Package ‘mRMRe’

February 14, 2019

Type Package

Title "Parallelized Minimum Redundancy, Maximum Relevance (mRMR) Ensemble Feature Selection"

Version 2.0.9

Date 2019-02-13

Author Nicolas De Jay, Simon Papillon-Cavanagh, Catharina Olsen, Gianluca Bontempi, Benjamin Haibe-Kains

Maintainer Benjamin Haibe-Kains <benjamin.haibe.kains@utoronto.ca>

Description "Computes mutual information matrices from continuous, categorical and survival variables, as well as feature selection with minimum redundancy, maximum relevance (mRMR) and a new ensemble mRMR technique with DOI: N De Jay et al. (2013) <doi:10.1093/bioinformatics/btt383>."

License Artistic-2.0

Depends R (>= 2.10), survival, igraph, methods

URL http://www.pmgenomics.ca/bhklab/

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-02-14 09:20:14 UTC

R topics documented:

  adjacencyMatrix .................................................. 2
  causality ........................................................... 3
  cgps ................................................................. 4
  correlate ........................................................... 5
  export_concordance_index ......................................... 6
  export_filters ...................................................... 7
  export_filters_bootstrap .......................................... 8
  export_mim .......................................................... 9
  featureCount ....................................................... 10
  featureData ....................................................... 10
adjacencyMatrix

Accessor function for the 'adjacencyMatrix' information in a mRMRe.Network object.

Description

The adjacency matrix is a directed matrix of 0's and 1's indicating if there is a link between features.

Usage

```r
## S4 method for signature 'mRMRe.Network'
adjacencyMatrix(object)
```

Arguments

- `object` a mRMRe.Network object.

Author(s)

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains
causality

Examples

```r
set.thread.count(2)
data(cgps)
feature_data <- mRMR.data(data = data.frame(cgps.ge))

# Build an mRMR-based network and display adjacency matrix (topology)
network <- new("mRMRe.Network", data = feature_data, target_indices = c(1, 2),
levels = c(2, 1), layers = 1)
adjacencyMatrix(network)
```

causality

Accessor function for the 'causality' information in a mRMRe.Filter and mRMRe.Network object.

Description

The causality data is compute using the co-information lattice algorithm on each V-structure (feature, target, feature). Given that this procedure is computed for each pair of features, the minimum result is kept. A negative score indicates putative causality of the feature to the target.

Usage

```r
## S4 method for signature 'mRMRe.Filter'
causality(object)
## S4 method for signature 'mRMRe.Network'
causality(object)
```

Arguments

- `object`: a mRMRe.Filter or mRMRe.Network object.

Author(s)

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

Examples

```r
set.thread.count(2)
data(cgps)
feature_data <- mRMR.data(data = data.frame(cgps.ge))
filter <- mRMR.classic("mRMRe.Filter", data = feature_data, target_indices = 3:5,
feature_count = 2)
causality(filter)
```
**cgps**

Part of the large pharmacogenomic dataset published by Garnett et al. within the Cancer Genome Project (CGP)

---

**Description**

This dataset contains gene expression of 200 cancer cell lines for which sensitivity (IC50) to Camptothecin was measured (release 2).

**Usage**

data(cgps)

**Format**

The `cgps` dataset is composed of three objects

- **cgps.annot** Dataframe containing gene annotations
- **cgps.ge** Matrix containing expressions of 1000 genes; cell lines in rows, genes in columns
- **cgps.ic50** Drug sensitivity measurements (IC50) for Camptothecin

**Details**

Camptothecin is a drug mainly used in colorectal cancer.

**Source**

http://www.ebi.ac.uk/arrayexpress/experiments/E-MTAB-783
http://www.nature.com/nature/journal/v483/n7391/extref/nature11005-s2.zip

**References**


**Examples**

```r
set.nthread(n=2)
data(cgps)

message("Gene expression data:")
print(cgps.ge[1:3, 1:3])

message("Gene annotations:")
print(head(cgps.annot))

message("Drug sensitivity (IC50) values:")
print(head(cgps.ic50))
```
correlate

**Function to compute various correlation measures between two variables**

**Description**

Correlate is a function that estimates correlation between two variables, which can be either continuous, categorical (ordered factor) or censored (survival data).

