Package ‘ddgraph’

April 5, 2014

Imports bnlearn (>= 2.8), gtools, pcalg, RColorBrewer, plotrix, MASS,Rcpp

Maintainer Robert Stojnic <robert.stojnic@gmail.com>

License GPL-3

Title Distinguish direct and indirect interactions with Graphical Modelling

LinkingTo Rcpp

Type Package

LazyLoad yes

Author Robert Stojnic

Description Distinguish direct from indirect interactions in gene regulation and infer combinatorial code from highly correlated variables such as transcription factor binding profiles. The package implements the Neighbourhood Consistent PC algorithm (NCPC) and draws Direct Dependence Graphs to represent dependence structure around a target variable. The package also provides a unified interface to other Graphical Modelling (Bayesian Network) packages for distinguishing direct and indirect interactions.

Version 1.6.3

biocViews Bioinformatics, GraphsAndNetworks

Date 2012-06-18

Depends graph, methods, Rcpp

Suggests testthat, Rgraphviz, e1071, ROCR, testthat

Collate 'AllClassses.R' 'AllGenerics.R' 'calcDependence.R' 'citest.R' 'combinations.R' 'dsep.R' 'furlong.R' 'methods-CITestResult.R' 'methods-DDDataSet.R' 'methods-DDGraphEdge.R' 'methods-DDGraph.R' 'misc.R' 'ncpc.R' 'other-algorithms.R' 'plot-colour.R' 'plot.R' 'randomnet.R' 'resampling.R' 'svm.R'

R topics documented:

`ddgraph-package` *ddgraph package overview*

Description

This package implements the Neighbourhood Consistent PC Algorithm (NCPC) for inferring the causal neighbourhood and Markov Blanket of a target variable, and a Direct Dependence Graphs (DDGraphs) for representing the conditional independence relationships.

The main goal of the NCPC algorithm is to infer direct from indirect dependencies of a set of variable to a target variable. The direct dependencies make up the causal neighbourhood of the target variable. This is achieved by performing conditional independence tests and therefore establishing statistical independence properties. NCPC has been shown to have a larger recall rate in scenarios with highly correlated variables which are weakly associated to a sparse target variable. For more details on the NCPC algorithm see (Stojnic et al, 2012).

Details

Package:	ddgraph
Type:	Package
License:	GPL-3
LazyLoad:	yes

This package implements the NCPC/NCPC* algorithms, but also provides a unified front-end for inferring causal neighbourhood and Markov Blanket via Bayesian Network inference as provided by packages `bnlearn` and `pcalg`.

The package comes with two example datasets (Zizen et al 2009):

- `mesoBin` - binary dataset with 7 target variables - cis-regulatory module (CRM) classes. The variable correspond to transcription factor (TF) binding profiles over 1-5 time intervals.
- `mesoCont` - the original continuous version of the dataset.

The main front-end function is `calcDependence()`.

Author(s)

Robert Stojnic
Cambridge Systems Biology Centre
University of Cambridge, UK

Maintainer: Robert Stojnic `<robert.stojnic@gmail.com>`

References

- R. Stojnic et al (2012): "A Graphical Modelling Approach to the Dissection of Highly Correlated Transcription Factor Binding Site Profiles", in press, PloS Computational Biology.

- R. Zinzen et al (2009): "Combinatorial binding predicts spatio-temporal cis-regulatory activity" Nature 462, no. 7269: 65-70.

`activePaths` *Find all active paths in a (partially) directed graph...*

Description

Find all active paths in a (partially) directed graph

Usage

```
activePaths(graph, node, nodeNames)
```

Arguments

`graph`	the graph either in one of the package `graph` classes, or of class `bn` or `pcAlgo`
`node`	the source node of the path (index not name)
`nodeNames`	optionally specify node names which can be used to return those instead of indicies

Value

a list of active paths with node as its source

`adjC.allVarInx` *Get all the variable indicies in adjC, both target and condSet...*

Description

Get all the variable indicies in adjC, both target and condSet

Usage

```
adjC.allVarInx(adjC)
```

Arguments

`adjC`	the adjC list of conditional independence tests for variables "adjacent" to target variable C

Value

numeric vector (unique values)

`adjC.allVarNames` *Get all the variable names in adjC, both target and condSet...*

Description

Get all the variable names in adjC, both target and condSet

Usage

```
adjC.allVarNames(adjC)
```

Arguments

`adjC`	the adjC list of conditional independence tests for variables "adjacent" to target variable C

Value

character vector (unique names)

`adjC.condSetSize` *Returns the total size of conditioning set for adjC (i...*

Description

Returns the total size of conditioning set for adjC (i.e. all variables present in adjC)

Usage

```
adjC.condSetSize(adjC)
```

Arguments

`adjC`	the adjC list of conditional independence tests for variables "adjacent" to target variable C

Value

sum of all conditioning set sizes plus size of adjC, i.e. all variables present in adjC

`adjC.targetInx` *Get all the targetInx values in adjC...*

Description

Get all the targetInx values in adjC

Usage

```
adjC.targetInx(adjC)
```

Arguments

`adjC`	the adjC list of conditional independence tests for variables "adjacent" to target variable C

Value

numeric vector (unique values)

`adjC.toIDs` *Make a list of conditional independence tests and converts them to IDs...*

Description

Make a list of conditional independence tests and converts them to IDs

Usage

```
adjC.toIDs(adjC)
```

Arguments

`adjC`	a list of conditional independence tests

`biased.bn.fit` *Random network with a biased degree distribution*

Description

A version of random.bn.fit which generates a graph based on degree distribution and beta distribution for probabilities

Usage

```
biased.bn.fit(nodes, beta.est, in.degree.distr, bn.graph)
```

Arguments

`nodes`	character vector of node names
`beta.est`	the beta distribution parameters for different degrees of a node. Should be a list where [[2]] corresponds to 2-dimenstional contingency table (i.e. one parent, one output). It contains a data.frame with columns shape1, shape2 for the beta distribution, and rows are degrees of freedom (in this case 2, when P(Out=0|Parent=0) and P(Out=0|Parent=1))
`in.degree.distr`	a vector with degree distribution for all the nodes in the network (names are ignored, and degree is randomly sampled from this vector)
`bn.graph`	if the graph structure is already available, then the graph structure in object of class "bn"

Value

a list of two elements: `bn` - a `bn` object which contains the structure and `bn.fit` - a `bn.fit` object with filled in conditional probabilities

Examples

```
# nodes, conditional probability distribution, an indegree distribution
nodes = letters[1:5]
beta.est = list(data.frame(shape1=2,shape2=3), data.frame(shape1=c(2,4), shape2=c(5,2)), data.frame(shape1=c(1,2
in.degree.distr = c(0, 1, 1, 2, 2)
# make a random graph using these parameters
biased.bn.fit(nodes, beta.est, in.degree.distr)
```

`biased.graph` *Generate random network with degree distribution*

Description

Generate a random directed graph with the given node ordering and degree distribution

Usage

```
biased.graph(nodes, in.degree.distr)
```

Arguments

`nodes`	character vector of node names which species the node ordering
`in.degree.distr`	the node in-degree distribution

Value

an object of class bn with the random graph

Examples

```
# a random network of 5 nodes with choosen in-degree distribution
biased.graph(letters[1:5], c(0, 1, 1, 2, 2))
```

`blockingNodes` *Find all such nodes in neighbourhood of source node that are blocking at least one active path leading to another node...*

Description

Find all such nodes in neighbourhood of source node that are blocking at least one active path leading to another node

Usage

```
blockingNodes(allPaths, nodes)
```

Arguments

`allPaths`	a list of active paths from a source node (as produced by activePaths())
`nodes`	a vector of target nodes for which we are finding blocking nodes

Value

a list with blocking nodes and minimal length to the target node: target node => blocked by => number of steps

---

`blockingVariables` *Version of blockingNodes() for DDGraphs...*

Description

Version of blockingNodes() for DDGraphs

Usage

```
blockingVariables(obj, nodes)
```

Arguments

`obj`	DDGraph object
`nodes`	the selected nodes

Value

same as blockingNodes(): a list with blocking nodes and minimal length to the target node: target node => blocked by => number of steps

---

`calcDependence` *Dependence with target variable*

Description

Calculate dependence with a target variable

Usage

```
calcDependence(dd, method="ncpc", ...)
```

Arguments

`dd`	An object of type DDDataSet
`method`	Algorithm to use. Valid values are: • ncpc - Neighbourhood Consistent PC algorithm • ncpc* - Neighbourhood Consistent PC algorithm star version • hc - Hill-climbing with custom penalty functions • hc-bic - Hill-climbing with BIC penalization (package bnlearn) • hc-bde - Hill-climbing with BDe penalization (package bnlearn) • iamb - IAMB algorithm (package bnlearn) • fast.iamb - FastIAMB algorithm (package bnlearn) • inter.iamb - InterIAMB algorithm (package bnlearn)

- pc - PC algorithm (package pcalg)
- mmpc - MMPC algorithm (package bnlearn)
- mmhc - MMHC with custom penalty functions
- mmhc-bic - MMHC with BIC penalization (package bnlearn)
- mmhc-bde - MMHC with BDe penalization (package bnlearn)

`...` Extra parameters passed to backend functions `ncpc()`, `plotBNLearn()` and `plotPCAlgo()` depending on the picked algorithm (parameter `method`).

