrucca L (2017). "On some extensions to GA package: hybrid optimisation, parallelisation and islands evolution." *The R Journal*, **9**(1), 187–206. https://journal.r-project.org/articles/RJ-2017-008/.

Yang J, Honavar V (1998). "Feature subset selection using a genetic algorithm." In *Feature extraction, construction and selection*, 117–136. Springer.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a search process in a feature space
## Classification problem

# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('IEConsistency')

# Generates the search function with Genetic algorithm
ga_search <- geneticAlgorithm()
# Performs the search process directly (parameters: dataset, target variable and evaluator)
ga_search(iris, 'Species', filter_evaluator)
```

---

# giniIndex *Gini index measure*

---

## Description

Generates an evaluation function that calculates the gini index (Ceriani and Verme 2012) of discrete features (set measure). This function is called internally within the filterEvaluator function.

## Usage

```
giniIndex()
```

## Value

Returns a function that is used to generate an evaluation set measure using the Gini index value for the selected features.

## Author(s)

Adan M. Rodriguez

## References

Ceriani L, Verme P (2012). "The origins of the Gini index: extracts from Variabilità e Mutabilità (1912) by Corrado Gini." *The Journal of Economic Inequality*, **10**(3), 421–443. ISSN 1573-8701, doi:10.1007/s108880119188x.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly to evaluate a set of features
## Classification problem

data("Titanic")
titanic <- as.data.frame(Titanic)

# A discrete dataset is used (in this case we use only several discrete columns)
titanic_subset <- titanic[, c("Class", "Sex", "Age", "Survived")]

# Generate the evaluation function with Gini Index
giniIndex_evaluator <- giniIndex()
# Evaluate the features (parameters: dataset, target variable and features)
giniIndex_evaluator(titanic_subset, "Survived", c("Class", "Sex", "Age"))
```

---

# hillClimbing *Hill-Climbing*

---

## Description

Generates a search function based on the hill climbing method. This function is called internally within the `searchAlgorithm` function. The Hill-Climbing (Russell and Norvig 2009) method starts with a certain set of features and in each iteration it searches among its neighbors to advance towards a better solution. The method ends as soon as no better solutions are found.

## Usage

```
hillClimbing(start = NULL, nneigh = NULL, repeats = 1, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| start | Binary vector with the set of initial features |
| nneigh | Number of neighbors to evaluate in each iteration of the algorithm. By default: all posibles. It is important to note that a high value of this parameter considerably increases the computation time. |
| repeats | Number of repetitions of the algorithm |
| verbose | Print the partial results in each iteration |

## Value

Returns a search function that is used to guide the feature selection process.

## Author(s)

Francisco Aragón Royón

## References

Russell S, Norvig P (2009). *Artificial Intelligence: A Modern Approach*, 3rd edition. Prentice Hall Press, Upper Saddle River, NJ, USA. ISBN 0136042597, 9780136042594.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a search process in a feature space
## Classification problem

# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('IEConsistency')

# Generates the search function with Hill-Climbing
hc_search <- hillClimbing()
# Performs the search process directly (parameters: dataset, target variable and evaluator)
hc_search(iris, 'Species', filter_evaluator)
```

hybridFeatureSelection

*Hybrid Feature Selection Proccess*

## Description

Performs the hybrid feature selection process. Given a hybrid search algorithm and an two evaluation methods, it uses the hybrid search algorithm in combination with the evaluation results to guide the feature selection process to an optimal subset.

## Usage

```
hybridFeatureSelection(data, class, hybridSearcher, evaluator_1, evaluator_2)
```

## Arguments

| | |
|---|---|
| data | A data.frame with the input dataset where the examples are in the rows and the features and the target variable are in the columns. The dataset should be discrete (feature columns are expected to be factors) if the following filter methods are used as evaluation methods: Rough Set Consistency, Binary Consistency, IE Consistency, IEP Consistency, Mutual Information, Gain Ratio, Symmetrical Uncertain, Gini Index or MDLC. The Jd and F-Score filter methods only work on classification problems with 2 classes in the target variable. |
| class | The name of the dependent variable |
| hybridSearcher | The algorithm to guide the hybrid search in the feature space. See hybridSearchAlgorithm. |
| evaluator_1 | The first evaluation method. This method can be a filter (see filterEvaluator) or a wrapper method (see wrapperEvaluator). |
| evaluator_2 | The second evaluation method. This method can be a filter (see filterEvaluator) or a wrapper method (see wrapperEvaluator). If the LCC algorithm is used, the measure must evaluate feature sets. |

## Value

A list is returned with the results of the hybrid feature selection process:

**bestFeatures** A vector with all features. Selected features are marked with 1, unselected features are marked with 0.

**bestValue** Evaluation measure obtained with the feature selection.

**evaluationType_1** Type of evaluation based on how the features have been evaluated.

**evaluationMethod_1** Evaluation method used for the first evaluator.

**measureType_1** Type of evaluation measure for the first evaluator.

**evaluationType_2** Type of evaluation based on how the features have been evaluated for the first evaluator.

**evaluationMethod_2** Evaluation method used for the second evaluator.

**measureType_2** Type of evaluation measure for the second evaluator.

**searchMethod** Search method used during the feature selection process for the second evaluator.

**target** A character indicating if the objective of the process is to minimize or maximize the evaluation measure.

**numFeatures** Number of features in the problem.

**xNames** Name of the features.

**yNames** Name of the dependent variable.

**time** Value of class 'proc_time' containing the user time, system time, and total time of the feature selection process.

## Author(s)

Francisco Aragón Royón

## References

There are no references for Rd macro `\insertAllCites` on this help page.