**Usage**

```r
correlate(x, y, method = c("pearson", "spearman", "kendall", "frequency", "cramersv", "cindex"), strata, weights, outX = TRUE, bootstrap_count = 0, alpha = 0.05, alternative = c("two.sided", "less", "greater"))
```

**Arguments**

- **x** Vector of type numeric, ordered factor, or Surv.
- **y** Vector of type numeric, ordered factor, or Surv of same length as `x`.
- **method** One of the following values: pearson, spearman, kendall, frequency, cramersv, or cindex.
- **strata** Vector of type factor corresponding to the sample strata.
- **weights** Vector of type numeric corresponding to the sample weights.
- **outX** For cindex, if set to TRUE, ignore ties; otherwise, take them into account when computing the concordance index.
- **bootstrap_count** If set to 0, analytical standard error for the correlation estimate in each strata is used to compute the meta-estimate (inverse-variance weighting average); otherwise a number of bootstraps are used to compute standard errors.
- **alpha** The probability of Type I error that is, rejecting a null hypothesis when it is in fact true
- **alternative** a character string specifying the alternative hypothesis, must be one of two.sided (default), greater or less. You can specify just the initial letter.

**Details**

The `correlate` function could be used to measure correlation between any types of variables:

- **numeric vs. numeric** Pearson, Spearman, Kendall or concordance index
- **numeric vs. ordered factor** concordance index (Somers’ Dxy)
- **numeric vs. survival data** concordance index (Somers’ Dxy)
- **ordered factor vs. ordered factor** Carmer’s V
- **ordered factor vs. survival data** concordance index (Somers’ Dxy)
- **survival data vs. survival data** concordance index (Somers’ Dxy)
Part of the code underlying `correlate` is also used in `mim` method of the `mRMRe` object because correlations are used to build the mutual information matrix in order for feature selection to take place. This is why these two functions have many arguments in common.

**Value**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>estimate</td>
<td>point estimate</td>
</tr>
<tr>
<td>se</td>
<td>standard error</td>
</tr>
<tr>
<td>lower</td>
<td>lower confidence bound</td>
</tr>
<tr>
<td>upper</td>
<td>upper confidence bound</td>
</tr>
<tr>
<td>p</td>
<td>p-value</td>
</tr>
<tr>
<td>n</td>
<td>sample size</td>
</tr>
</tbody>
</table>

**Author(s)**

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

**See Also**

`mRMRe.Data-class`

**Examples**

```r
setNthreadNcount(RI
CC load data
data(cgps)
CC spearman correlation coefficent between the first gene and Camptothecin IC50
correlate(X=pgps.ge[ ,1], Y=pgps.ic50, method="spearman")
CC concordance index between the first gene and Camptothecin IC50
correlate(X=pgps.ge[ ,1], Y=pgps.ic50, method="cindex")
```

**Description**

Export the concordance index
export_filters

Arguments

samplesA ... 
samplesB ... 
samplesC ... 
samplesD ... 
sampleStrata ... 
sampleWeights ... 
sampleStratumCount ... 
outX ... 
ratio ... 
concordantWeights ... 
discordantWeights ... 
uninformativeWeights ... 
relevantWeights ...

Author(s)

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

Description

Export the filters

Arguments

childrenCountPerLevel ...
dataMatrix ...
priorsMatrix ...
priorsWeight ...
sampleStrata ...
sampleWeights ...
featureTypes ...
sampleCount ...
featureCount  ...
sampleStratumCount
  ...
targetFeatureIndices
  ...
continuousEstimator
  ...
outX  ...
bootstrapCount  ...
mMatrix  ...

Author(s)
Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

export_filters_bootstrap

export filters bootstrap

Description
Export the filters

Arguments
solutionCount  ...
solutionLength  ...
dataMatrix  ...
priorsMatrix  ...
priorsWeight  ...
sampleStrata  ...
sampleWeights  ...
featureTypes  ...
sampleCount  ...
featureCount  ...
sampleStratumCount
  ...
targetFeatureIndices
  ...
continuousEstimator
  ...
outX  ...
bootstrapCount  ...
mMatrix  ...
Author(s)
Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

Description
Export mim

Arguments
- dataMatrix ...
- priorsMatrix ...
- priorsWeight ...
- sampleStrata ...
- sampleWeights ...
- featureTypes ...
- sampleCount ...
- featureCount ...
- sampleStratumCount ...
- continuousEstimator ...
- outX ...
- bootstrapCount ...
- miMatrix ...
**featureCount**

*Accessor function for the ‘featureCount’ information in a mRMRe.Data, mRMRe.Filter and mRMRe.Network object.*

**Description**

The feature count is simply the total number of feature considered in the mRMRe procedure.

**Usage**

```r
## S4 method for signature 'mRMRe.Data'
featureCount(object)
## S4 method for signature 'mRMRe.Filter'
featureCount(object)
## S4 method for signature 'mRMRe.Network'
featureCount(object)
```

**Arguments**

- **object**: a `mRMRe.Data`, `mRMRe.Filter` or `mRMRe.Network` object.

