Package: miic (via r-universe)

October 15, 2026

Title Learning Causal or Non-Causal Graphical Models Using Information Theory

Version 2.0.4

Description Multivariate Information-based Inductive Causation, better known by its acronym MIIC, is a causal discovery method, based on information theory principles, which learns a large class of causal or non-causal graphical models from purely observational data, while including the effects of unobserved latent variables. Starting from a complete graph, the method iteratively removes dispensable edges, by uncovering significant information contributions from indirect paths, and assesses edge-specific confidences from randomization of available data. The remaining edges are then oriented based on the signature of causality in observational data. The recent more interpretable MIIC extension (iMIIC) further distinguishes genuine causes from putative and latent causal effects, while scaling to very large datasets (hundreds of thousands of samples). Since the version 2.0, MIIC also includes a temporal mode (tMIIC) to learn temporal causal graphs from stationary time series data. MIIC has been applied to a wide range of biological and biomedical data, such as single cell gene expression data, genomic alterations in tumors, live-cell time-lapse imaging data (CausalXtract), as well as medical records of patients. MIIC brings unique insights based on causal interpretation and could be used in a broad range of other data science domains (technology, climatology, economy, ...). For more information, you can refer to: Simon et al., eLife 2024, <doi:10.1101/2024.02.06.579177>, Ribeiro-Dantas et al., iScience 2024, <doi:10.1016/j.isci.2024.109736>, Cabeli et al., NeurIPS 2021, <https://why21.causalai.net/papers/WHY21_24.pdf>, Cabeli et al., Comput. Biol. 2020, <doi:10.1371/journal.pcbi.1007866>, Li et al., NeurIPS 2019, <https://papers.nips.cc/paper/9573-constraint-based-causal-structure-learning-with-consistent-separating-sets>, Verny et al., PLoS Comput. Biol. 2017,

<doi:10.1371/journal.pcbi.1005662>, Affeldt et al., UAI 2015, <https://auai.org/uai2015/proceedings/papers/293.pdf>. Changes from the previous 1.5.3 release on CRAN are available at <https://github.com/miicTeam/miic_R_package/blob/master/NEWS.md>.

License GPL (>= 2)

URL https://github.com/miicTeam/miic_R_package

BugReports https://github.com/miicTeam/miic_R_package/issues

Imports ppcor, Rcpp, scales, stats,

Suggests igraph, grDevices, ggplot2 (>= 3.3.0), gridExtra

LinkingTo Rcpp

LazyData true

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation yes

Author Franck Simon [aut, cre], Ali Chemkhi [aut], Tiziana Tocci [aut], Nikita Lagrange [aut], Orianne Debeaupuis [aut], Louise Dupuis [aut], Vincent Cabeli [aut], Honghao Li [aut], Marcel Ribeiro Dantas [aut], Nadir Sella [aut], Louis Verny [aut], Severine Affeldt [aut], Hervé Isambert [aut]

Maintainer Franck Simon <franck.simon@curie.fr>

Repository https://cran.r-universe.dev

Date/Publication 2026-09-14 20:50:02 UTC

RemoteUrl https://github.com/cran/miic

RemoteRef HEAD

RemoteSha 9ac760ffb2f8252ff5ef7925db6e96a11e091648

Contents

computeMutualInfo . . . 3
computeThreePointInfo . . . 5
cosmicCancer . . . 7
cosmicCancer_stateOrder . . . 8
covidCases . . . 9
discretizeMDL . . . 9
discretizeMutual . . . 10
estimateTemporalDynamic . . . 12
export . . . 14
hematoData . . . 16
miic . . . 17
plot.miic . . . 29
plot.tmiic . . . 30
writeCytoscapeNetwork . . . 32
writeCytoscapeStyle . . . 33

Index **34**

`computeMutualInfo` *Compute (conditional) mutual information*

Description

For discrete or categorical variables, the (conditional) mutual information is computed using the empirical frequencies minus a complexity cost (computed as BIC or with the Normalized Maximum Likelihood). When continuous variables are present, each continuous variable is discretized for each mutual information estimate so as to maximize the mutual information minus the complexity cost (see Cabeli 2020).

Usage

```
computeMutualInfo(
  x,
  y,
  df_conditioning = NULL,
  maxbins = NULL,
  cplx = c("nml", "bic"),
  n_eff = -1,
  sample_weights = NULL,
  is_continuous = NULL,
  plot = FALSE
)
```

Arguments

`x`	[a vector] The X vector that contains the observational data of the first variable.
`y`	[a vector] The Y vector that contains the observational data of the second variable.
`df_conditioning`	[a data frame] The data frame of the observations of the conditioning variables.
`maxbins`	[an integer] When the data contain continuous variables, the maximum number of bins allowed during the discretization. A smaller number makes the computation faster, a larger number allows finer discretization.
`cplx`	[a string] The complexity model: • ["bic"] Bayesian Information Criterion • ["nml"] Normalized Maximum Likelihood, more accurate complexity cost compared to BIC, especially on small sample size.
`n_eff`	[an integer] The effective number of samples. When there is significant autocorrelation between successive samples, you may want to specify an effective number of samples that is lower than the total number of samples.
`sample_weights`	[a vector of floats] Individual weights for each sample, used for the same reason as the effective number of samples but with individual weights.