Extra parameters for ncpc and ncpc*:

- alpha - the alpha (P-value) cutoff for conditional independence tests (default: 0.05)
- p.value.adjust.method - the multiple testing correction adjustment method (default: "none")
- test.type - the type of conditional independence test (default: "mc-x2-c"). See the documentation for `ciTest` for available conditional independence tests
- max.set.size - the maximal number of variables to condition on, if NULL estimated from number of positives in class labels. Needs to be specified for continuous data. (default: NULL)
- mc.replicates - the number of Monte-Carlo replicates for the conditional independence test, if applicable (default: 5000)
- report.file - name of the file where a detailed report is to be printed, reporting is suppressed if NULL (default: NULL)
- verbose - if to print out information about how the algorithm is progressing (default: TRUE)
- min.table.size - the minimal number of samples in a contingency table per conditioning set (applicable only for discrete data) (default: 10)

Extra parameters for hc, mmhc:

- score - score function to use, accepts all from bnlearn package. For discrete data: "loglik", "aic", "bic", "bde", "k2". For continuous: "loglik-g", "aic-g", "bic-g", "bge". For more details see help page for `package-bnlearn`.
- make.plot - if to make a plot or just return the network (default: FALSE)
- blacklist - a data frame with two columns (optionally labeled "from" and "to"), containing a set of arcs not to be included in the graph. (default: NULL)
- restart - the number of random restarts for score-based algorithms (default: 0)
- scale - the colour scaling (default: 1.5)
- class.label - the label to use for the target variable (default: "target")
- use.colors - if to colour code the enrichment/depletion in a plot (default: TRUE)

Extra parameters for hc-bic, hc-bde, mmhc-bic, mmhc-bde:

- make.plot - if to make a plot or just return the network (default: FALSE)
- blacklist - a data frame with two columns (optionally labeled "from" and "to"), containing a set of arcs not to be included in the graph. (default: NULL)

- restart - the number of random restarts for score-based algorithms (default: 0)
- scale - the colour scaling (default: 1.5)
- class.label - the label to use for the target variable (default: "target")
- use.colors - if to colour code the enrichment/depletion in a plot (default: TRUE)

Extra parameters for iamb, fast.iamb, inter.iamb, mmpc:

- make.plot - if to make a plot or just return the network (default: FALSE)
- alpha - the alpha value of conditional independence tests (default: 0.05)
- test - the type of conditional independence test (default: "mc-mi"). For conditional independence tests available consult the bnlearn package help page (?bnlearn).
- B - the number of Monte-Carlo runs for conditional independence tests, if applicable (default: 5000)
- blacklist - a data frame with two columns (optionally labeled "from" and "to"), containing a set of arcs not to be included in the graph. (default: NULL)
- scale - the colour scaling (default: 1.5)
- class.label - the label to use for the target variable (default: "target")
- use.colors - if to colour code the enrichment/depletion in a plot (default: TRUE)

Extra parameters for pc:

- alpha - the alpha value cut-off for the conditional independence tests (default: 0.05)
- verbose - if to show progress (default: FALSE)
- directed - if TRUE applies PC algorithm, if FALSE applies PC-skeleton (default: TRUE)
- make.plot - if to make a plot of the final inferred network (default: FALSE)
- scale - the scaling parameter for color-coding (default: 1.5)
- indepTest - the independence test wrapper function (default: mcX2Test). The following functions are available: `mcX2Test` (a wrapper around mc-x2-c (Monte Carlo X2 test) with B=5000), `mcX2TestB50k` (a wrapper around mc-x2-c (Monte Carlo X2 test) test with B=50000), `mcMITest` (wrapper around mc-mi test from `bnlearn` with B=5000). The package `pcalg` additionally provide following tests: `binCItest` for binary data (performs a G^2 test) and `gaussCItest` for continuous data (performs Fisher's Z transformation), `dicCItest` for discrete data (performs G^2 test).
- class.label - the label to show for target variable (default: "target")
- use.colors - if to colour code the results (default: TRUE)

Details

This function is a front-end convenience function to access predictions of direct dependence with a target variable by various Graphical Modelling algorithm.

Consider a set of variable X_1, ..., X_m and a target variable T. We say that that X_i is directly dependent with T if there is no other set of variable X_j, X_k, ... such that it renders X_i conditionally independent of T. In other words, X_i is the most immediate casual cause/consequence of T in the set of chosen variables.

Note that the above statement is different from that of classical feature selection for classification. A set of features obtained with feature selection have the property that a good classifier can be made based on them alone, while the above statement establishes statistical properties of variables. The set of variables with direct dependence might not be optimal for classification, since classification performance can be strongly influenced by false negatives (Friedman et al, 1997).

Value

A list with elements:

- obj - the resulting object, either of class `DDGraph` for ncpc and ncpc* algorithms, or of class `bn` for `bnlearn` algorithms, or of class `pcAlgo` for PC algorithm.
- nbr - the variables with direct dependence (i.e. target node neighbourhood in the causal graph). For both ncpc and ncpc* includes variables with direct and joint dependence.
- mb - the variables in Markov Blanket of target variable. Not applicable for ncpc algoritm. For ncpc* algorithm includes variables with direct, joint and conditional dependence.
- labels - for ncpc and ncpc* contains the set of labels that are output of the algorithm.

References

Nir Friedman, Dan Geiger, and Moises Goldszmidt, "Bayesian Network Classifiers", Machine Learning 29 (November 1997): 131-163.

Examples

```
# load in the data for fly mesoderm
data(mesoBin)

# increase alpha to 0.1, suppress progress output
calcDependence(mesoBin$VM, "ncpc", alpha=0.05)

# run ncpc* with mutual information with shrinkage and minimal numbers of
# samples per conditioning set of 15
calcDependence(mesoBin$VM, "ncpc*", test.type="mi-sh", min.table.size=15)

# run PC algorithm using the G^2 test from pcalg package
calcDependence(mesoBin$VM, "pc", indepTest=pcalg::binCItest)

# run hill-climbing with BIC penalization and plot the resulting Bayesian Network
# NOTE: plotting requires the Rgraphviz package
if(require("Rgraphviz"))
calcDependence(mesoBin$VM, "hc-bic", make.plot=TRUE)

# continuous data example
data(mesoCont)
```

```
# run ncpc with linear correlation test and with maximal conditioning set of 3
res <- calcDependence(mesoCont$VM, "ncpc", max.set.size=3, test.type="cor")
# plot the resulting ddgraph with colours
if(require("Rgraphviz"))
plot(res$obj, col=TRUE)
```

`calculateNCPCRobustnessStats` *Calculate NCPCRobustness statistics...*

Description

Calculate NCPCRobustness statistics

Usage

```
calculateNCPCRobustnessStats(obj)
```

Arguments

`obj`	NCPCRobustness object

Details

Calculate the statistics for the NCPCRobustness object - this is separate from object construction for convenience of testing, should always be called after object creation. Never use directly (except for testing), use instead via `DDDataSet::NCPCRobustness()`.

Value

the modified NCPCRobustness object with the statistics calculated

---

`chisq.val` *Get the value of chi-square statistics...*

Description

Get the value of chi-square statistics

Usage

```
chisq.val(x, correct=FALSE)
```

Arguments

`x`	is the contingency table
`correct`	if to do the Yates correction

Value

chisq statistics

`ciTest,DDDataSet-method`

Do conditional independence test on DDDataSet...

Description

Do conditional independence test on DDDataSet

Usage

```
## S4 method for signature DDDataSet
ciTest(obj, var1, var2, cond, test.type="mc-x2-c", B, min.table.size, ...)
```

Arguments

`obj`	DDDataSet object on which (conditional) independence test needs to be done
`var1`	the name or index of the first variable to be tested
`var2`	the name or index of the second variable
`cond`	the names or indexes of variables to condition on (defaults to NULL)
`test.type`	the type of statistical test (defaults to mc-x2)
`B`	the number of replicates for MC-based tests (default to NULL)
`min.table.size`	the minimal number of samples in a contingency table per conditioning set (makes sense only for discrete data)
`...`	unused

Details

This function does a conditional independence var1 indep var2 | cond. The following test types are available (implemented by package `bnlearn`).