**Author(s)**

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

**Examples**

```r
set.thread.count(2)
data(cgps)
feature_data <- mRMRe.data(data = data.frame(cgps.ge))
featureCount(feature_data)
filter <- mRMRe.classic("mRMRe.Filter", data = feature_data, target_indices = 3:5,
feature_count = 2)
featureCount(filter)
```

---

**featureData**

*Accessor function for the ‘featureData’ information in a mRMRe.Data object*

**Description**

The featureData consists of the numerical value of each feature for each sample considered.

**Usage**

```r
## S4 method for signature 'mRMRe.Data'
featureData(object)
```
featureNames

Arguments

object a mRMRe.Data object.

Author(s)

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

Examples

```r
set_thread_count(2)
data(cgps)
feature_data <- mRMRe.data(data = data.frame(cgps.ge))
featureData(feature_data)

featureNames

Accessor function for the 'featureNames' information in a mRMRe.Data, mRMRe.Filter and mRMRe.Network object

Description

featureNames are the names of the features given as input to the mRMRe procedure.

Usage

## S4 method for signature 'mRMRe.Data'
featureNames(object)
## S4 method for signature 'mRMRe.Filter'
featureNames(object)
## S4 method for signature 'mRMRe.Network'
featureNames(object)

Arguments

object a mRMRe.Data, mRMRe.Filter or mRMRe.Network object.

Author(s)

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

Examples

```r
set_thread_count(2)
data(cgps)
feature_data <- mRMRe.data(data = data.frame(cgps.ge))
featureData(feature_data)
filter <- mRMRe.classic("mRMRe.Filter", data = feature_data, target_indices = 3:5, feature_count = 2)
featureNames(filter)
## Description

This methods allows you to retrieve the number of cores currently accessible to openMP.

## Usage

```python
get_thread_count()
```

## Author(s)

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

## Examples

```python
get_thread_count()
```

## Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread_count</td>
<td>number of OPENMP threads to be used</td>
</tr>
</tbody>
</table>

## Author(s)

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains
mim

Accessory function for the 'mim' information in a mRMRe.Data, mRMRe.Filter and mRMRe.Network object

Description

In both mRMRe.Filter and mRMRe.Network objects, a sparse mutual information matrix is computed for the mRMRe procedure and this lazy-evaluated matrix is returned. In the context of a mRMRe.Data 'mim', the full pairwise mutual information matrix is computed and returned.

Usage

## S4 method for signature 'mRMRe.Data'
mim(object, prior_weight, continuous_estimator, outX, bootstrap_count)
## S4 method for signature 'mRMRe.Filter'
mim(object, method)
## S4 method for signature 'mRMRe.Network'
mim(object)

Arguments

- **object**: a mRMRe.Data, mRMRe.Filter or mRMRe.Network object.
- **prior_weight**: a numeric value [0,1] of indicating the impact of priors (mRMRe.Data only).
- **continuous_estimator**: an estimator of the mutual information between features: either "pearson", "spearman", "kendall", "frequency" (mRMRe.Data only).
- **outX**: a boolean used in the concordance index estimator to keep or throw out ties (mRMRe.Data only).
- **bootstrap_count**: an integer indicating the number of bootstrap resampling used in estimation (mRMRe.Data only).
- **method**: either "mi" or "cor"; the latter will return the correlation coefficients (rho) while the former will return the mutual information (-0.5 * log(1 - (rho^2))).

Author(s)

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

Examples

```r
set.thread.count(2)
data(cgps)
feature_data <- mRMR.data(data = data.frame(cgps.ge))

# Calculate the pairwise mutual information matrix
mim(feature_data)
filter <- mRMR.classic("mRMRe.Filter", data = feature_data, target_indices = 3:5,
```

mRMRe.Data-class

Description

`mRMRe.Data` is the class containing datasets. Most if not all of the routines in the mRMRe package use `mRMRe.Data` objects as primary input.

Such an object is instantiated with a data frame containing the sample sets and optionally, stratum, weight vectors and a prior matrix. In addition to basic accession functions, we describe several methods which serve to manipulate the contents of the dataset.

Note that `mRMRe.data` function is a wrapper to easily create `mRMRe.Data` objects.

Instantiation

Objects are created via calls of the form `new("mRMRe.Data", data, strata, weights, priors)`. `data`: is expected to be a data frame with samples and features respectively organized as rows and columns. The columns have to be of type :numeric, ordered factor, Surv and respectively interpreted as :continuous, discrete and survival variables.

`strata`: is expected to be a vector of type :ordered factor with the strata associated to the samples provided in `data`.

`weights`: is expected to be a vector of type :numeric with the weights associated to the samples provided in `data`.

`priors`: is expected to be a matrix of type :numeric where `priors[i, j]`: denotes an forced association between features i and j in `data`. The latter takes into consideration the directionality of the relationship and must be a value between 0 and 1.