`is_continuous` [a vector of booleans] Specify if each variable is to be treated as continuous (TRUE) or discrete (FALSE), must be of length 'ncol(df_conditioning) + 2', in the order $X, Y, U1, U2, ...$ If not specified, factors and character vectors are considered as discrete, and numerical vectors as continuous.

`plot` [a boolean] Specify whether the resulting XY optimum discretization is to be plotted (requires 'ggplot2' and 'gridExtra').

Details

For a pair of continuous variables X and Y, the mutual information $I(X;Y)$ will be computed iteratively. In each iteration, the algorithm optimizes the partitioning of X and then of Y, in order to maximize

$$Ik(X_d;Y_d) = I(X_d;Y_d) - cplx(X_d;Y_d)$$

where $cplx(X_d;Y_d)$ is the complexity cost of the corresponding partitioning (see Cabeli 2020). Upon convergence, the information terms $I(X_d;Y_d)$ and $Ik(X_d;Y_d)$, as well as the partitioning of X_d and Y_d in terms of cutpoints, are returned.

For conditional mutual information with a conditioning set U, the computation is done based on

$$Ik(X;Y|U) = 0.5 * (Ik(X_d;Y_d,U_d) - Ik(X_d;U_d) + Ik(Y_d;X_d,U_d) - Ik(Y_d;U_d)),$$

where each of the four summands is estimated separately.

Value

A list that contains :

- cutpoints1: Only when X is continuous, a vector containing the cutpoints for the partitioning of X.
- cutpoints2: Only when Y is continuous, a vector containing the cutpoints for the partitioning of Y.
- n_iterations: Only when at least one of the input variables is continuous, the number of iterations it takes to reach the convergence of the estimated information.
- iteration1, iteration2, ... Only when at least one of the input variables is continuous, the list of vectors of cutpoints of each iteration.
- info: The estimation of (conditional) mutual information without the complexity cost.
- infok: The estimation of (conditional) mutual information with the complexity cost ($Ik = I - cplx$).
- plot: Only when 'plot == TRUE', the plot object.

References

- Cabeli *et al.*, PLoS Comput. Biol. 2020, Learning clinical networks from medical records based on information estimates in mixed-type data
- Affeldt *et al.*, UAI 2015, Robust Reconstruction of Causal Graphical Models based on Conditional 2-point and 3-point Information

Examples

```
library(miic)
N <- 1000
# Dependence, conditional independence : X <- Z -> Y
Z <- runif(N)
X <- Z * 2 + rnorm(N, sd = 0.2)
Y <- Z * 2 + rnorm(N, sd = 0.2)
res <- computeMutualInfo(X, Y, plot = FALSE)
message("I(X;Y) = ", res$info)
res <- computeMutualInfo(X, Y, df_conditioning = matrix(Z, ncol = 1), plot = FALSE)
message("I(X;Y|Z) = ", res$info)


# Conditional independence with categorical conditioning variable : X <- Z -> Y
Z <- sample(1:3, N, replace = TRUE)
X <- -as.numeric(Z == 1) + as.numeric(Z == 2) + 0.2 * rnorm(N)
Y <- as.numeric(Z == 1) + as.numeric(Z == 2) + 0.2 * rnorm(N)
res <- miic::computeMutualInfo(X, Y, cplx = "nml")
message("I(X;Y) = ", res$info)
res <- miic::computeMutualInfo(X, Y, matrix(Z, ncol = 1), is_continuous = c(TRUE, TRUE, FALSE))
message("I(X;Y|Z) = ", res$info)


# Independence, conditional dependence : X -> Z <- Y
X <- runif(N)
Y <- runif(N)
Z <- X + Y + rnorm(N, sd = 0.1)
res <- computeMutualInfo(X, Y, plot = TRUE)
message("I(X;Y) = ", res$info)
res <- computeMutualInfo(X, Y, df_conditioning = matrix(Z, ncol = 1), plot = TRUE)
message("I(X;Y|Z) = ", res$info)
```

---

computeThreePointInfo *Compute (conditional) three-point information*

Description

Three point information is defined and computed as the difference of mutual information and conditional mutual information, e.g.

$$I(X;Y;Z|U) = I(X;Y|U) - Ik(X;Y|U,Z)$$

For discrete or categorical variables, the three-point information is computed with the empirical frequencies minus a complexity cost (computed as BIC or with the Normalized Maximum Likelihood).