For binary data:

- "fisher" - Fisher's exact test (only for unconditional independence)
- "mi" - Mutual Information (discrete)
- "mi-sh" - Mutual Information (discrete, shrinkage)
- "mc-mi" - Mutual Information (discrete, Monte Carlo)
- "aict" - AIC-like Test
- "x2" - Pearson's X^2
- "mc-x2" - Pearson's X^2 (Monte Carlo)
- "mc-x2-c" - Pearson's X^2 (Monte Carlo) the corrected version

- "g2" - G^2 test (requires pcalg package)

For continuous data:

- "mi-g" - Mutual Information (Gaussian)
- "mi-g-sh" - Mutual Information (Gaussian, shrinkage)
- "mc-mi-g" - Mutual Information (Gaussian, Monte Carlo)
- "cor" - Pearson's Linear Correlation
- "mc-cor" - Pearson's Linear Correlation (Monte Carlo)
- "zf" - Fisher's Z Test
- "mc-zf" - Fisher's Z Test (Monte Carlo)

Value

CITestResult object with the result of the test

Examples

```
data(mesoBin)
# test if tin_4.6 is independent of class labels
ciTest(mesoBin$Meso, "Tin 4-6h", "class")
# test if tin_4.6 is independent of class conditioned on twi_2.4
ciTest(mesoBin$Meso, "Tin 4-6h", "class", "Twi 2-4h")
# repeat the test using G2 asymptotic distribution
ciTest(mesoBin$Meso, "Tin 4-6h", "class", "Twi 2-4h", test.type="g2")
```

`CITestResult-class` *Data class to store the results of a conditional independence test...*

Description

Data class to store the results of a conditional independence test

Details

This class stored the results from `DDDataSet::ciTest()`. It stores the indexes and names of two variables involved in the test, the conditioning set as well as the P-value and type of test.

Slots

`targetInx`: (`numeric`) the index of the first variable

`targetName`: (`character`) the name of the first variable

`sourceInx`: (`numeric`) the index of the second variable

`sourceName`: (`character`) the name of the second variable

`condSetInx`: (`numeric`) the indexes of variables we condition on `condSetName`: (`character`) the names of variables we condition on

`pValue`: (`numeric`) the associated p value

`testType`: (`character`) the type of the conditional independence test performed

`reliable`: (`logical`) if this appears to be a reliable test of conditional independence

Methods

`$` `signature(x = "CITestResult")`: Access slots using the dollar notation

`[[` `signature(x = "CITestResult", i = "ANY", j = "ANY")`: Access slots using the double square bracket notation

`names` `signature(x = "CITestResult")`: Names of slots that can be accessed with $ notation

`show` `signature(object = "CITestResult")`: show method for CITestResult

`CITestResultID` *Provide a unique ID composing of target, source and conditioning set (all names)...*

Description

Provide a unique ID composing of target, source and conditioning set (all names)

Usage

```
CITestResultID(citest)
```

Arguments

`citest`	a CITestResult object

Value

a character ID

`CITestResultVar` *Return a string representation of a variable represented with this CITest...*

Description

Return a string representation of a variable represented with this CITest

Usage

```
CITestResultVar(citest)
```

Arguments

`citest`	an object of class CITestResult

`classLabels,FurlongDataSet-method` *Class labels*

Description

Retrieve the vector of class labels (as factors)

Usage

```
## S4 method for signature FurlongDataSet
classLabels(object)
```

Arguments

`object`	FurlongDataSet object

`color.legend.DDGraph` *Plot color coding legend*

Description

This function is a slightly modified version of function `color.legend()` function from plotrix package. It plots a color legend at the given coordinates. This version extends the original plotrix function with additional label and ability to plot into margins.

Usage

```
color.legend.DDGraph(xl, yb, xr, yt, legend, rect.col, cex=1, align="lt", gradient="x",
    title="", ...)
```

Arguments

`xl`	lower left corner x coordinate
`yb`	lower left corner y coordinate
`xr`	upper right corner x coordinate
`yt`	upper right corner y coordinate
`legend`	the text to be plotted below the color coding rectangle
`rect.col`	the color that will fill the rectangle
`cex`	character expansion factor for the labels
`align`	how to align the labels relative to the color rectangle
`gradient`	whether to have a horizontal (x) or vertical (y) color gradient
`title`	the title to be printed above the color coding rectangle
`...`	the additional arguments passed to `text()`

`combinationsTest` *Significant combinations of variables*

Description

Calculate which combinations of values of variables are significantly different in the two classes (only for binary data). This function takes an DDDataSet and a number of variables and finds those combinations of values of those variables that have significantly different frequencies in the two class labels.

Usage

```
combinationsTest(obj, selected.vars, cutoff=0.05, p.adjust.method="none", verbose=TRUE)
```

Arguments

`obj`	DDDataSet object
`selected.vars`	indexes or names of variables selected for the test
`cutoff`	the p value cutoff for reporting (default: 0.05)
`p.adjust.method`	the multiple adjustment method (default: none)
`verbose`	if to print progress output and additional information

Value

`data.frame` with ordered combinatorial patterns of selected variables

Examples

```
data(mesoBin)
# find significant differences at 0.2 FDR
combinationsTest(mesoBin$Meso, c("Twi 2-4h", "Tin 6-8h", "Mef2 6-8h"), 0.2, "fdr")
```

`convertPvalueToColorIndex`
Convert P-values to color index...

Description

Convert P-values to color index

Usage

```
convertPvalueToColorIndex(p.vals, scale="auto", max.color.index, minimal.p.value=1e-04)
```

Arguments

`p.vals`	the P-values (after any multiple testing correction)
`scale`	the color is calculated liked -log10(p.value) * scale, thus scale is used to scale the -log10 to the desired range. Either a number or "auto" for automatic
`max.color.index`	the maximal color index to return
`minimal.p.value`	the minimal P-value we accept (since from Monte Carlo we can get 0)

Details

Convert p values to a color index to color nodes in a graph. The P-values are fit into a range from 1 to `max.color.index` by applying a scale. Before fitting, P-value are transformed by taking a log10, and a minimal P-value is needed to avoid -Inf results for very small P-values. Scale can either be a number or "auto" in which case color coding is such that all P-values fit into the range.

Value

a `list` with following elements: `col` - the color indexes, `zlim` - the actual scale range (in log10) over the colors

Examples

```
# scale the P values into the log10 space of [1e-3,1] represented by max 6 colours
convertPvalueToColorIndex(c(0.01, 0.2, 0.3), scale="auto", max.color.index=6, minimal.p.value=1e-3)
```

`convertToFactor` *Convert data to factor representation*

Description

Convert a matrix, dataframe or vector into a factor representation. Each column is going to be separately converted into a factor.

Usage

```
convertToFactor(x)
```

Arguments

`x`	the input vector, data.frame or matrix

Examples

```
# works on vectors, matrices and data frames
convertToFactor(0)
convertToFactor(c(1, 0, 0, 1, 0))
convertToFactor(matrix(c(1,0), nrow=2, ncol=2))
convertToFactor(data.frame("a"=c(1,0), "b"=c(0,1)))
```

`customPlotPCAlgo` *Custom plotting for pcalgo*

Description

Custom plotting function of PC algorithm to have nice highlighting

Usage

```
customPlotPCAlgo(x, main, labels, colors, ...)
```

Arguments

`x`	an object of one of the pcalg classes
`main`	the main title
`labels`	the labels of each of the nodes (doesn't need to be a named vector)
`colors`	the colors we want to assign to each node (doesn't need to be a named vector)
`...`	additional parameters to pass to `layoutGraph()`

`datasetName,DDDataSet-method`
Dataset name...

Description

Dataset name

Usage

```
## S4 method for signature DDDataSet
datasetName(obj, ...)
```

Arguments

`obj`	the DDDataSet object
`...`	unused

Value

the name of the dataset used in plotting etc

`dataType,DDDataSet-method` *Return data type*

Description

Return the data type ("binary" or "continuous")

Usage

```
## S4 method for signature DDDataSet
dataType(obj, ...)
```

Arguments

`obj`	the DDDataSet object
`...`	unused

Value

the data type

`DDDataSet-class` *Dataset class for Direct Dependence Graphs...*

Description

Dataset class for Direct Dependence Graphs

Details

This is the main class to hold data to be used in Direct Dependence Graphs. The data is stored in a `data` data.frame with the last column named "class". Dataset can be either binary, or continuous. Mixtures of binary and continuous variables are currently not supported.