Mutual Information Matrix

The `mim` method computes and returns a mutual information matrix. A correlation between continuous features is estimated using an estimator specified in `continuous_estimator`; currently, :pearson, spearman, kendall, frequency are supported. The estimator for discrete features is Cramer’s V and for all other combinations, concordance index.

When `outX` is set to `TRUE`, ties are ignored when computing the concordance index and otherwise, these are considered. The correlations are first computed per strata and these are then combined by the inverse variance weight mean of the estimates using a `bootstrap_count` number of bootstraps if the former parameter is greater than 0, and by the relative weights of each strata otherwise. The resulting correlation is then summated with the corresponding value in the priors matrix with the latter being weighed for a proportion `prior_weight` of a final, biased correlation.
Slots

- `sample_names`: Object of class "character" containing the sample names.
- `feature_names`: Object of class "character" containing the feature names.
- `feature_types`: Object of class "numeric" containing the internal representation of features/variables:
  - 1 for numeric, 2 for ordered factor, and 3 for survival data
- `data`: Object of class "matrix" containing the internal representation of the data set.
- `strata`: Object of class "numeric" containing the feature strata.
- `weights`: Object of class "numeric" containing sample weights.
- `priors`: Object of class "matrix" containing the priors.

Methods

- `featureCount` signature(object = "mRMRe.Data"): Returns the number of features.
- `featureData` signature(object = "mRMRe.Data"): Returns a data frame corresponding to the data set.
- `featureNames` signature(object = "mRMRe.Data"): Returns a vector containing the feature names.
- `mim` signature(object = "mRMRe.Data", prior_weight = 0, continuous_estimator = c("pearson", "spearman", "kendall")): Computes and returns the mutual information matrix.
- `priors` signature(object = "mRMRe.Data"): Returns a matrix containing the priors.
- `priors<-` signature(object = "mRMRe.Data", value): Sets the prior matrix.
- `sampleCount` signature(object = "mRMRe.Data"): Returns the number of samples.
- `sampleNames` signature(object = "mRMRe.Data"): Returns a vector containing sample names.
- `sampleStrata` signature(object = "mRMRe.Data"): Returns a vector containing sample strata.
- `sampleStrata<-` signature(object = "mRMRe.Data", value): Sets the sample strata.
- `sampleWeights` signature(object = "mRMRe.Data"): Returns a vector containing sample weights.
- `sampleWeights<-` signature(object = "mRMRe.Data"): Sets the sample weights.
- `subsetData` signature(object = "mRMRe.Data", row_indices, column_indices): Returns another data object containing only the specified samples and features (rows and columns, respectively.)

Author(s)

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

See Also

- `mRMRe.Filter-class`
- `mRMRe.Network-class`
Examples

```r
showClass("mRMRe.Data")
set.thread.count(2)

## load data
data(cgps)

g <- mRMRe.data(data = data.frame(cgps.ge[, 1:10, drop=FALSE]))
g <- new("mRMRe.Data", data = data.frame(cgps.ge[, 1:10, drop=FALSE]))

## print data
print(featureData(g)[1:3, 1:3])

## print feature names
print(featureNames(g))

## print the first sample names
print(head(sampleNames(g)))

## print the first sample weights
print(head(sampleWeights(g)))
```

**mRMRe.Filter-class**  

**Class** "mRMRe.Filter"

**Description**

mRMRe.Filter is a wrapper for various variants of the maximum relevance minimum redundancy (mRMR) feature selection/filter.

Note that mRMRe.classic and mRMRe.ensemble functions are wrappers to easily perform classical (single) and ensemble mRMR feature selection.

**Instantiation**

Objects are created via calls of the form `new("mRMRe.Filter", data, prior_weight,target_indices,levels,method,continuous_estimator)`.  
- `data`: is expected to be a `mRMRe.Data` object.
- `target_indices`: is expected to be a vector of type `integer` containing the indices of the features that will serve as targets for the feature selections.
- `levels`: is expected to be a vector of type `integer` containing the number of children of each element at each level of the resulting filter tree.
- `method`: is expected to be either exhaustive or bootstrap. The former uses the whole dataset to pick siblings in the tree according to the mRMR metric, while the latter perform the classical mRMR feature selection on several bootstrap selections of the dataset.
- `continuous_estimator`: it specifies the estimators for correlation between two continuous variables; value is either `pearson`, `spearman`, `kendall`, `frequency`,
outX: set to TRUE (default value) to not count pairs of observations tied on x as a relevant pair. This results in a Goodman-Kruskal gamma type rank correlation.

bootstrap_count: Number of bootstraps to statistically compare teh mRMR scores of each solution.

Since a mutual information matrix must be computed in order for feature selection to take place, the remaining arguments are identical to those required by the mim method of the mrmreNdata object.