## Examples

```
## Examples of the hybrid feature selection process
## Classification problem with filter

# Generates the first filter evaluation function (individual or set measure)
f_evaluator_1 <- filterEvaluator('determinationCoefficient')
# Generates the second filter evaluation function (mandatory set measure)
f_evaluator_2 <- filterEvaluator('ReliefFeatureSetMeasure')
# Generates the hybrid search function
hybrid_search_method <- hybridSearchAlgorithm('LCC')
# Runs the hybrid feature selection process
res <- hybridFeatureSelection(iris,'Species',hybrid_search_method,f_evaluator_1,f_evaluator_2)


## Classification problem with wrapper

# Generates the first wrapper evaluation function (individual or set measure)
w_evaluator_1 <- wrapperEvaluator('rf')
# Generates the second wrapper evaluation function (mandatory set measure)
w_evaluator_2 <- wrapperEvaluator('knn')
# Generates the hybrid search function
hybrid_search_method <- hybridSearchAlgorithm('LCC')
# Runs the hybrid feature selection process
res <- hybridFeatureSelection(iris,'Species',hybrid_search_method,w_evaluator_1,w_evaluator_2)


## Classification problem mixed (with filter & wrapper)

# Generates the first filter evaluation function (individual or set measure)
f_evaluator <- filterEvaluator('determinationCoefficient')
```

```
# Generates the second wrapper evaluation function (mandatory set measure)
w_evaluator <- wrapperEvaluator('knn')
# Generates the hybrid search function
hybrid_search_method <- hybridSearchAlgorithm('LCC')
# Runs the hybrid feature selection process
res <- hybridFeatureSelection(iris, 'Species', hybrid_search_method, f_evaluator, w_evaluator)
```

---

`hybridSearchAlgorithm` *Hybrid search algorithm generator*

---

## Description

Generates a hybrid search function. This function in combination with the evaluator guides the feature selection process. Specifically, the result of calling this function is another function that is passed on as a parameter to the `featureSelection` function. However, you can run this function directly to perform a search process in the features space.

## Usage

```
hybridSearchAlgorithm(hybridSearcher, params = list())
```

## Arguments

| | |
|---|---|
| `hybridSearcher` | Name of the hybrid search algorithm. The available hybrid search algorithms are:<br>**LCC** Linear Consistency-Constrained algorithm (LCC). See `LCC` |
| `params` | List with the parameters of each hybrid search method. For more details see each method. Default: empty list. |

## Value

Returns a hybrid search function that is used to guide the feature selection process.

## Author(s)

Francisco Aragón Royón

## References

There are no references for Rd macro `\insertAllCites` on this help page.

## Examples

```
## Examples of a hybrid search algorithm generation

hybrid_search_method <- hybridSearchAlgorithm('LCC')


## Examples of a hybrid search algorithm generation (with parameters)

hybrid_search_method <- hybridSearchAlgorithm('LCC', list(threshold = 0.8))



## The direct application of this function is an advanced use that consists of using this
# function directly to perform a hybrid search process on a feature space
## Classification problem

# Generates the first filter evaluation function (individual or set measure)
filter_evaluator_1 <- filterEvaluator('determinationCoefficient')
# Generates the second filter evaluation function (mandatory set measure)
filter_evaluator_2 <- filterEvaluator('ReliefFeatureSetMeasure')

# Generates the hybrid search function
hybrid_search_method <- hybridSearchAlgorithm('LCC')
# Run the search process directly (params: dataset, target variable, evaluator1 & evaluator2)
hybrid_search_method(iris, 'Species', filter_evaluator_1, filter_evaluator_2)
```

---

`IEConsistency` *Inconsistent Examples consistency measure*

---

## Description

Generates an evaluation function that calculates the inconsistent examples consistency value (Dash and Liu 2003), using hash tables (set measure). This function is called internally within the `filterEvaluator` function.

## Usage

```
IEConsistency()
```

## Value

Returns a function that is used to generate an evaluation set measure using the inconsistent examples consistency value for the selected features.

## Author(s)

Adan M. Rodriguez

## References

Dash M, Liu H (2003). "Consistency-based Search in Feature Selection." *Artif. Intell.*, **151**(1-2), 155–176. ISSN 0004-3702, doi:10.1016/S00043702(03)000791, http://dx.doi.org/10.1016/S0004-3702(03)00079-1.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly to evaluate a set of features
## Classification problem

data("Titanic")
titanic <- as.data.frame(Titanic)

# A discrete dataset is used (in this case we use only several discrete columns)
titanic_subset <- titanic[, c("Class", "Sex", "Age", "Survived")]

# Generate the evaluation function with Binary Consistency
IEC_evaluator <- IEConsistency()
# Evaluate the features (parameters: dataset, target variable and features)
IEC_evaluator(titanic_subset, "Survived", c("Class", "Sex", "Age"))
```

---

# IEPConsistency *Inconsistent Examples Pairs consistency measure*

---

## Description

Generates an evaluation function that calculates the inconsistent examples pairs consistency value, using hash tables (Arauzo-Azofra et al. 2007) (set measure). This function is called internally within the `filterEvaluator` function.

## Usage

```
IEPConsistency()
```

## Value

Returns a function that is used to generate an evaluation set measure using the inconsistent examples pairs consistency value for the selected features.

## Author(s)

Adan M. Rodriguez

## References

Arauzo-Azofra A, Benitez JM, Castro JL (2007). "Consistency measures for feature selection." *Journal of Intelligent Information Systems*, **30**(3), 273–292. ISSN 1573-7675, doi:10.1007/s10844-00700370, http://dx.doi.org/10.1007/s10844-007-0037-0.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly to evaluate a set of features
## Classification problem

data("Titanic")
titanic <- as.data.frame(Titanic)

# A discrete dataset is used (in this case we use only several discrete columns)
titanic_subset <- titanic[, c("Class", "Sex", "Age", "Survived")]

# Generate the evaluation function with Binary Consistency
IEPC_evaluator <- IEPConsistency()
# Evaluate the features (parameters: dataset, target variable and features)
IEPC_evaluator(titanic_subset, "Survived", c("Class", "Sex", "Age"))
```

---

`is.discrete` *is.discrete(collection)*

---

## Description

Estimate if a collection contains discrete values

## Usage

```
is.discrete(collection)
```

## Arguments

| | |
|---|---|
| `collection` | • A collection of values |

## Value

- True if the collecion is discrete, False otherwise

## Author(s)

Alfonso Jiménez Vílchez

## Examples

```
is.discrete(mtcars$gear)
is.discrete(iris$Sepal.Length)
```

---

`isDataframeContinuous` *isDataframeContinuous(dataframe)*

---

## Description

Estimate if all variables in a data frame are continuous

## Usage

```
isDataframeContinuous(dataframe)
```

## Arguments

| | |
|---|---|
| `dataframe` | • A data frame |

## Value

- True if all variables are continuous, False otherwise

## Author(s)

Alfonso Jiménez Vílchez

## Examples

```
isDataframeContinuous(mtcars)
isDataframeContinuous(iris)
```

---

`isDataframeDiscrete` *isDataFrameDiscrete(dataframe)*

---

## Description

Estimate if all variables in a data frame are discrete

## Usage

```
isDataframeDiscrete(dataframe)
```

## Arguments

| | |
|---|---|
| `dataframe` | • A data frame |

## Value

- True if all variables are discrete, False otherwise

## Author(s)

Alfonso Jiménez Vílchez

## Examples

```
isDataframeDiscrete(mtcars)
isDataframeDiscrete(iris)
```

---

Jd *Jd evaluation measure*

---

## Description

Generates an evaluation function that applies the discriminant function designed by Narendra and Fukunaga (Narendra and Fukunaga 1977) to generate an evaluation measure for a set of features (set measure). This function is called internally within the `filterEvaluator` function.