Slots

`name`: (`character`) a descriptive name of this dataset used as caption for graphs, etc

`data`: (`data.frame`) data.frame containing the variables as columns, and the special column "class" as last column

`dataType`: (`character`) either "binary" or "continuous" are supported, indicated the type of variables present (all need to be either binary or continuous)

Methods

- `ciTest` `signature(obj = "DDDataSet")`: Do conditional independence test on DDDataSet
- `rawData` `signature(obj = "DDDataSet")`: Return the raw data frame with the variables, and the last column being "class"
- `dataType` `signature(obj = "DDDataSet")`: Return the data type ("binary" or "continuous")
- `datasetName` `signature(obj = "DDDataSet")`: Dataset name
- `names` `signature(x = "DDDataSet")`: Names of variables (including "class")
- `variableNames` `signature(obj = "DDDataSet")`: Names of variables (without "class")
- `$` `signature(x = "DDDataSet")`: access a specific variable in the dataset by name
- `[[` `signature(x = "DDDataSet")`: access a specific variable in the dataset by name
- `[` `signature(x = "DDDataSet", i = "ANY", j = "ANY")`: access a specific variable in the dataset by name
- `initialize` `signature(.Object = "DDDataSet")`: Construct new DDDataSet object
- `show` `signature(object = "DDDataSet")`: show method for DDDataSet

`DDGraph-class` *Direct Dependence Graph class...*

Description

Direct Dependence Graph class

Arguments

`dataset`	the DDDataSet object used to make the DDGraph
`params`	the parameters used in making the DDGraph
`stats`	the values of statistics used to make the DDGraph
`direct`	the list of indicies of direct variables
`indirect`	the list of indicies of indirect variables
`joint`	the list of indicies of joint variables
`conditional`	the list of indicies of conditional variables
`conditionalJoint`	the list of indicies of conditionally joint variables
`edges`	the list of edges (type DDGraphEdge) the describe the graph

Details

This class represents one Direct Dependence Graphs (generated by a certain conditional independence test, alpha value, etc). It contains the original `DDDataSet` object from which it stems, the set of parameters, the set of informative statistics as well as lists of direct, joint and indirect variables. Finally, it contains the edges needed to draw the graph.

Methods

- `initialize` `signature(.Object = "DDGraph")`: Construct new DDGraph object
- `names` `signature(x = "DDGraph")`: Names of different properties that can be accessed with $ operator
- `$` `signature(x = "DDGraph")`: access a property by name
- `show` `signature(object = "DDGraph")`: show method for DDGraph
- `plot` `signature(x = "DDGraph", y = "missing")`: Plot DDGraphs using RGraphviz

`DDGraphEdge-class` *An edge in an DDGraph...*

Description

An edge in an DDGraph

Details

This class represents an edge in an Direct Dependence Graph. It is normally found in the `DDGraph::edges` list. It records the source and target nodes for the edge, the edge type, as well as the conditional independence tests it represents.

Slots

- **`fromInx`:** (`numeric`) the index of the first variable from which the edge goes
- **`fromName`:** (`character`) the name of the first variable from which the edge goes
- **`toInx`:** (`numeric`) the index of the second variable to which the edge goes
- **`toName`:** (`character`) the name of the second variable to which the edge goes
- **`ciTests`:** (`list`) a list of associated `CITestResult` objects
- **`type`:** (`character`) type of edge: "directed", "undirected", "bidirectional", "dashed"

Methods

- `show` `signature(object = "DDGraphEdge")`: show method for DDGraphEdge

`entropyFromFreq` *Calculate entropy from frequencies of observations for discrete data...*

Description

Calculate entropy from frequencies of observations for discrete data

Usage

```
entropyFromFreq(x)
```

Arguments

`x`	the vector of frequencies, or a pdf of distribution

Value

the entropy in bits

`estimateNetworkDistribution`
Estimate network distribution parameters

Description

Estimate the in.degree distribution and conditional probability distribution from data

Usage

```
estimateNetworkDistribution(obj, use.class=FALSE)
```

Arguments

`obj`	and object of class DDDataSet
`use.class`	if to include the class variable into the estimate

Details

The algorithm uses hill-climbing with BIC to construct the network and estimate the parameters. Then, provided that for each in-degree there is at least two nodes, it estimates the beta distribution parameters.

Value

a list of two elements: in.degree.distr - distribution of in-degrees, and beta.est - estimate beta distribution values

Examples

```
data(mesoBin)
estimateNetworkDistribution(mesoBin$Meso)
```

`extract.targetInx` *Extract all values of targetInx from a list of CITestResult...*

Description

Extract all values of targetInx from a list of CITestResult

Usage

```
extract.targetInx(adjC)
```

Arguments

`adjC`	a list of CITestResult

`extractCITestResultProperty` *Extract CITestResult properties*

Description

This is a helper function for `DDDataSet::ncpc()`. From a list of ciTestResult object extract a list containing only one property.

Usage

```
extractCITestResultProperty(ciTestList, prop.name)
```

Arguments

`ciTestList`	a two-level list of ciTestResult objects
`prop.name`	the name of the property to extract (one of the slot names)

Value

a vector with the extracted property

`foldChangeFromFreq` *Calculate the fold change when x is of size two (always show it >1)...*

Description

Calculate the fold change when x is of size two (always show it >1)

Usage

```
foldChangeFromFreq(x)
```

Arguments

`x`	input vector of size two

Value

the proportion of x[1]/x[2] or x[2]/x[1] depending which is larger

---

`formulaFalseNeg` *Generate class labels by a noisy formula with high false negative rate*

Description

Generate class labels by using the readout mechanism. Logical formula is applied to two variables which are read out from the real data using the var1 and var2 probabilities. This only works with binary variables.

Usage

```
formulaFalseNeg(data, var1, var2, false.neg, logical.formula)
```

Arguments

`data`	a matrix or data.frame containing binary observations (columns are variables)
`var1`	index or name of the first variable
`var2`	index or name of the second variable
`false.neg`	a false negative probability
`logical.formula`	logical formula to apply

Value

a binary vector containing the class labels

Examples

```
# noisy OR function with 0.1 probability of error for reading "a" and "b" (error in both 1 and 0)
data <- cbind("a"=c(0,0,1,1), "b"=c(0,1,0,1))
formulaFalseNeg(data, "a", "b", 0.8, "a | b")
```

`FurlongDataSet-class` *Data class for the Furlong dataset...*

Description

Data class for the Furlong dataset

Details

A class to hold data from (Zizen 2009) paper (Supp Table 8). This class contains methods to convert it to both binary and continuous DDDataSet objects.

Slots

`signalMatrix`**:** (`matrix`) the signal matrix

`targetClasses`**:** (`factor`) the target class names

Methods

- `names` `signature(x = "FurlongDataSet")`: Get the names of variables (column names of signal matrix)
- `signalMatrix` `signature(object = "FurlongDataSet")`: Retrieve the matrix with raw signal values
- `classLabels` `signature(object = "FurlongDataSet")`: Retrieve the vector of class labels (as factors)
- `toDDDataSet` `signature(obj = "FurlongDataSet")`: Make the DDDataSet objects by selecting different tissues

References

Robert P. Zinzen et al., "Combinatorial binding predicts spatio-temporal cis-regulatory activity," Nature 462, no. 7269 (November 5, 2009): 65-70.

`graph.to.bn` *Convert graphNEL and friends representation to bn...*

Description

Convert graphNEL and friends representation to bn

Usage

```
graph.to.bn(graph)
```

Arguments

`graph`	graphNEL or graphAM object

`independent.contributions.formula` *Generate class labels by independent contributions of two variables*

Description

Generate class labels by using the readout mechanism. Logical formula is applied to two variables which are read out from the real data using the var1 and var2 probabilities. This only works with binary variables.

Usage

```
independent.contributions.formula(data, var1, var2, var1.prob1, var1.prob0, var2.prob1, var2.prob0,
    logical.formula, false.neg=0, false.pos=0)
```

Arguments

`data`	a matrix or data.frame containing binary observations (columns are variables)
`var1`	index or name of the first variable
`var2`	index or name of the second variable
`var1.prob1`	the conditional probability P(class labels = 1|var1=1)
`var1.prob0`	the conditional probability P(class labels = 1|var1=0)
`var2.prob1`	the conditional probability P(class labels = 1|var2=1)
`var2.prob0`	the conditional probability P(class labels = 1|var2=0)
`logical.formula`	logical formula to apply
`false.neg`	a false negative probability
`false.pos`	a false positive probability

Value

a binary vector containing the class labels

Examples

```
# noisy OR function with 0.1 probability of error for reading "a" and "b" (error in both 1 and 0)
data <- cbind("a"=c(0,0,1,1), "b"=c(0,1,0,1))
independent.contributions.formula(data, "a", "b", 0.9, 0.1, 0.9, 0.1, "a | b")
```

`independent.contributions.formula.mul` *Generate class labels by independent contributions of two variables*

Description

Version of `independent.contributions.formula` that works with any number of variables. See the help page for `independent.contributions.formula` for description of functionality.

Usage

```
independent.contributions.formula.mul(data, target.vars, prob1, prob0, logical.formula)
```

Arguments

`data`	a matrix or data.frame containing binary observations (columns are variables)
`target.vars`	indexes of target variables
`prob1`	vector of P(class labels = 1|varX=1) for different X
`prob0`	vector of P(class labels = 1|varX=0) for different X
`logical.formula`	a character string for the formula

Value

a vector of binary class labels

Examples

```
# noisy OR function with three variables and with noise level of 0.1 for a, b, and 0.2 for c
data <- cbind("a"=c(0,0,0,0,1,1,1,1), "b"=c(0,0,1,1,0,0,1,1), "c"=c(0,1,0,1,0,1,0,1))
independent.contributions.formula.mul(data, c("a", "b", "c"), c(0.9, 0.9, 0.8), c(0.1, 0.1, 0.2), "a | b | c")
```

`initialize,DDDataSet-method`

Construct new DDDataSet object...