Slots

filters: Object of class "list" containing for each target a solutions matrix.

mi_matrix: Object of class "matrix" containing the combined mutual information matrix of the relevant targets.

causality_list: Object of class "list" containing for each target a vector of causality coefficients between the target and its predictors.

sample_names: Object of class "character" containing the sample names.

feature_names: Object of class "character" containing the feature names.

target_indices: Object of class "integer" containing the target indices.

levels: Object of class "integer" containing the desired topology of the tree.

scores: Object of class "list" containing the mRMR score of selected features, respective to filters.

Methods

causality signature(object = "mRMRe.Filter"): ...

featureCount signature(object = "mRMRe.Filter"): Returns the number of features.

featureNames signature(object = "mRMRe.Filter"): Returns a vector containing the feature names.

mim signature(object = "mRMRe.Filter"): Returns the potentially partial mutual information matrix used for feature selection.

sampleCount signature(object = "mRMRe.Filter"): Returns the number of samples.

sampleNames signature(object = "mRMRe.Filter"): Returns a vector containing sample names.

solutions signature(object = "mRMRe.Filter", mi_threshold = -Inf, causality_threshold = Inf): Returns a matrix in which each column represents a different solution (path from root of the tree to a leaf.)

target signature(object = "mRMRe.Filter"): Returns a vector containing the target indices.

Author(s)

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

References

**mRMRe.Network-class**

**Description**

*mRMRe.Network* is a wrapper for inferring a network of features based on mRMR feature selection.

**Instantiation**

Objects are created via calls of the form `new("mRMRe.Network", data, prior_weight, target_indices, levels, layers, mi_threshold, causality_threshold)`. The `layers` parameter is expected to be an integer specifying the number of layers of network inference desired. When multiple layers are desired, the elements of the solutions found in the last step of feature selection are used as the targets of the next step. Since networking involves filter processing, the remaining arguments are identical to those required by `solutions` method of the *mRMRe.Filter* object and `mim` method of the *mRMRe.Data* object.

**Slots**

- `topologies`: Object of class "list" --
- `mi_matrix`: Object of class "matrix" containing the combined mutual information matrix of the network elements.

**See Also**

*mRMRe.Data-class*

**Examples**

```r
showClass("mRMRe.Filter")
set.thread.count(2)

## load data
data(cgps)

gE <- mRMRe.data(data = data.frame(cgps.ge[,1:100, drop=FALSE]))

## perform a classic (single) mRMR to select the 10 genes the most correlated with
## the first gene but the less correlated between each other
exect <- system.time(fs <- new("mRMRe.Filter", data = ge, target_indices = 1,
levels = c(8, 1, 1, 1, 1)))
print(exect)

## print the index of the selected features for each distinct mRMR solutions
print(solutions(fs)[[1]])

## print the names of the selected features for each distinct mRMR solutions
print(apply(solutions(fs)[[1]], 2, function(x, y) { return(y[x] ) }, y=featureNames(ge)))
```
causality_list: Object of class "list" containing for each target a vector of causality coefficients between the target and its predictors.

sample_names: Object of class "character" containing the sample names.

feature_names: Object of class "character" containing the feature names.

target_indices: Object of class "integer" containing the target indices.

Methods

adjacencyMatrix signature(object = "mRMRe.Network"): Returns a matrix describing the topology of the network.

adjacencyMatrixSum signature(object = "mRMRe.Network"): ...

causality signature(object = "mRMRe.Network"): Returns a list containing vectors containing causality coefficients between targets and predictors.

featureNames signature(object = "mRMRe.Network"): Returns a vector containing the feature names.

mim signature(object = "mRMRe.Network"): ...

sampleNames signature(object = "mRMRe.Network"): Returns a vector containing sample names.

solutions signature(object = "mRMRe.Network"): ...

visualize signature(object = "mRMRe.Network"): ...

Author(s)

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

See Also

mRMRe.Filter-class, mRMRe.Data-class

Examples

showClass("mRMRe.Network")

set.thread.count(2)

## load data
data(cgps)

## build an mRMRe.Data object
ge <- mRMRe.data(data = data.frame(cgps.ge[, 1:100, drop=FALSE]))

## build a network object with the 10 first genes and their children,
## 8 distinct mRMR feature selections of 5 genes for each gene
exect <- system.time(netw <- new("mRMRe.Network", data = ge, target_indices = 1:10,
levels = c(8, 1, 1, 1, 1), layers = 2))
print(exect)

## plot network using igraph
priors

Accessor function for the 'priors' information in a mRMRe.Data object

Description

The priors matrix consists of a prior bias to be used in computation to mutual information between features.