## Usage

```
Jd()
```

## Value

Returns a function that is used to generate an evaluation set measure using the Jd.

## Author(s)

Alfonso Jiménez-Vílchez

## References

Narendra P, Fukunaga K (1977). "A Branch and Bound Algorithm for Feature Subset Selection." *IEEE Transactions on Computers*, **26**(9), 917–922. ISSN 0018-9340, doi:10.1109/TC.1977.1674939.

## Examples

```
## Not run:

## The direct application of this function is an advanced use that consists of using this
# function directly to evaluate a set of features
## Classification problem

# Generate the evaluation function with JD
Jd_evaluator <- Jd()
# Evaluate the features (parametes: dataset, target variable and features)
```

```
Jd_evaluator(ToothGrowth,'supp',c('len','dose'))

## End(Not run)
```

---

LasVegas *Las Vegas*

---

## Description

Generates a search function based on Las Vegas algorithm. This function is called internally within the `searchAlgorithm` function. The LasVegas method (Liu and Setiono 1996) starts with a certain set of features and in each step a new set is randomly generated, if the new set is better it is saved as the best solution. The algorithm ends when there are no improvements in a certain number of iterations.

## Usage

```
LasVegas(start = NULL, K = 50, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| `start` | Binary vector with the set of initial features (1: selected and 0: unselected) for the algorithm |
| `K` | The maximum number of iterations without improvement to finalize the algorithm |
| `verbose` | Print the partial results in each iteration |

## Value

Returns a search function that is used to guide the feature selection process.

## Author(s)

Francisco Aragón Royón

## References

Liu H, Setiono R (1996). "Feature Selection And Classification - A Probabilistic Wrapper Approach." In *in Proceedings of the 9th International Conference on Industrial and Engineering Applications of AI and ES*, 419–424.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a search process in a feature space

## Classification problem

# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('IEConsistency')

# Generates the search function with Las Vegas
LV_search <- LasVegas()

# Performs the search process directly (parameters: dataset, target variable and evaluator)
LV_search(iris, 'Species', filter_evaluator)
```

---

# LCC *Linear Consistency-Constrained algorithm*

---

## Description

Generates a hybrid search function based on Linear Consistency-Constrained algorithm described in (Shin and Xu 2009). The algorithm combines two evaluation measures, the first evaluates each feature individually, and the second measure evaluate feature sets.

## Usage

```
LCC(threshold = 0.9)
```

## Arguments

| | |
|---|---|
| threshold | Threshold |

## Value

Returns a hybrid search function that is used to guide the feature selection process.

## Author(s)

Alfonso Jiménez-Vílchez

## References

Shin K, Xu XM (2009). "Consistency-Based Feature Selection." In Velásquez JD, Ríos SA, Howlett RJ, Jain LC (eds.), *Knowledge-Based and Intelligent Information and Engineering Systems*, 342–350. ISBN 978-3-642-04595-0.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a hybrid search process in a feature space
## Classification problem

# Generates the first filter evaluation function (individual or set measure)
filter_evaluator_1 <- filterEvaluator('determinationCoefficient')
# Generates the second filter evaluation function (mandatory set measure)
filter_evaluator_2 <- filterEvaluator('ReliefFeatureSetMeasure')


# Generates the hybrid search function with LCC
LCC_hybrid_search <- LCC()
# Run the search process directly (params: dataset, target variable, evaluator1 & evaluator2)
LCC_hybrid_search(iris, 'Species', filter_evaluator_1, filter_evaluator_2)
```

---

MDLC *MDLC evaluation measure*

---

## Description

Generates an evaluation function that applies the Minimum-Description_Length-Criterion (MDLC) (Sheinvald et al. 1990) to generate an evaluation measure for a set of features (set measure). This function is called internally within the `filterEvaluator` function.

## Usage

```
MDLC()
```

## Value

Returns a function that is used to generate an evaluation set measure using MDLC value for the selected features

## Author(s)

Alfonso Jiménez-Vílchez

## References

Sheinvald J, Dom B, Niblack W (1990). "A modeling approach to feature selection." In *[1990] Proceedings. 10th International Conference on Pattern Recognition*, volume i, 535–539 vol. doi:10.1109/ICPR.1990.118160.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly to evaluate a set of features
## Classification problem

# Generate the evaluation function with Binary Consistency
MDLC_evaluator <- MDLC()
# Evaluate the features (parameters: dataset, target variable and features)
MDLC_evaluator(iris, "Species", c("Sepal.Length","Sepal.Width"))
```

---

`mutualInformation` *The mutual information measure*

---

## Description

Generates an evaluation function that calculates the mutual information value, using the information theory (Qian and Shu 2015) (set measure). This function is called internally within the `filterEvaluator` function.

## Usage

```
mutualInformation()
```

## Value

Returns a function that is used to generate an evaluation set measure using the mutual information value for the selected features.