Description

Construct new DDDataSet object

Usage

```
## S4 method for signature DDDataSet
initialize(.Object, ..., data=data.frame(), name=paste("Empty name created at",
    date()))
```

Arguments

`.Object`	the DDDataSet object
`data`	the data slot
`name`	the name slot
`...`	unused

Details

Try to initialise with anything that can be converted to matrix and vectors.

`initialize,DDGraph-method`

Construct new DDGraph object...

Description

Construct new DDGraph object

Usage

```
## S4 method for signature DDGraph
initialize(.Object, ..., direct=vector(mode = "numeric"), indirect=vector(mode =
    "numeric"), joint=vector(mode = "numeric"), conditional=vector(mode
    = "numeric"), conditionalJoint=vector(mode = "numeric"),
    edges=list(), dataset=new("DDDataSet"), params=list(), stats=list())
```

Arguments

`.Object`	DDGraph object
`direct`	direct variable indexes
`indirect`	indirect variable indexes
`joint`	joint variable indexes
`conditional`	conditional variable indexes
`conditionalJoint`	conditionally joint variable indexes
`edges`	edges list
`dataset`	DDDataSet object
`params`	parameters used to make this object
`stats`	the statistics used to make this object
`...`	unused

Details

Properly initialize the object

`is.binary` *Check if data structure has binary data in it*

Description

Check if a vector, data frame or matrix contains only binary (0,1) values.

Usage

```
is.binary(x)
```

Arguments

`x`	the input vector, data.frame or matrix

Value

boolean TRUE or FALSE

Examples

```
# works on vectors, matrices and data frames
is.binary(0)
is.binary(c(1, 0, 0, 1, 0))
is.binary(matrix(c(1,0), nrow=2, ncol=2))
is.binary(data.frame("a"=c(1,0), "b"=c(0,1)))

# returns FALSE if not binary
is.binary(c(1, 2, 3))
```

`logseq` *Generate sequence in log scale*

Description

Generate sequence but in log scale. This function takes takes the length of log-sequence and the minimal and maximal point. It returns the interval between `a` and `b` divided in log scale.

Usage

```
logseq(a, b, n=8)
```

Arguments

`a`	the smaller value in the interval
`b`	the bigger value in the interval
`n`	the number of intervals to divide a,b into

Value

a vector of numbers

Examples

```
# produces vector c(0.01, 0.1, 1)
logseq(0.01, 1, 3)
```

`loocv` *Leave-one-out cross validation*

Description

Leave-one-out cross validation systematically leaves out one row from the data, retrains the classifier and then uses the retrained classifier to make a prediction for the left-out row.

Usage

```
loocv(data, train.fun, eval.fun, verbose=FALSE)
```

Arguments

`data`	The `data.frame` with data. Columns are variables, rows are observations.
`train.fun`	The training function that takes the data without one of the rows left out.
`eval.fun`	The prediction function that takes the trained model and the left out data point.
`verbose`	If to print progress indication

Value

A vector of length `nrow(data)` containing predictions from `eval.fun` when each row is left out once

---

`makeDDDataSet` *Construct an DDDataSet object...*

Description

Construct an DDDataSet object

Usage

```
makeDDDataSet(signal, name, classLabels, classLabelsCol, removeZeroVar=FALSE)
```

Arguments

`signal`	the matrix or data frame where rows are observations and columns variables
`name`	the name of the dataset (to be used in plotting, etc)
`classLabels`	the vector of class labels or target responses (aka target variable)
`classLabelsCol`	the column which should be interpreted as class labels (either name or index)
`removeZeroVar`	if to remove zero variance columns without producing an error (default: TRUE)

Value

a new DDDataSet object

Examples

```
# columns are features, rows observations
data <- matrix(rbinom(50, 1, 0.5), ncol=5)
# target class labels
labels <- c(0, 0, 0, 0, 0, 1, 1, 1, 1, 1)
makeDDDataSet(data, name="example data", classLabels=labels)
```

`makeNCPCRobustness` *Make a new NCPCRobustness object...*

Description

Make a new NCPCRobustness object

Usage

```
makeNCPCRobustness(dataset, raw, params)
```

Arguments

`dataset`	the DDDataSet object
`raw`	the list of raw resampling classification of variables (direct, joint, etc..)
`params`	the parameters used to generate the data (only the non-default one are listed)

Details

Make a new NCPCRobustness object just with the raw resampling data and parameters used to generate them. Should never directly use this function, but only via `DDDataSet::NCPCRobustness()`.

Value

a new NCPCRobustness object

`mapEnrichmentToColors` *Map enrichment values to colors...*

Description

Map enrichment values to colors

Usage

```
mapEnrichmentToColors(obj, palette, class.col, scale="auto")
```

Arguments

`obj`	an object of type DDGraph
`palette`	the color palette to use (by default Orange-Red)
`class.col`	the color to use for class labels, if applicable (by default light green)
`scale`	by how much to scale the -log10(p.value) when color coding: either a number of "auto" for automatic

Details

The enrichment of every variable is calculated during construction of DDGraph objects (in `ncpc()`). Use this information to color code the node in the graph. By default the Orange-Red is used and shown the strength of enrichment and depletion. No difference is made for enriched/depleted variables.

Value

the p values color-coded by convertPvalueToColorIndex() function

---

`mapEnrichmentToColorsDual` *Map enrichment values into two different palettes for enriched/depleted variables...*

Description

Map enrichment values into two different palettes for enriched/depleted variables

Usage

```
mapEnrichmentToColorsDual(obj, palette.pos, palette.neg, class.col, scale="auto")
```

Arguments

`obj`	an object of type DDGraph
`palette.pos`	the palette to use for enrichment (by default Orange-Red)
`palette.neg`	the palette to use for depletion (by default Purple-Blue)
`class.col`	the colour to use for class labels, if applicable (by default light green)
`scale`	by how much to scale the -log10(p.value) when color coding

Value

the p values color-coded by convertPvalueToColorIndex() function

Examples

```
## Not run:
data(mesoBin)
meso <- ncpc(mesoBin$Meso)
# use heat colours for both enrichment and depletion
mapEnrichmentToColorsDual(meso, palette.pos=heat.colors(10), palette.neg=heat.colors(10))

## End(Not run)
```

`mcMITest` *Wrapper around the bnlearn mc-x2 test*

Description

Implements the mc-mi test in format needed for pcalg.

Usage

```
mcMITest(x, y, S, suffStat)
```

Arguments

`x`	the index of the first variable
`y`	the index of the second variable
`S`	the conditioning set
`suffStat`	the sufficient statistics to do the test, in this case a list of one element: dm where the values matrix is stored

Value

p value of the test

Examples

```
suffStat <- list(dm = cbind("a"=c(0,1,0,0,1,0), "b"=c(1,0,0,0,1,0), "c"=c(0,0,0,1,1,1)))
# test if a is independent of b
mcMITest(1, 2, NULL, suffStat)
# test if a is independent of b conditioned on c
mcMITest(1, 2, 3, suffStat)
```

`mcX2CLoop` *the inner loop for myX2c is implemented in C...*

Description

the inner loop for myX2c is implemented in C

Usage

```
mcX2CLoop(B, numTable, rowSums, colSums)
```

Arguments

`B`	the number of Monte Carlo replicates
`numTable`	the number of conditional tables
`rowSums`	the matrix or row sums for each conditional table (numTables x 4)
`colSums`	the matrix or column sums for each conditional table (numTables x 4)

Value

The values of chi-square statistics from random runs

`mcX2Test` *Wrapper around the bnlearn mc-x2 test*

Description

Implements the mc-x2 test in format needed for pcalg.

Usage

```
mcX2Test(x, y, S, suffStat)
```

Arguments

`x`	the index of the first variable
`y`	the index of the second variable
`S`	the conditioning set
`suffStat`	the sufficient statistics to do the test, in this case a list of one element: dm where the values matrix is stored

Value

p value of the test

Examples

```
suffStat <- list(dm = cbind("a"=c(0,1,0,0,1,0), "b"=c(1,0,0,0,1,0), "c"=c(0,0,0,1,1,1)))
# test if a is independent of b
mcX2Test(1, 2, NULL, suffStat)
# test if a is independent of b conditioned on c
mcX2Test(1, 2, 3, suffStat)
```

`mcX2TestB50k` *Wrapper around the bnlearn mc-x2 test (B=50k)*

Description

Version of `mcX2Test()` with 50000 Monte Carlo replicates.