Usage

```r
## S4 method for signature 'mRMRe.Data'
priors(object)
## S4 replacement method for signature 'mRMRe.Data'
priors(object) <- value
```

Arguments

- `object`: a mRMRe.Data object.
- `value`: a numeric matrix containing values from 0 to 1 (or NA), one per pairwise feature bias.

Author(s)

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

Examples

```r
set.thread.count(2)
data(cgps)
feature_data <- mRMRe.data(data = data.frame(cgps.ge))
priors(feature_data)
```
sampleCount

Accessor function for the 'sampleCount' information in a mRMRe.Data, mRMRe.Filter and mRMRe.Network object.

Description
The feature count is simply the total number of samples considered in the mRMRe procedure.

Usage
```
## S4 method for signature 'mRMRe.Data'
sampleCount(object)
## S4 method for signature 'mRMRe.Filter'
sampleCount(object)
## S4 method for signature 'mRMRe.Network'
sampleCount(object)
```

Arguments
- object: a mRMRe.Data, mRMRe.Filter or mRMRe.Network object.

Author(s)
Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

Examples
```
set.thread.count(2)
data(cgps)
feature_data <- mRMRe.data(data = data.frame(cgps.ge))
sampleCount(feature_data)
filter <- mRMRe.classic("mRMRe.Filter", data = feature_data, target_indices = 3:5,
feature_count = 2)
sampleCount(filter)
```

sampleNames

Accessor function for the 'sampleNames' information in a mRMRe.Data, mRMRe.Filter and mRMRe.Network object.

Description
sampleNames are the names of the samples given as input to the mRMRe procedure.
Usage

```r
## S4 method for signature 'mRMRe.Data'
sampleStrata(object)
## S4 method for signature 'mRMRe.Filter'
sampleStrata(object)
## S4 method for signature 'mRMRe.Network'
sampleStrata(object)
```

Arguments

- `object` a `mRMRe.Data`, `mRMRe.Filter` or `mRMRe.Network` object.

Author(s)

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

Examples

```r
set.thread.count(2)
data(cgps)
feature_data <- mRMRe.data(data = data.frame(cgps.ge))
sampleNames(feature_data)
filter <- mRMRe.classic("mRMRe.Filter", data = feature_data, target_indices = 3:5, feature_count = 2)
sampleNames(filter)
```

---

**sampleStrata**  
Accessory function for the 'sampleStrata' information in a `mRMRe.Data` object

Description

The `sampleStrata` vector consists of a sampling stratification that will be used in computing mutual information between features. If known batch effects or sample stratification is present between samples, identify such subsets using this.

Usage

```r
## S4 method for signature 'mRMRe.Data'
sampleStrata(object)
## S4 replacement method for signature 'mRMRe.Data'
sampleStrata(object) <- value
```

Arguments

- `object` a `mRMRe.Data` object.
- `value` a factor vector identifying the stratification of samples.
sampleWeights

Author(s)

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

Examples

```r
set.thread.count(2)
data(cgps)
feature_data <- mRMR.data(data = data.frame(cgps.ge))

# No stratification (default)
sampleStrata(feature_data)

# Random stratification
sampleStrata(feature_data) <- as.factor(sample(c(0,1),
sampleCount(feature_data), replace=TRUE))
# Show result
sampleStrata(feature_data)
```

---

sampleWeights  

Accessor function for the 'sampleWeights' information in a mRMRe.Data object

Description

TODO

Usage

```r
## S4 method for signature 'mRMRe.Data'
sampleWeights(object)
## S4 replacement method for signature 'mRMRe.Data'
sampleWeights(object) <- value
```

Arguments

- **object**  
a mRMRe>Data object.
- **value**  
a numeric vector containing the biases of each sample in the mutual information computation.

Author(s)

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains
Examples

```r
set.thread.count(2)
data(cgps)
feature_data <- mRMR.data(data = data.frame(cgps.ge))

# Uniform weight (default)
sampleWeights(feature_data)

# Random weighting
sampleWeights(feature_data) <- runif(sampleCount(feature_data))
# Show result
sampleWeights(feature_data)
```

---

**scores**

*mRMR Scores as per the MI gain for each feature*

Description

The scores method returns the scores of individual features in respect to previously selected features as per standard mRMR procedure. For each target, the score of a feature is defined as the mutual information between the target and this feature minus the average mutual information of previously selected features and this feature.

Usage

```r
## S4 method for signature 'mRMRe.Data'
scores(object, solutions)
## S4 method for signature 'mRMRe.Filter'
scores(object)
## S4 method for signature 'mRMRe.Network'
scores(object)
```

Arguments

- **object**: a `mRMRe.Data`, `mRMRe.Filter` or `mRMRe.Network` object.
- **solutions**: a set of solutions from `mRMRe.Filter` or `mRMRe.Network` to be used in computing the scores from a `mRMRe.Data` set.