## Author(s)

Adan M. Rodriguez

## References

Qian W, Shu W (2015). "Mutual information criterion for feature selection from incomplete data." *Neurocomputing*, **168**, 210–220. ISSN 18728286, doi:10.1016/j.neucom.2015.05.105, http://dx.doi.org/10.1016/j.neucom.2015.05.105.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly to evaluate a set of features
## Classification problem
data("Titanic")
titanic <- as.data.frame(Titanic)
```

```
# A discrete dataset is used (in this case we use only several discrete columns)
titanic_subset <- titanic[, c("Class", "Sex", "Age", "Survived")]

# Generate the evaluation function with Mutual Information
mi_evaluator <- mutualInformation()
# Evaluate the features (parameters: dataset, target variable and features)
mi_evaluator(titanic_subset, "Survived", c("Class", "Sex", "Age"))
```

---

`normalizedRelief` *Normalized Relief*

---

## Description

Generates an evaluation function that calculates a measure of the set of features between 0 and 1 with relief (individual measure). The relief algorithm (Kira and Rendell 1992) finds weights of continous and discrete attributes basing on a distance between instances. Adapted from Piotr Romanski's Fselector package (Romanski and Kotthoff 2018). This function is called internally within the `filterEvaluator` function.

## Usage

```
normalizedRelief(neighbours.count = 5, sample.size = 10)
```

## Arguments

| | |
|---|---|
| `neighbours.count` | • number of neighbours to find for every sampled instance |
| `sample.size` | • number of instances to sample |

## Details

relief classification and regression continous and discrete data

## Value

Returns a function that is used to generate an individual evaluation measure using relief

## Author(s)

Alfonso Jiménez-Vílchez

## References

Kira K, Rendell LA (1992). "A practical approach to feature selection." In *Machine Learning Proceedings 1992*, 249–256. Elsevier.

Romanski P, Kotthoff L (2018). *FSelector: Selecting Attributes*. R package version 0.31, https://CRAN.R-project.org/package=FSelector.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly to individually evaluate a set of features
## Classification problem

# Generate the evaluation function with Cramer
relief_evaluator <- normalizedRelief()
# Evaluate the features (parameters: dataset, target variable and features)
relief_evaluator(iris,'Species',c('Sepal.Length'))
```

---

`normalizedReliefFeatureSetMeasure`
*Relief Feature Set Measure evaluation measure*

---

## Description

Generates an evaluation function that applies Feature set measure based on Relief (set measure). Described in (Arauzo-Azofra et al. 2004). This function is called internally within the `filterEvaluator` function.

## Usage

```
normalizedReliefFeatureSetMeasure(iterations = 5, kNeightbours = 4)
```

## Arguments

| | |
|---|---|
| `iterations` | Number of iterations |
| `kNeightbours` | Number of neighbours |

## Value

Returns a function that is used to generate an evaluation set measure (between -1 and 1) using RFSM value for the selected features.

## Author(s)

Alfonso Jiménez-Vílchez

## References

Arauzo-Azofra A, Benítez J, Castro J (2004). "A feature set measure based on Relief." *Proceedings of the 5th International Conference on Recent Advances in Soft Computing*.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly to evaluate a set of features
## Classification problem

# Generate the evaluation function with Cramer
RFSM_evaluator <- ReliefFeatureSetMeasure()
# Evaluate the features (parameters: dataset, target variable and features)
RFSM_evaluator(iris,'Species',c('Sepal.Length','Sepal.Width','Petal.Length','Petal.Width'))
```

---

`relief` *Relief*

---

## Description

Generates an evaluation function that calculates a measure of the set of features with relief (individual measure). The relief algorithm (Kira and Rendell 1992) finds weights of continous and discrete attributes basing on a distance between instances. Adapted from Piotr Romanski's Fselector package (Romanski and Kotthoff 2018). This function is called internally within the `filterEvaluator` function.

## Usage

```
relief(neighbours.count = 5, sample.size = 10)
```

## Arguments

| | |
|---|---|
| `neighbours.count` | • number of neighbours to find for every sampled instance |
| `sample.size` | • number of instances to sample |

## Details

relief classification and regression continous and discrete data

## Value

Returns a function that is used to generate an individual evaluation measure using relief

## Author(s)

Alfonso Jiménez-Vílchez

## References

Kira K, Rendell LA (1992). "A practical approach to feature selection." In *Machine Learning Proceedings 1992*, 249–256. Elsevier.

Romanski P, Kotthoff L (2018). *FSelector: Selecting Attributes*. R package version 0.31, https://CRAN.R-project.org/package=FSelector.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly to individually evaluate a set of features
## Classification problem

# Generate the evaluation function with Cramer
relief_evaluator <- relief()
# Evaluate the features (parameters: dataset, target variable and features)
relief_evaluator(iris,'Species',c('Sepal.Length'))
```

---

`ReliefFeatureSetMeasure` *Relief Feature Set Measure evaluation measure*

---

## Description

Generates an evaluation function that applies Feature set measure based on Relief (set measure). Described in (Arauzo-Azofra et al. 2004). This function is called internally within the `filterEvaluator` function.

## Usage

```
ReliefFeatureSetMeasure(iterations = 5, kNeightbours = 4)
```

## Arguments

| | |
|---|---|
| `iterations` | Number of iterations |
| `kNeightbours` | Number of neighbours |

## Value

Returns a function that is used to generate an evaluation set measure (between -1 and 1) using RFSM value for the selected features.

## Author(s)

Alfonso Jiménez-Vílchez

## References

Arauzo-Azofra A, Benítez J, Castro J (2004). "A feature set measure based on Relief." *Proceedings of the 5th International Conference on Recent Advances in Soft Computing*.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly to evaluate a set of features
## Classification problem

# Generate the evaluation function with Cramer
RFSM_evaluator <- ReliefFeatureSetMeasure()
# Evaluate the features (parameters: dataset, target variable and features)
RFSM_evaluator(iris,'Species',c('Sepal.Length','Sepal.Width','Petal.Length','Petal.Width'))
```

---

# roughsetConsistency *Rough Set consistency measure*

---

## Description

Generates an evaluation function that calculates the rough sets consistency value (Pawlak 1982) (Pawlak 1991), using hash tables (set measure). This function is called internally within the `filterEvaluator` function.

## Usage

```
roughsetConsistency()
```

## Value

Returns a function that is used to generate an evaluation set measure using the rough sets consistency value for the selected features.

## Author(s)

Adan M. Rodriguez

## References

Pawlak Z (1982). "Rough sets." *International Journal of Computer & Information Sciences*, **11**, 341–356. ISSN 0091-7036, doi:10.1007/BF01001956, https://www.sciencedirect.com/science/article/abs/pii/S0377042717302078.

Pawlak Z (1991). *Rough sets: Theoretical aspects of reasoning about data*, volume 9(1). Springer, Dordrecht. doi:10.1007/9789401135344, http://dx.doi.org/10.1007/978-94-011-3534-4.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly to evaluate a set of features
## Classification problem

data("Titanic")
titanic <- as.data.frame(Titanic)

# A discrete dataset is used (in this case we use only several discrete columns)
titanic_subset <- titanic[, c("Class", "Sex", "Age", "Survived")]

# Generate the evaluation function with Rough Set Consistency
rsc_evaluator <- roughsetConsistency()
# Evaluate the features (parameters: dataset, target variable and features)
rsc_evaluator(titanic_subset, "Survived", c("Class", "Sex", "Age"))
```

---

# searchAlgorithm *Search algorithm generator*

---

## Description

Generates a search function. This function in combination with the evaluator guides the feature selection process. Specifically, the result of calling this function is another function that is passed on as a parameter to the `featureSelection` function. However, you can run this function directly to perform a search process in the features space.