Usage

```
mcX2TestB50k(x, y, S, suffStat)
```

Arguments

`x`	the index of the first variable
`y`	the index of the second variable
`S`	the conditioning set
`suffStat`	the sufficient statistics to do the test, in this case a list of one element: dm where the values matrix is stored

Value

p value of the test

Examples

```
suffStat <- list(dm = cbind("a"=c(0,1,0,0,1,0), "b"=c(1,0,0,0,1,0), "c"=c(0,0,0,1,1,1)))
# test if a is independent of b
mcX2TestB50k(1, 2, NULL, suffStat)
# test if a is independent of b conditioned on c
mcX2TestB50k(1, 2, 3, suffStat)
```

`mesoBin` *A list of binary `DDDataSet` objects.*

Description

`mesoBin` is a list of objects of class `DDDataSet`. It has been generated with the following code:

`mesoBin <- toDDDataSet(readFurlongData(), prettyNames=TRUE)`

Usage

```
data(mesoBin)
```

Details

The dataset represents binary binding signal for 5 transcription factors (TFs) at 1-5 time points during embryonic mesoderm development in Drosophila Melanogaster (Zinzen et al, 2009). The original data has been binarized by taking any signal greater than the threshold authors used as positive binding event.

The list contains 7 objects of type `DDDataSet` for 7 cis-regulatory module (CRM) classes. These classes are: neg (negative class of CRMs), Meso (CRMs active in early mesoderm), Meso_SM (CRMs active in early mesoderm and somatic muscle), VM (visceral muscle), SM (somatic muscle), VM_SM (active in both somatic and visceral muscle) and CM (active in cardiac muscle).

References

Robert P. Zinzen et al., "Combinatorial binding predicts spatio-temporal cis-regulatory activity," Nature 462, no. 7269 (November 5, 2009): 65-70.

See Also

`mesoCont`.

Examples

```
data(mesoBin)
names(mesoBin)
class(mesoBin$VM)
```

`mesoCont` *A list of continuous* `DDDataSet` *objects.*

Description

`mesoCont` is a list of objects of class `DDDataSet`. It has been generated with the following code:

```
mesoCont <- toDDDataSet(readFurlongData(), prettyNames=TRUE, convertToBinary=FALSE)
```

Usage

```
data(mesoCont)
```

Details

The dataset represents original continuous binding signal for 5 transcription factors (TFs) at 1-5 time points during embryonic mesoderm development in Drosophila Melanogaster (Zinzen et al, 2009). The original data is retained (from Supplementary Table 8 of the paper).

The list contains 7 objects of type `DDDataSet` for 7 cis-regulatory module (CRM) classes. These classes are: neg (negative class of CRMs), Meso (CRMs active in early mesoderm), Meso_SM (CRMs active in early mesoderm and somatic muscle), VM (visceral muscle), SM (somatic muscle), VM_SM (active in both somatic and visceral muscle) and CM (active in cardiac muscle).

References

Robert P. Zinzen et al., "Combinatorial binding predicts spatio-temporal cis-regulatory activity," Nature 462, no. 7269 (November 5, 2009): 65-70.

See Also

`mesoBin`.

Examples

```
data(mesoCont)
names(mesoCont)
class(mesoCont$VM)
```

`myX2c` *The Monte-Carlo chi-square test...*

Description

The Monte-Carlo chi-square test

Usage

```
myX2c(x, y, C, B=5000)
```

Arguments

`x`	the first variable (vector of values)
`y`	the second variable (vector of values)
`C`	the variables to condition on - either a vector, or a list of vectors
`B`	the number of Monte Carlo runs (defaults to 5000 if given NULL)

Details

This is the reimplementation of Monte Carlo chi-square test to be sure it works correctly. The Monte Carlo loop is implemented using `Rcpp` and uses the R function `r2dtable()` to generate random contingency tables with fixed marginals.

Value

the P-value of the test

`names,CITestResult-method` *Names of slots that can be accessed with $ notation...*

Description

Names of slots that can be accessed with $ notation

Usage

```
## S4 method for signature CITestResult
names(x)
```

Arguments

`x`	the CITestResult object

`names,DDDataSet-method` *Names of variables (+class)*

Description

Names of variables (including "class")

Usage

```
## S4 method for signature DDDataSet
names(x)
```

Arguments

`x`	the DDDataSet object

Value

the names of the variables

`names,DDGraph-method` *Names of properties*

Description

Names of different properties that can be accessed with $ operator

Usage

```
## S4 method for signature DDGraph
names(x)
```

Arguments

`x`	the DDGataSet object

Value

the names of the variables

`names,FurlongDataSet-method` *Names of variables*

Description

Get the names of variables (column names of signal matrix)

Usage

```
## S4 method for signature FurlongDataSet
names(x)
```

Arguments

`x`	FurlongDataSet object

`ncpc` *Make a Direct Dependence Graph using the NCPC algorithm...*

Description

Make a Direct Dependence Graph using the NCPC algorithm

Usage

```
ncpc(obj, alpha=0.05, p.value.adjust.method="none", test.type=c("mc-x2-c",
    "cor"), max.set.size=NULL, mc.replicates=5000, report.file=NULL,
    verbose=FALSE, star=FALSE, min.table.size=10)
```

Arguments

`obj`	DDDataSet object
`alpha`	the alpha (P-value) cutoff for conditional independence tests
`p.value.adjust.method`	the multiple testing correction adjustment method
`test.type`	the type of conditional independence test (default: Monte Carlo x2 test "mc-x2-c" for binary data and partial correlation "cor" for continuous data) . See the documentation for `ciTest` for other available conditional independence tests
`max.set.size`	the maximal number of variables to condition on, if NULL estimated from number of positives in class labels (default: NULL)
`mc.replicates`	the number of Monte-carlo replicates, if applicable (default: 5000)
`report.file`	name of the file where a detailed report is to be printed, reporting is suppressed if NULL (default: NULL)
`verbose`	if to print out information about how the algorithm is progressing (default: TRUE)
`star`	if to use the NCPC* algorithm (default: FALSE)
`min.table.size`	the minimal number of samples in a contingency table per conditioning set (makes sense only for discrete data)

Details

Make a Direct Dependence Graph using a P-value and conditional independence tests. There are two version of the algorithm: NCPC and NCPC*. NCPC finds the causal neighbourhood while the NCPC* infers the full Markov Blanket.

The full algorithm is given in (Stojnic et al, 2012).

Value

DDGraph object

References

R. Stojnic et al (2012): "A Graphical Modelling Approach to the Dissection of Highly Correlated Transcription Factor Binding Site Profiles", in press, PloS Computational Biology.

Examples

```
### load binary data for Mesoderm
data(mesoBin)
# run the NCPC algorithm with alpha=0.05 (on discrete data)
ncpc(mesoBin$Meso, alpha=0.05, test.type="mc-x2-c")
# run the NCPC* algorithm with alpha=0.05 (on discrete data)
res <- ncpc(mesoBin$Meso, alpha=0.05, test.type="mc-x2-c", star=TRUE)

# analysis of results:
class(res)
# although of class DDGraph, behaves much like a list
names(res)
# parameters used in obtaining results
res$params
# labels for each of the variables
res$final.calls
# direct variables
res$direct

### load continous data
data(mesoCont)
# run the NCPC algorith with alpha=0.05 (on continuous data)
ncpc(mesoCont$Meso, alpha=0.05, test.type="cor", max.set.size=1)
# run the NCPC* algorith with alpha=0.05 (on continuous data)
ncpc(mesoCont$Meso, alpha=0.05, test.type="cor", max.set.size=1, star=TRUE)
```

`ncpcResampling` *NCPC Robustness from resampling*

Description

Estimate the NCPC robustness using either jackknife or bootstrap resampling.

Usage

```
ncpcResampling(obj, method="bootstrap", method.param, verbose=TRUE, ...)
```

Arguments

`obj`	the DDDataSet object
`method`	the method to use to estimate how robust is the feature selection (valid values: "jackknife", or "bootstrap").
`method.param`	the parameter to method, either number of data points to remove for "jackknife" (default: 1) or number of boostrap runs for "bootstrap" (default: 100).

`verbose`	if to print out the progress
`...`	other parameters to pass to ncpc()

Details

Estimate the robustness of NCPC predictions (i.e. variable types: direct, joint, indirect, no dependence) using resampling. Two type of resampling are available: bootstrap (where the whole dataset is resampled with replacement), and jackknifing (where 1 or more observation are removed at each resampling step).