Author(s)

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains
Examples

```R
set.thread.count(2)
data(cgps)
feature_data <- mRMR.data(data = data.frame(cgps.ge))

# Create an mRMR filter and obtain the indices of selected features
filter <- mRMR.classic("mRMR.e.Filter", data = feature_data, target_indices = 3:5,
feature_count = 2)
scores(filter)
```

---

**set.thread.count**

**openMP Thread Count**

Description

This method allows you to set the number of cores currently accessible to openMP.

Usage

```R
set.thread.count(thread_count)
```

Arguments

- `thread_count` number of OPENMP threads to be used

Author(s)

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

Examples

```R
# Access to number of available threads
threads <- get.thread.count()
# Force a single threaded openMP job
set.thread.count(1)

# Revert back to all accessible threads
set.thread.count(threads)
```
solutions

---

set_thread_count  

**openMP Thread Count**

---

**Description**

This method allows you to set the number of cores currently accessible to openMP.

**Arguments**

- thread_count: number of OPENMP threads to be used

**Author(s)**

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

---

solutions  

**Basic result of the mRMR procedure**

---

**Description**

The 'solutions' method allows one to access the set of selected features resulting of the mRMR algorithm. More generally, the set of feature are identified by their indices in the inputed feature set (1 being the first feature (column)). At the network level, 'solutions' consists of the topology of the network, identifying which features is connected to others.

**Usage**

```r
## S4 method for signature 'mRMRe.Filter'
solutions(object, mi_threshold, causality_threshold)
## S4 method for signature 'mRMRe.Network'
solutions(object)
```

**Arguments**

- object: a mRMRe.Filter or mRMRe.Network object.
- mi_threshold: a numeric value used in filtering the features based on their mRMR scores, features that do not pass the threshold will be set at NA.
- causality_threshold: a numeric value used in filtering the features based on their causality scores, features that do not pass the threshold will be set at NA

**Author(s)**

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains
subsetData

Examples

```r
set.thread.count(2)
data(cgps)
feature_data <- mRMR.data(data = data.frame(cgps.ge))

# Create an mRMR filter and obtain the indices of selected features
filter <- mRMR.classic("mRMR.Filter", data = feature_data, target_indices = 3:5,
                      feature_count = 2)
solutions(filter)

# Build an mRMR-based network and obtain feature connections (topology)
network <- new("mRMR.Network", data = feature_data, target_indices = c(1, 2),
               levels = c(2, 1), layers = 1)
solutions(network)
```

subsetData

Returns a mRMR.Data object using a subset of the current mRMR.Data object.

Description

This method is used to extract a subset of the current mRMR.Data object.

Usage

```r
## S4 method for signature 'mRMR.Data'
subsetData(object, row_indices, column_indices)
```

Arguments

- `object` a mRMR.Data object.
- `row_indices` An integer vector of the rows to be included in the subset.
- `column_indices` An integer vector of the columns to be included in the subset.

Author(s)

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

Examples

```r
set.thread.count(2)
data(cgps)
feature_data <- mRMR.data(data = data.frame(cgps.ge))

# Subset the same dimensions, equivalent to making a copy
feature_data_copy <- subsetData(feature_data,
                                 row_indices = sampleCount(feature_data),
                                 column_indices = featureCount(feature_data))
```
# Use only half of the samples
feature.data.samples <- subsetData(feature.data, row_indices=sampleCount(feature.data)/2)

# Use only half of the features
feature.data.features <- subsetData(feature.data, column_indices=featureCount(feature.data))

---

target | mRMR Target(s)

### Description

The `target` method allows you to access the target of a mRMR procedure. In a mRMRe.Network setting, the target consists of the seed or the starting set of features given in the network building.

### Usage

```r
## S4 method for signature 'mRMRe.Filter'
target(object)
## S4 method for signature 'mRMRe.Network'
target(object)
```

### Arguments

- `object` a mRMRe.Filter or mRMRe.Network object.

### Author(s)

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

### Examples

```r
set_thread.count(2)
data(cgps)
feature.data <- mRMR.data(data = data.frame(cgps.ge))

# Create an mRMR filter and obtain the targets of that filter
filter <- mRMR.classic("mRMR.FILTER", data = feature.data, target_indices = 3:5, feature_count = 2)
target(filter)

# Build an mRMR-based network and obtain targets (seeds) of the network
network <- new("mRMR.Network", data = feature.data, target_indices = c(1, 2), levels = c(2, 1), layers = 1)
target(network)
```
**visualize**  

*mRMRe Network display*

**Description**

The 'visualize' methods allows the visual display of an inferred mRMRe.Network topology.