## Usage

```
searchAlgorithm(searcher, params = list())
```

## Arguments

`searcher` Name of the search algorithm. The available search algorithms are:

**antColony** Ant colony optimization (ACO). See `antColony`

**breadthFirst** Breadth first search. See `breadthFirst`

**deepFirst** Deep first search. See `deepFirst`

**geneticAlgorithm** Genetic algorithm (GA). See `geneticAlgorithm`

**hillClimbing** Hill-Climbing (HC). See `hillClimbing`

**LasVegas** Las Vegas (LV). See `LasVegas`

**sequentialBackwardSelection** Sequential backward selection (sbs). See `sequentialBackwardSelecti`

**sequentialFloatingForwardSelection** Sequential floating forward selection (sffs). See `sequentialFloatingForwardSelection`

**sequentialFloatingBackwardSelection** Sequential floating backward selection (sfbs). See `sequentialFloatingBackwardSelection`

**sequentialForwardSelection** Sequential forward selection (sfs). See sequentialForwardSelection

**simulatedAnnealing** Simulated annealing (SA). See simulatedAnnealing

**tabu** Tabu search (TS). See tabu

**whaleOptimization** Whale optimization algorithm (WOA). See whaleOptimization

`params` List with the parameters of each search method. For more details see each method. Default: empty list.

## Value

Returns a search function that is used to guide the feature selection process

## Author(s)

Francisco Aragón Royón

## References

There are no references for Rd macro `\insertAllCites` on this help page.

## Examples

```
## Examples of a search algorithm generation

search_method_1 <- searchAlgorithm('antColony')
search_method_2 <- searchAlgorithm('sequentialBackwardSelection')
search_method_3 <- searchAlgorithm('tabu')


## Examples of a search algorithm generation (with parameters)

search_method_1 <- searchAlgorithm('antColony', list(population=25, iter=50, verbose=TRUE))
search_method_2 <- searchAlgorithm('sequentialBackwardSelection', list(stop=TRUE))
search_method_3 <- searchAlgorithm('tabu', list(intensification=1, iterIntensification=25))


## The direct application of this function is an advanced use that consists of using this
# function directly to perform a search process on a feature space
## Classification problem

# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('IEConsistency')

# Generates the search function
search_method <- searchAlgorithm('hillClimbing')
# Performs the search process directly (parameters: dataset, target variable and evaluator)
search_method(iris, 'Species', filter_evaluator)
```

`selectDifference` *Select difference*

## Description

Generates a direct search function that selects features (in descending order from the best evaluation measure to the lowest) until evaluation difference is over a threshold (The features evaluation is individual). This function is called internally within the `directSearchAlgorithm` function.

## Usage

```
selectDifference(d.threshold = 0.2)
```

## Arguments

| | |
|---|---|
| `d.threshold` | • Number between 0 and 1, to calculate the slope |

## Value

Returns a direct search function that is used in the feature selection process.

## Author(s)

Adan M. Rodriguez

Francisco Aragón Royón

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a direct search process
## Classification problem


# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('IEConsistency')

# Generates the direct search function with difference
sd_direct_search <- selectDifference()
# Performs the direct search process directly (parameters: dataset, target variable and evaluator)
sd_direct_search(iris, 'Species', filter_evaluator)
```

selectKBest *Select K best*

## Description

Generates a direct search function that takes the 'k' features with the greatest evaluations (The features evaluation is individual). This function is called internally within the directSearchAlgorithm function.

## Usage

```
selectKBest(k = 1)
```

## Arguments

| | |
|---|---|
| k | Number (positive integer) of returned features |

## Value

Returns a direct search function that is used in the feature selection process.

## Author(s)

Adan M. Rodriguez

Francisco Aragón Royón

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a direct search process
## Classification problem


# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('IEConsistency')

# Generates the direct search function with k-best
skb_direct_search <- selectKBest()
# Performs the direct search process directly (parameters: dataset, target variable and evaluator)
skb_direct_search(iris, 'Species', filter_evaluator)
```

`selectPercentile` *Select Percentile*

## Description

Generates a direct search function that selects a fraction, given as a percentage, of the total number of available features (The features evaluation is individual). This function is called internally within the `directSearchAlgorithm` function.

## Usage

```
selectPercentile(percentile = 80)
```

## Arguments

| | |
|---|---|
| `percentile` | Number (positive integer) between 0 and 100 |

## Value

Returns a direct search function that is used in the feature selection process.

## Author(s)

Adan M. Rodriguez

Francisco Aragón Royón

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a direct search process
## Classification problem


# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('IEConsistency')

# Generates the direct search function with percentile
sp_direct_search <- selectPercentile()
# Performs the direct search process directly (parameters: dataset, target variable and evaluator)
sp_direct_search(iris, 'Species', filter_evaluator)
```

selectSlope *Select slope*

## Description

Generates a direct search function that selects features (in descending order from the best evaluation measure to the lowest) until the slope to the next feature is over a threshold (The features evaluation is individual). The slope is calculated as: (s.threshold) / (number of features). This function is called internally within the directSearchAlgorithm function.

## Usage

```
selectSlope(s.threshold = 1.5)
```

## Arguments

| | |
|---|---|
| s.threshold | • Number between 0 and 1 |

## Value

Returns a direct search function that is used in the feature selection process.

## Author(s)

Adan M. Rodriguez

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a direct search process
## Classification problem


# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('IEConsistency')

# Generates the direct search function with slope
ss_direct_search <- selectSlope()
# Performs the direct search process directly (parameters: dataset, target variable and evaluator)
ss_direct_search(iris, 'Species', filter_evaluator)
```

`selectThreshold` *Select threshold*

## Description

Generates a direct search function that selects the features whose evaluation is over/under a user given threshold (It depends on the method that generates the evaluation measure. For example: under for regression methods, over for classification methods, etc.)(The features evaluation is individual). Features that do not satisfy the threshold, will be removed. This function is called internally within the `directSearchAlgorithm` function.