NCPC is run for the resampled datasets and statistics is produced about how many times is each variable assigned one of the four types (direct, joint, indirect, no dependence). The final call for each variable is then made according to the following algorithm (#direct is number of times variable is called direct):

1. if #no dependence > #direct+joint+indirect => "no dependence"
2. else if #indirect > #direct+joint => "indirect"
3. else if #joint > #direct => "joint"
4. else "direct"

Value

NCPCRobustness object with the raw results from resampling and summarized results

Examples

```
## Not run:
# load the example data
data(mesoBin)

# run bootstrap resampling for NCPC with alpha=0.05
ncpcResampling(mesoBin$VM_SM, "bootstrap", 100, alpha=0.05)
# run bootstrap resampling for NCPC* with alpha=0.05
ncpcResampling(mesoBin$VM_SM, "bootstrap", 100, alpha=0.05, star=TRUE)

# run jackknifing for NCPC
ncpcResampling(mesoBin$VM_SM, "jackknife", 1, alpha=0.05)

## End(Not run)
```

`NCPCRobustness-class` *NCPC resampling robustness...*

Description

NCPC resampling robustness

Details

Data class that stores the robustness information associated with an NCPC result from resampling runs (bootstrap of jackknifing). It contains the results from the resampling runs as well as summary statistics. The `final.calls` slot contains the final assigned types based on resampling.

Slots

`dataset`: (`DDDataSet`) the associated DDDataSet object

`raw`: (`list`) the raw data from the robustness analysis

`params`: (`list`) the parameters used to generate the data (including the resampling method)

`tables`: (`list`) the frequencies of assigning each variable to a class

`runs`: (`numeric`) the number of resampling runs

`enriched.pss`: (`data.frame`) the table with reports for consistently enriched variables split

`enriched.ps`: (`data.frame`) the table with reports for consistently enriched variable split into two classes: directAndJoint, indirect

`not.enriched`: (`data.frame`) the table with reports for the consistently not enriched variables

`final.calls`: (`data.frame`) the table with finals calls for types of variables

---

`operators-CITestResult`

Access slots using the dollar notation...

Description

Access slots using the dollar notation

Usage

```
## S4 method for signature CITestResult
x$name
## S4 method for signature CITestResult,ANY,ANY
x[[i, j, ...]]
```

Arguments

`x`	the CITestResult object
`name`	the slot name
`i`	the slot name
`j`	unused
`...`	unused

`operators-DDDataSet` *access a specific variable in the dataset by name...*

Description

access a specific variable in the dataset by name

Usage

```
## S4 method for signature DDDataSet
x$name
## S4 method for signature DDDataSet
x[[i, j]]
## S4 method for signature DDDataSet,ANY,ANY
x[i, j, ..., drop=TRUE]
```

Arguments

`x`	the DDDataSet object
`name`	the variable name
`i`	variable name
`j`	unused
`drop`	unused
`...`	unused

`operators-DDGraph` *access a property by name...*

Description

access a property by name

Usage

```
## S4 method for signature DDGraph
x$name
```

Arguments

`x`	the DDGraph object
`name`	the variable name

`pcalgMB` *Find the markov blanket for the PC algorithm output...*

Description

Find the markov blanket for the PC algorithm output

Usage

```
pcalgMB(pc, node)
```

Arguments

`pc`	output of PC algorithm from package pcAlgo, object of class "pcAlgo"
`node`	the index of the node for which we are seeking the Markov Blanket

Value

the inidicies of nodes that constitute the Markov Blanket

`pcalgNBR` *Find the neighbourhood for the PC algorithm output...*

Description

Find the neighbourhood for the PC algorithm output

Usage

```
pcalgNBR(pc, node)
```

Arguments

`pc`	output of PC algorithm from package pcAlgo, object of class "pcAlgo"
`node`	the index of the node for which we are seeking the neighbourhood

Value

the inidicies of nodes that constitute the (undirected) neighbourhood

---

`plot,DDGraph,missing-method`

Plot DDGraphs using RGraphviz...

Description

Plot DDGraphs using RGraphviz

Usage

```
## S4 method for signature DDGraph,missing
plot(x, y, ..., col=NULL, legend=FALSE, only.legend=FALSE, plot.class=TRUE,
    class.label=datasetName(x@dataset), ci.symbol="dot",
    plot.pvals=TRUE, pvals.format=function(x) sprintf("%.2f", x),
    pvals.fontsize=12, main=NULL)
```

Arguments

`x`	DDGraph object
`y`	unusued
`col`	specifies the colors to be used to color nodes. Can be any of the following: • named vector of colors • logical value (TRUE = nodes colored in default 0.1 to 1e-3 range, FALSE = no node coloring) - only available for binary datasets. • list of parameters to pass to mapEnrichmentToColorsDual(), valid pameters are: "palette.pos", "palette.neg", "class.col", "scale", "max.color.index"
`legend`	if to plot the color legend
`only.legend`	if to plot only the legend
`plot.class`	if to plot class labels node
`class.label`	if plot.class=TRUE the label of the class node
`plot.pvals`	if to plot p values on top of edges
`ci.symbol`	the RGraphviz arrowtail/head symbol name for conditional independence tests
`pvals.format`	a function to format the p values to be displayed on directed edges
`pvals.fontsize`	the size of the font for p values
`main`	main title
`...`	other parameters passed to layoutGraph()

Examples

```
## Not run:
# load data
data(mesoBin)
# make DDGraph
g <- ncpc(mesoBin$Meso)

# default plot
plot(g)

# use colours
plot(g, col=TRUE)

## End(Not run)
```

`plotBNLearn` *A custom plotting function for the BNlearn graphs...*

Description

A custom plotting function for the BNlearn graphs

Usage

```
plotBNLearn(d, bnlearn.function.name="hc", alpha=0.05, test="mc-mi",
    make.plot=FALSE, blacklist, B, restart=0, scale=1.5,
    class.label="target", use.colors=TRUE, score="bic")
```

Arguments

`d`	an object of type DDDataSet
`bnlearn.function.name`	the bnlearn reconstruction algorithm to use (default: hc)
`alpha`	the alpha value of conditional independence tests (if applicable)
`test`	the type of conditional independence test (if applicable)
`make.plot`	if to make a plot or just return the network (default: FALSE)
`blacklist`	a data frame with two columns (optionally labeled "from" and "to"), containing a set of arcs not to be included in the graph.
`B`	the number of bootstrap runs of permutations (for iamb and such algorithms)
`restart`	the number of random restarts for score-based algorithms
`scale`	the color scaling
`class.label`	the label to use for the class variable
`use.colors`	if to color code the results
`score`	the scoring penalization metric to use (when applicable)

Value

an object of class "bn" representing the inferred network

Examples

```
data(mesoBin)
# use hill-climbing to make the causal network and plot with enrichment colours
plotBNLearn(mesoBin$Meso, make.plot=TRUE)
```

`plotPCalg` *Plot the network inferred by the PC algorithm*

Description

Infer a network using PC algorithm and plot it.

Usage

```
plotPCalg(d, name, alpha=0.05, verbose=FALSE, directed=TRUE, make.plot=FALSE,
    scale=1.5, indepTest=mcX2Test, class.label="target",
    use.colors=TRUE)
```

Arguments

`d`	DDDataSet object
`name`	the name to show (defaults to dataset name)
`alpha`	the alpha value cut-off for the conditional independence tests
`verbose`	if to show progress
`directed`	if TRUE applies PC algorithm, if FALSE applies PC-skeleton
`make.plot`	if to output the plot into the active device
`scale`	the scaling parameter for color-coding
`indepTest`	the independence test wrapper function (as needed by package `pcalg`)
`class.label`	the label to show for class variable
`use.colors`	if to color code the results

Examples

```
data(mesoBin)
# use PC algorithm to construct a causal network and colour it according to enrichment/depletion
plotPCalg(mesoBin$Meso, alpha=0.05, directed=TRUE, make.plot=TRUE)
```

`plotSVMPerformance` *Plot SVM performance into a pdf file*

Description

A companion function for `svmFeatureSelectionLOOCV()` to plot the results.

Usage

```
plotSVMPerformance(obj, results, plot.file)
```

Arguments

`obj`	the DDDataSet object for which the SVM performance is measured
`results`	the results from svmFeatureSelectionLOOCV
`plot.file`	the name of the output pdf file

`predSVM` *Calculate the decision value of an SVM model*

Description

Calculate the decision value of an SVM model. Note this is different from the actual prediction which is either 0 or 1, while decision values go from -1 to 1. (taken from [Zizen 2009] supplementary code)

Usage

```
predSVM(f, feature)
```

Arguments

`f`	The trained SVM model object.
`feature`	The input value to which output is needed.

Value

Decision value in the range -1 to 1.

`prob.distr.norm` *Normal distribution function for* `random.bn.fit`

Description

Generate 2^n numbers from distribution with most of the pdf mass in extreme probabilities (mirrored normal). We use standard deviation of 1/3 and modulo-1 of normal distribution.