**Usage**

```r
## S4 method for signature 'mRMRe.Network'
visualize(object)
```

**Arguments**

- `object`  
  a `mRMRe.Network` object.

**Author(s)**

Nicolas De Jay, Simon Papillon-Cavanagh, Benjamin Haibe-Kains

**Examples**

```r
set.thread.count(2)
data(cgps)
feature_data <- mRMR.data(data = data.frame(cgps.ge))

# Build an mRMR-based network and display it
network <- new("mRMRe.Network", data = feature_data, target_indices = c(1),
levels = c(3, 1), layers = 2)
visualize(network)
```
Index

*Topic classes
  mRMRe.Data-class, 14
  mRMRe.Filter-class, 16
  mRMRe.Network-class, 18

*Topic datasets
cgps, 4

*Topic methods
  adjacencyMatrix, 2
  causality, 3
  export_concordance_index, 6
  export_filters, 7
  export_filters_bootstrap, 8
  export_mim, 9
  featureCount, 10
  featureData, 10
  featureNames, 11
  get_thread_count, 12
  get_thread_count, 12
  mim, 13
  priors, 20
  sampleCount, 21
  sampleNames, 21
  sampleStrata, 22
  sampleWeights, 23
  scores, 24
  set_thread_count, 25
  set_thread_count, 26
  solutions, 26
  subsetData, 27
  target, 28
  visualize, 29

*Topic univar
correlate, 5

adjacencyMatrix, 2
adjacencyMatrix, mRMRe.Network-method (adjacencyMatrix), 2
adjacencyMatrixSum (adjacencyMatrix), 2
adjacencyMatrixSum, mRMRe.Network-method (adjacencyMatrix), 2

causality, 3
causality, mRMRe.Filter-method (causality), 3
causality, mRMRe.Network-method (causality), 3
cgps, 4

correlate, 5

export_concordance_index, 6
export_filters, 7
export_filters_bootstrap, 8
export_mim, 9

featureCount, 10
featureCount, mRMRe.Data-method (featureCount), 10
featureCount, mRMRe.Filter-method (featureCount), 10
featureCount, mRMRe.Network-method (featureCount), 10
featureData, 10
featureData, mRMRe.Data-method (featureData), 10
featureNames, 11
featureNames, mRMRe.Data-method (featureNames), 11
featureNames, mRMRe.Filter-method (featureNames), 11
featureNames, mRMRe.Network-method (featureNames), 11

get_thread_count, 12
generate, 12

mim, 13
mim, mRMRe.Data-method (mim), 13
mim, mRMRe.Filter-method (mim), 13
mim, mRMRe.Network-method (mim), 13
mRMRe.classic (mRMRe.Filter-class), 16
mRMRe.data (mRMRe.Data-class), 14
INDEX

mRMR.ensemble (mRMRe.Filter-class), 16
mRMR.network (mRMRe.Network-class), 18
mRMR.Data-class, 14
mRMR.Filter-class, 16
mRMRe.Network-class, 18

priors, 20
priors, mRMR.e.Data-method (priors), 20
priors<- (priors), 20
priors<-., mRMR.e.Data-method (priors), 20

sampleCount, 21
sampleCount, mRMR.e.Data-method
  (sampleCount), 21
sampleCount, mRMR.Filter-method
  (sampleCount), 21
sampleCount, mRMR.e.Network-method
  (sampleCount), 21
sampleNames, 21
sampleNames, mRMR.e.Data-method
  (sampleNames), 21
sampleNames, mRMR.Filter-method
  (sampleNames), 21
sampleNames, mRMR.e.Network-method
  (sampleNames), 21
sampleStrata, 22
sampleStrata, mRMR.e.Data-method
  (sampleStrata), 22
sampleStrata<- (sampleStrata), 22
sampleStrata<-., mRMR.e.Data-method
  (sampleStrata), 22
sampleWeights, 23
sampleWeights, mRMR.e.Data-method
  (sampleWeights), 23
sampleWeights<- (sampleWeights), 23
sampleWeights<-., mRMR.e.Data-method
  (sampleWeights), 23
scores, 24
scores, mRMR.e.Data-method (scores), 24
scores, mRMR.Filter-method (scores), 24
scores, mRMR.e.Network-method (scores), 24
set_thread_count, 25
set_thread_count, 26
solutions, 26
solutions, mRMR.Filter-method
  (solutions), 26
solutions, mRMR.e.Network-method
  (solutions), 26
subsetData, 27
subsetData, mRMR.e.Data-method
  (subsetData), 27
target, 28
target, mRMR.Filter-method (target), 28
target, mRMR.e.Network-method (target), 28
visualize, 29
visualize, mRMR.e.Network-method
  (visualize), 29