## Usage

```
selectThreshold(threshold = 0.1)
```

## Arguments

| | |
|---|---|
| `threshold` | • Number between 0 and 1 |

## Value

Returns a direct search function that is used in the feature selection process.

## Author(s)

Adan M. Rodriguez

Francisco Aragón Royón

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a direct search process
## Classification problem


# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('IEConsistency')

# Generates the direct search function with threshold
st_direct_search <- selectThreshold()
# Performs the direct search process directly (parameters: dataset, target variable and evaluator)
st_direct_search(iris, 'Species', filter_evaluator)
```

selectThresholdRange *Select threshold range*

## Description

Generates a direct search function that selects the features whose evaluation is over a threshold, where this threshold is given as: (((min - max) * p.threshold) + max)(The features evaluation is individual). This function is called internally within the directSearchAlgorithm function.

## Usage

```
selectThresholdRange(p.threshold = 0.8)
```

## Arguments

| | |
|---|---|
| p.threshold | • Number between 0 and 1 |

## Value

Returns a direct search function that is used in the feature selection process.

## Author(s)

Adan M. Rodriguez

Francisco Aragón Royón

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a direct search process
## Classification problem


# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('IEConsistency')

# Generates the direct search function with threshold range
str_direct_search <- selectThresholdRange()
# Performs the direct search process directly (parameters: dataset, target variable and evaluator)
str_direct_search(iris, 'Species', filter_evaluator)
```

`sequentialBackwardSelection` *Sequential Backward Selection*

## Description

Generates a search function based on sequential backward selection. This function is called internally within the `searchAlgorithm` function. The SBS method (Marill and Green 1963) starts with all the features and removes a single feature at each step with a view to improving the evaluation of the set.

## Usage

```
sequentialBackwardSelection(stopCriterion = -1, stop = FALSE)
```

## Arguments

| | |
|---|---|
| `stopCriterion` | Define a maximum number of iterations. Disabled if the value is -1 (default: -1 ) |
| `stop` | If true, the function stops if next iteration does not improve current results (default: FALSE) |

## Value

Returns a search function that is used to guide the feature selection process.

## Author(s)

Adan M. Rodriguez

Alfonso Jiménez-Vílchez

Francisco Aragón Royón

## References

Marill T, Green D (1963). "On the effectiveness of receptors in recognition systems." *Information Theory, IEEE Transactions on*, **9**(1), 11–17. doi:10.1109/TIT.1963.1057810, http://dx.doi.org/10.1109/TIT.1963.1057810.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a search process in a feature space
## Classification problem

# Generates the filter evaluation function with sbs
filter_evaluator <- filterEvaluator('IEConsistency')
```

```
# Generates the search function
sbs_search <- sequentialBackwardSelection()
# Performs the search process directly (parameters: dataset, target variable and evaluator)
sbs_search(iris, 'Species', filter_evaluator)
```

---

`sequentialFloatingBackwardSelection`

*Sequential Floating Backward Selection*

---

## Description

Generates a search function based on sequential floating backward selection. This function is called internally within the `searchAlgorithm` function. The sfbs method (Pudil et al. 1994) starts with all the features and removes a single feature at each step with a view to improving the evaluation of the set. In addition, it checks whether adding any of the removed features, improve the value of the set.

## Usage

```
sequentialFloatingBackwardSelection()
```

## Value

Returns a search function that is used to guide the feature selection process.

## Author(s)

Adan M. Rodriguez

Francisco Aragón Royón

## References

Pudil P, Novovičová J, Kittler J (1994). "Floating search methods in feature selection." *Pattern recognition letters*, **15**(11), 1119–1125.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a search process in a feature space
## Classification problem

# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('IEConsistency')

# Generates the search function with sfbs
sfbs_search <- sequentialFloatingBackwardSelection()
```

```
# Performs the search process directly (parameters: dataset, target variable and evaluator)
sfbs_search(iris, 'Species', filter_evaluator)
```

---

`sequentialFloatingForwardSelection`
*Sequential Floating Forward Selection*

---

## Description

Generates a search function based on sequential floating forward selection. This function is called internally within the `searchAlgorithm` function. The sffs method (Pudil et al. 1994) starts with an empty set of features and add a single feature at each step with a view to improving the evaluation of the set. In addition, it checks whether removing any of the included features, improve the value of the set.

## Usage

```
sequentialFloatingForwardSelection()
```

## Value

Returns a search function that is used to guide the feature selection process.

## Author(s)

Adan M. Rodriguez

Francisco Aragón Royón

Alfonso Jiménez Vílchez

## References

Pudil P, Novovičová J, Kittler J (1994). "Floating search methods in feature selection." *Pattern recognition letters*, **15**(11), 1119–1125.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a search process in a feature space
## Classification problem

# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('IEConsistency')

# Generates the search function with sffs
sffs_search <- sequentialFloatingForwardSelection()
# Performs the search process directly (parameters: dataset, target variable and evaluator)
sffs_search(iris, 'Species', filter_evaluator)
```

sequentialForwardSelection
*Sequential Forward Selection*

## Description

Generates a search function based on sequential forward selection. This function is called internally within the `searchAlgorithm` function. The SFS method (Whitney 1971) starts with an empty set of features and add a single feature at each step with a view to improving the evaluation of the set.

## Usage

```
sequentialForwardSelection(stopCriterion = -1, stop = FALSE)
```

## Arguments

| | |
|---|---|
| `stopCriterion` | Define a maximum number of iterations. Disabled if the value is -1 (default: -1 ) |
| `stop` | If true, the function stops if next iteration does not improve current results (default: FALSE) |

## Value

Returns a search function that is used to guide the feature selection process.

## Author(s)

Adan M. Rodriguez

Alfonso Jiménez-Vílchez

Francisco Aragón Royón

## References

Whitney AW (1971). "A Direct Method of Nonparametric Measurement Selection." *IEEE Trans. Comput.*, **20**(9), 1100–1103. ISSN 0018-9340, doi:10.1109/TC.1971.223410, http://dx.doi.org/10.1109/T-C.1971.223410.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a search process in a feature space
## Classification problem

# Generates the filter evaluation function with sfs
filter_evaluator <- filterEvaluator('IEConsistency')
```

```
# Generates the search function
sfs_search <- sequentialForwardSelection()
# Performs the search process directly (parameters: dataset, target variable and evaluator)
sfs_search(iris, 'Species', filter_evaluator)
```