Usage

```
prob.distr.norm(n, sd=1/3)
```

Arguments

`n`	number of variables
`sd`	the standard deviation of distribution

Value

vector of 2^n random numbers

Examples

```
# return 8 random numbers since n=3
prob.distr.norm(3)
```

`prob.distr.unif` *Uniform distribution for* `random.bn.fit`

Description

Uniform distribution function for `random.bn.fit`

Usage

```
prob.distr.unif(n)
```

Arguments

`n`	number of variables

Details

Generate 2^n uniformly distributed numbers in range 0 to 1

Value

vector of 2^n uniform random numbers

Examples

```
# return 8 random uniform numbers
prob.distr.unif(3)
```

`pValueAfterMultipleTesting`

Multiple testing correction procedure for ncpc()

Description

This function is only for DDGraph with multiple testing correction enabled. The overall procedure is similar to that described in (Li&Wang 2009). This is a helper function for `DDDataSet:ncpc()`. The single P-value of D-separation is substituted in the list of P-values, P-values adjusted and the resulting P-value after correction in the context of other P-values reported.

Usage

```
pValueAfterMultipleTesting(dsep, x, adjC.pvals.at.n, p.value.adjust.method)
```

Arguments

`dsep`	the conditional independence test result (of type `CITestResult`)
`x`	the index of the variables
`adjC.pvals.at.n`	the p values associated with the variables at size n of conditioning set (list [[n]] -> [pvals])
`p.value.adjust.method`	the p value adjustment method (same as in p.adjust())

Value

the p value after multiple test correction (if any)

References

J. Li and Z. J Wang, "Controlling the false discovery rate of the association/causality structure learned with the PC algorithm" The Journal of Machine Learning Research 10 (2009): 475-514.

`random.bn.fit` *Generate a random* `bn.fit` *network*

Description

Generate a random Bayesian network using package `bnlearn`. The nodes specify the partial ordering of the graph, and the conditional probabilities are sampled from given distribution. The network is generated to have on average given number of neighbours (i.e. both in-going and out-going edges)

Usage

```
random.bn.fit(nodes, num.neigh=2, prob.distr=prob.distr.norm, bn.graph)
```

Arguments

`nodes`	a vector of desired node names (basis for partial ordering)
`num.neigh`	expected number of neighbours per node in the random graph
`prob.distr`	the probability distribution function to use
`bn.graph`	the `bn` object with an already laid out graph, if not supplied will be generated

Value

a list of two elements: `bn` - a `bn` object which contains the structure and `bn.fit` - a `bn.fit` object with filled in conditional probabilities

Examples

```
# a random network with 3 nodes "A", "B", "C" with average of 1 neighbour
random.bn.fit(c("A", "B", "C"), num.neigh=1)
```

`rawData,DDDataSet-method` *Raw data.frame with data*

Description

Return the raw data frame with the variables, and the last column being "class"

Usage

```
## S4 method for signature DDDataSet
rawData(obj, ...)
```

Arguments

`obj`	the DDDataSet object
`...`	unused

Value

the raw dataframe that contains all the data

`readFurlongData` *Read the Furlong Dataset*

Description

Read the Furlong data into a FurlongDataSet object.

Usage

```
readFurlongData(infile)
```

Arguments

`infile`	the filename to load from, default to supplementary_table_8_training_set.txt in extdata/ of package

Details

Read the Furlong Dataset form the Supplementary Table 8 file provided with the package. An alternative filename can be specified as well.

Value

an object of type FurlongDataSet witht the loaded data

Examples

```
# read the furlong dataset that is provided with the package
readFurlongData()
```

`recalculateSVMparams` *Calculate SVM hyperparameters based on grid search*

Description

Find the cost/gamma parameters based on a grid search by best AUC and by limiting the number of support vectors. Currently only supports discreet binary data.

Usage

```
recalculateSVMparams(cost.range, gamma.range, d,
    class.weight=1/table(convertToFactor(d$class)), kernel="radial",
    max.prop.SV=0.9)
```

Arguments

`cost.range`	the range of cost parameter values to evaluate
`gamma.range`	the range of gamma parameter values to evaluate
`d`	the data.frame with variables as columns, the class labels must be labelled with "class"
`class.weight`	the class weights to use (if there is an large bias for positive/negative class)
`kernel`	kernel type to use (takes valid package e1071 names like "radial")
`max.prop.SV`	the maximal proportion of support vectors to number of data points (rows in d)

Value

a list with the two parameters that give best AUC in LOOCV

Examples

```
## Not run:
data(mesoBin)
# get SVM AUC etc over cost rage of 1, 100, and gamma range of 0.1, 1
recalculateSVMparams(c(1, 100), c(0.1, 1), convertToFactor(rawData(mesoBin$Meso)))

## End(Not run)
```

`show,CITestResult-method` *show method for CITestResult...*

Description

show method for CITestResult

Usage

```
## S4 method for signature CITestResult
show(object)
```

Arguments

`object`	the CITestResult object

`show,DDDataSet-method` *show method for DDDataSet...*

Description

show method for DDDataSet

Usage

```
## S4 method for signature DDDataSet
show(object)
```

Arguments

`object`	the DDDataSet object

`show,DDGraph-method` *show method for DDGraph...*

Description

show method for DDGraph

Usage

```
## S4 method for signature DDGraph
show(object)
```

Arguments

`object`	the DDGraph object

`show,DDGraphEdge-method`
show method for DDGraphEdge...

Description

show method for DDGraphEdge

Usage

```
## S4 method for signature DDGraphEdge
show(object)
```

Arguments

`object`	the DDGraphEdge object

`signalMatrix,FurlongDataSet-method`
Raw values

Description

Retrieve the matrix with raw signal values

Usage

```
## S4 method for signature FurlongDataSet
signalMatrix(object)
```

Arguments

`object`	FurlongDataSet object

`svmFeatureSelectionLOOCV`
Nested variable selection using LOOCV

Description

Nested variable selection using LOOCV

Usage

```
svmFeatureSelectionLOOCV(obj, selectionMode="direct", alpha=0.1, p.value.adjust.method="none",
    test.type="mc-x2", mc.replicates=5000, cost.range=logseq(0.01,
    1e+05, 8), gamma.range=logseq(1e-05, 100, 8), max.prop.SV=0.9,
    kernel="radial", skip.DDGraph=FALSE)
```

Arguments

`obj`	the DDDataSet object
`selectionMode`	which variables to take, possible values: "direct" (alias "p"), "direct and joint" (alias "ps"), "joint if no direct" (alias "snp")
`alpha`	the alpha cutoff to use
`p.value.adjust.method`	the p value adjustment for multiple testing to be applied
`test.type`	the type of conditional independence test to be used
`mc.replicates`	the number of Monte-Carlo replicates when determining p values
`cost.range`	the range of cost parameter values to evaluate
`gamma.range`	the range of gamma parameter values to evaluate

`max.prop.SV`	the maximal proportion of support vectors to number of data points (rows in d)
`kernel`	kernel type to use (takes valid package e1071 names like "radial")
`skip.DDGraph`	if to skip DDGraph-based variable selection

Details

A function to select variables in nested way using the following algorithm:

1. repeat for each row in dataset:
   (a) make new DDDataSet by removing one row and apply DDGraphs to select features
   (b) select best parameters using recalculateSVMparams (i.e. in an inner LOOCV loop)
   (c) make the classifier with best parameters and calculate output on the unseen row (removed in step 1)
2. return the collected predictions from step 1.3

Value

the predictions for class labels from LOOCV

`toDDDataSet,FurlongDataSet-method`
DDDataSet object from FurlongDataSet

Description

Make the DDDataSet objects by selecting different tissues

Usage

```
## S4 method for signature FurlongDataSet
toDDDataSet(obj, tissues=c(), convertToBinary=TRUE, prettyNames=FALSE, ...)
```

Arguments

`obj`	the FurlongDataSet object
`tissues`	tissue names for which DDDataSet objects should be generated (default to all available tissues)
`convertToBinary`	if to convert the signal into binary values
`prettyNames`	if to make the names pretty, e.g. twi_2.4 -> Twi 2-4h
`...`	unused

Value

either single DDDataSet object, or a list of them (depending on number of selected tissues)

Examples

```
# load binarized data with prettified names
all.data <- toDDDataSet(readFurlongData(), prettyNames=TRUE)
# load continuous data with original names
all.data <- toDDDataSet(readFurlongData(), convertToBinary=FALSE)
```

`toyExample` *A binary fictional toy example* `DDDataSet` *object.*

Description

`toyExample` is an example dataset representing a set of 200 fictional cis-regulatory modules (CRMs). The dataset contains binding patterns for two transcription factors A and B. It is used only in the package vignette.

Usage

```
data(toyExample)
```

Details

In this fictional dataset we represent binding patterns of two transcription factors A and B on a set of CRMs. The target variable (T) is another binary vector that represents if a CRM is tissue specific or not (as obtained by e.g. transgenic reporter assays).

For more information and detailed examples see the package vignette.

Examples

```
data(toyExample)
calcDependence(toyExample)
```

`variableNames,DDDataSet-method` *Names of variables (-class)*

Description

Names of variables (without "class")

Usage

```
## S4 method for signature DDDataSet
variableNames(obj, ...)
```

Arguments

`obj`	the DDDataSet object
`...`	unused

Value

only the names of the variables (i.e. without "class")

Index