---

`simulatedAnnealing` *Simulated Annealing*

---

## Description

Generates a search function based on simulated annealing. This function is called internally within the `searchAlgorithm` function. The simulatedAnnealing method (Kirkpatrick et al. 1983) starts with a certain set of features and in each iteration modifies an element of the previous feature vector and decreases the temperature. If the energy of the new feature vector is better than that of the old vector, it is accepted and moved towards it, otherwise it is moved towards the new vector according to an acceptance probability. The algorithm ends when the minimum temperature has been reached. Additionally, a number of internal iterations can be performed within each iteration of the algorithm. In this case, the same temperature value of the outer iteration is used for the inner iterations

## Usage

```
simulatedAnnealing(
  start = NULL,
  temperature = 1,
  temperature_min = 0.01,
  reduction = 0.6,
  innerIter = 1,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| `start` | Binary vector with the set of initial features |
| `temperature` | Temperature initial |
| `temperature_min` | Temperature to stops in the outer loop |
| `reduction` | Temperature reduction in the outer loop |
| `innerIter` | Number of iterations of inner loop. By default no inner iterations are established |
| `verbose` | Print the partial results in each iteration |

## Value

Returns a search function that is used to guide the feature selection process.

## Author(s)

Francisco Aragón Royón

## References

Kirkpatrick S, Gelatt CD, Vecchi MP (1983). "Optimization by simulated annealing." *SCIENCE*, **220**(4598), 671–680. doi:10.1126/science.220.4598.671, http://dx.doi.org/10.1126/science.220.4598.671.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a search process in a feature space
## Classification problem

# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('IEConsistency')

# Generates the search function with Simulated annealing
sa_search <- simulatedAnnealing()
# Performs the search process directly (parameters: dataset, target variable and evaluator)
sa_search(iris, 'Species', filter_evaluator)
```

---

# symmetricalUncertain *Symmetrical uncertain measure*

---

## Description

Generates an evaluation function that calculates the symmetrical uncertain value (Witten and Frank 2005), using the information theory (set measure). This function is called internally within the `filterEvaluator` function.

## Usage

```
symmetricalUncertain()
```

## Value

Returns a function that is used to generate an evaluation set measure using the symmetrical uncertain value for the selected features.

## Author(s)

Adan M. Rodriguez

## References

Witten IH, Frank E (2005). *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd edition. Morgan Kaufmann, San Francisco.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly to evaluate a set of features
## Classification problem
data("Titanic")
titanic <- as.data.frame(Titanic)

# A discrete dataset is used (in this case we use only several discrete columns)
titanic_subset <- titanic[, c("Class", "Sex", "Age", "Survived")]

# Generate the evaluation function with Symmetrical Uncertain
su_evaluator <- symmetricalUncertain()
# Evaluate the features (parameters: dataset, target variable and features)
su_evaluator(titanic_subset, "Survived", c("Class", "Sex", "Age"))
```

---

# tabu *Tabu Search*

---

## Description

Generates a search function based on the tabu search. This function is called internally within the searchAlgorithm function. The Tabu Search(Glover 1986) (Glover 1989) method starts with a certain set of features and in each iteration it searches among its neighbors to advance towards a better solution. The method has a memory (tabu list) that prevents returning to recently visited neighbors. The method ends when a certain number of iterations are performed, or when a certain number of iterations are performed without improvement, or when there are no possible neighbors. Once the method is finished, an intensification phase can be carried out that begins in the space of the best solutions found, or a diversification phase can be carried out in which solutions not previously visited are explored.

## Usage

```
tabu(
  start = NULL,
  numNeigh = NULL,
  tamTabuList = 5,
  iter = 100,
  iterNoImprovement = NULL,
  intensification = NULL,
  iterIntensification = 50,
  interPercentaje = 75,
```

```
  tamIntermediateMemory = 5,
  diversification = NULL,
  iterDiversification = 50,
  forgetTabuList = TRUE,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| start | Binary vector with the set of initial features |
| numNeigh | The number of neighbor to consider in each iteration. By default: all posibles. It is important to note that a high value of this parameter considerably increases the computation time. |
| tamTabuList | The size of the tabu list. By default: 5 |
| iter | The number of iterations of the algorithm. By default: 100 |
| iterNoImprovement | Number of iterations without improvement to start/reset the intensification/diversification phase. By default, it is not taken into account (all iterations are performed) |
| intensification | Number of times the intensification phase is applied. None by default |
| iterIntensification | Number of iterations of the intensification phase |
| interPercentaje | Percentage of the most significant features to be taken into account in the intensification phase |
| tamIntermediateMemory | Number of best solutions saved in the intermediate memory |
| diversification | Number of times the diversification phase is applied. None by default |
| iterDiversification | Number of iterations of the diversification phase |
| forgetTabuList | Forget tabu list for intensification/diversification phases. By default: TRUE |
| verbose | Print the partial results in each iteration |

## Value

Returns a search function that is used to guide the feature selection process.

## Author(s)

Francisco Aragón Royón

## References

Glover F (1986). "Future Paths for Integer Programming and Links to Artificial Intelligence." *Comput. Oper. Res.*, **13**(5), 533–549. ISSN 0305-0548, doi:10.1016/03050548(86)900481, http://dx.doi.org/10.1016/0305-0548(86)90048-1.

Glover F (1989). "Tabu Search—Part I." *ORSA Journal on Computing*, **1**(3), 190-206. doi:10.1287/ijoc.1.3.190, https://doi.org/10.1287/ijoc.1.3.190.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a search process in a feature space
## Classification problem

# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('IEConsistency')

# Generates the search function wit Tabu search
ts_search <- tabu()
# Performs the search process directly (parameters: dataset, target variable and evaluator)
ts_search(iris, 'Species', filter_evaluator)
```

---

# whaleOptimization *Whale Optimization Algorithm (Binary Whale Optimization Algorithm)*

---

## Description

Generates a search function based on the whale optimization algorithm. This function is called internally within the searchAlgorithm function. Binary Whale Optimization Algorithm (Kumar and Kumar 2018) is an algorithm that simulates the social behavior of humpback whales. This algorithm employs a binary version of the bubble-net hunting strategy. The algorithm starts with an initial population of individuals, and in each iteration updates the individuals according to several possible actions: Encircling prey, Bubble-net attacking or Search for prey

## Usage

```
whaleOptimization(population = 10, iter = 10, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| population | The number of whales population |
| iter | The number of iterations of the algorithm |
| verbose | Print the partial results in each iteration |

## Value

Returns a search function that is used to guide the feature selection process.

## Author(s)

Francisco Aragón Royón

## References

Kumar V, Kumar D (2018). "Binary whale optimization algorithm and its application to unit commitment problem." *Neural Computing and Applications*. ISSN 1433-3058, doi:10.1007/s00521-01837963.

## Examples

```
## The direct application of this function is an advanced use that consists of using this
# function directly and performing a search process in a feature space
## Classification problem

# Generates the filter evaluation function
filter_evaluator <- filterEvaluator('IEConsistency')

# Generates the search function with WOA
woa_search <- whaleOptimization()
# Performs the search process directly (parameters: dataset, target variable and evaluator)
woa_search(iris, 'Species', filter_evaluator)
```

---

# wrapperEvaluator *Wrapper measure generator*

---

## Description

Generates a wrapper function to be used as an evaluator (Kohavi and John 1997) in the feature selection proccess, given a learner algorithm and related customizable parameters (from Jed Wing et al. 2018). More specifically, the result of calling this function is another function that is passed on as a parameter to the `featureSelection` function. However, you can also run this function directly to generate an evaluation measure.

## Usage

```
wrapperEvaluator(learner, resamplingParams = list(), fittingParams = list())
```

## Arguments

| | |
|---|---|
| `learner` | Learner to be used. The models available are the models available in caret: http://topepo.github.io/caret/available-models.html |
| `resamplingParams` | Control parameters for evaluating the impact of model tuning parameters. The arguments are the same as those of the caret trainControl function. By default an empty list. In this case the default caret values are used for resampling and fitting. |
| `fittingParams` | Control parameters for choose the best model across the parameters. The arguments are the same as those of the caret train function (minus the parameters: x, y, form, data, method and trainControl). By default an empty list. In this case the default caret values are used for resampling and fitting. |

## Details

generaWrapper

## Value

Returns a wrapper function that is used to generate an evaluation measure

## Author(s)

Alfonso Jiménez-Vílchez

Francisco Aragón Royón

## References

Kohavi R, John GH (1997). "Wrappers for feature subset selection." *Artificial intelligence*, **97**(1-2), 273–324.

from Jed Wing MKC, Weston S, Williams A, Keefer C, Engelhardt A, Cooper T, Mayer Z, Kenkel B, the R Core Team, Benesty M, Lescarbeau R, Ziem A, Scrucca L, Tang Y, Candan C, Hunt. T (2018). *caret: Classification and Regression Training*. R package version 6.0-80, https://CRAN.R-project.org/package=caret.

## Examples

```
## Examples of a wrapper evaluator generation

wrapper_evaluator_1 <- wrapperEvaluator('knn')
wrapper_evaluator_2 <- wrapperEvaluator('mlp')
wrapper_evaluator_3 <- wrapperEvaluator('randomForest')


## Examples of a wrapper evaluator generation (with parameters)

# Values for the caret trainControl function (resampling parameters)
```

```
resamplingParams <- list(method = "repeatedcv", repeats = 3)
# Values for the caret train function (fitting parameters)
fittingParams <- list(preProc = c("center", "scale"), metric="Accuracy",
                      tuneGrid = expand.grid(k = c(1:12)))

wrapper_evaluator <- wrapperEvaluator('knn', resamplingParams, fittingParams)


## The direct application of this function is an advanced use that consists of using this
# function directly to evaluate a set of features
## Classification problem

# Generates the wrapper evaluation function
wrapper_evaluator <- wrapperEvaluator('knn')
# Evaluates features directly (parameters: dataset, target variable and features)
wrapper_evaluator(iris,'Species',c('Sepal.Length','Sepal.Width','Petal.Length','Petal.Width'))
```

# Index


antColony, 3, *40*

binaryConsistency, 5, *16*
breadthFirst, 6, *40*

chiSquared, 7, *16*
cramer, 7, *16*

deepFirst, 8, *40*
determinationCoefficient, 9, *16*
directFeatureSelection, 10, *12*
directSearchAlgorithm, *10*, 12, *42–47*

featureSelection, 13, *16*, *25*, *40*, *57*
filterEvaluator, *5*, *7*, *9*, *10*, *14*, 16, *17*, *18*, *20*, *23*, *26*, *27*, *30*, *33–39*, *53*
fscore, *16*, 17

gainRatio, *16*, 18
geneticAlgorithm, 19, *40*
giniIndex, *16*, 20

hillClimbing, 21, *40*
hybridFeatureSelection, 23
hybridSearchAlgorithm, *23*, 25

IEConsistency, *16*, 26
IEPConsistency, 27
is.discrete, 28
isDataframeContinuous, 29
isDataframeDiscrete, 29

Jd, *16*, 30

LasVegas, 31, *40*
LCC, *25*, 32

MDLC, *16*, 33
mutualInformation, *16*, 34

normalizedRelief, 35
normalizedReliefFeatureSetMeasure, 36

relief, *16*, 37
ReliefFeatureSetMeasure, *16*, 38
roughsetConsistency, *16*, 39

searchAlgorithm, *3*, *6*, *8*, *14*, *19*, *21*, *31*, 40, *48–52*, *54*, *56*
selectDifference, *12*, 42
selectKBest, *12*, 43
selectPercentile, *12*, 44
selectSlope, *12*, 45
selectThreshold, *12*, 46
selectThresholdRange, *12*, 47
sequentialBackwardSelection, *40*, 48
sequentialFloatingBackwardSelection, *40*, 49
sequentialFloatingForwardSelection, *40*, 50
sequentialForwardSelection, *41*, 51
simulatedAnnealing, *41*, 52
symmetricalUncertain, *16*, 53

tabu, *41*, 54

whaleOptimization, *41*, 56
wrapperEvaluator, *10*, *14*, *23*, 57