Usage

```
computeThreePointInfo(
  x,
  y,
  z,
  df_conditioning = NULL,
  maxbins = NULL,
  cplx = c("nml", "bic"),
  n_eff = -1,
  sample_weights = NULL,
  is_continuous = NULL
)
```

Arguments

x	[a vector] The X vector that contains the observational data of the first variable.
y	[a vector] The Y vector that contains the observational data of the second variable.
z	[a vector] The Z vector that contains the observational data of the third variable.
df_conditioning	[a data frame] The data frame of the observations of the set of conditioning variables U.
maxbins	[an integer] When the data contain continuous variables, the maximum number of bins allowed during the discretization. A smaller number makes the computation faster, a larger number allows finer discretization.
cplx	[a string] The complexity model: • ["bic"] Bayesian Information Criterion • ["nml"] Normalized Maximum Likelihood, more accurate complexity cost compared to BIC, especially on small sample size.
n_eff	[an integer] The effective number of samples. When there is significant autocorrelation between successive samples, you may want to specify an effective number of samples that is lower than the total number of samples.
sample_weights	[a vector of floats] Individual weights for each sample, used for the same reason as the effective number of samples but with individual weights.
is_continuous	[a vector of booleans] Specify if each variable is to be treated as continuous (TRUE) or discrete (FALSE), must be of length 'ncol(df_conditioning) + 3', in the order $X, Y, Z, U1, U2, ...$ If not specified, factors and character vectors are considered as discrete, and numerical vectors as continuous.

Details

For variables X, Y, Z and a set of conditioning variables U, the conditional three point information is defined as

$$Ik(X;Y;Z|U) = Ik(X;Y|U) - Ik(X;Y|U,Z)$$

where Ik is the shifted or regularized conditional mutual information. See computeMutualInfo for the definition of Ik.

Value

A list that contains :

- i3: The estimation of (conditional) three-point information without the complexity cost.
- i3k: The estimation of (conditional) three-point information with the complexity cost (*i3k = i3 - cplx*).
- i2: For reference, the estimation of (conditional) mutual information $I(X;Y|U)$ used in the estimation of *i3*.
- i2k: For reference, the estimation of regularized (conditional) mutual information $Ik(X;Y|U)$ used in the estimation of *i3k*.

References

- Cabeli *et al.*, PLoS Comput. Biol. 2020, Learning clinical networks from medical records based on information estimates in mixed-type data
- Affeldt *et al.*, UAI 2015, Robust Reconstruction of Causal Graphical Models based on Conditional 2-point and 3-point Information

Examples

```
library(miic)
N <- 1000
# Dependence, conditional independence : X <- Z -> Y
Z <- runif(N)
X <- Z * 2 + rnorm(N, sd = 0.2)
Y <- Z * 2 + rnorm(N, sd = 0.2)
res <- computeThreePointInfo(X, Y, Z)
message("I(X;Y;Z) = ", res$i3)
message("Ik(X;Y;Z) = ", res$i3k)


# Independence, conditional dependence : X -> Z <- Y
X <- runif(N)
Y <- runif(N)
Z <- X + Y + rnorm(N, sd = 0.1)
res <- computeThreePointInfo(X, Y, Z)
message("I(X;Y;Z) = ", res$i3)
message("Ik(X;Y;Z) = ", res$i3k)
```

---

cosmicCancer *Genomic and ploidy alterations in breast tumors*

Description

The dataset contains 807 samples without predisposing Brca1/2 germline mutations and includes 204 somatic mutations (from whole exome sequencing) and expression level information for 91 genes.

Usage

```
data(cosmicCancer)
```

Format

A data.frame object.

References

Forbes SA, Beare D, Gunasekaran P, Leung K, Bindal N, et al. (2015) Nucleic Acids Res 43:D805–D811 (doi:10.1093/nar/gku1075).

`cosmicCancer_stateOrder`

Genomic and ploidy alterations in breast tumors

Description

The dataset contains 807 samples without predisposing Brca1/2 germline mutations and includes 204 somatic mutations (from whole exome sequencing) and expression level information for 91 genes, category order file.

Usage

```
data(cosmicCancer_stateOrder)
```

Format

A data.frame object.

References

Forbes SA, Beare D, Gunasekaran P, Leung K, Bindal N, et al. (2015) Nucleic Acids Res 43:D805–D811 (doi:10.1093/nar/gku1075).

covidCases *Covid cases*

Description

Demo dataset of chronological series to be used in temporal mode of miic. Evolution of Covid cases on a subset of EU countries from 12/31/2019 to 06/18/2020. Source of the data : European Centre for Disease Prevention and Control.

Usage

```
data(covidCases)
```

Format

A data.frame object.

References

ECDC (ECDC link)

discretizeMDL *Discretize a real valued distribution*

Description

This function performs minimum description length (MDL)-optimal histogram density estimation as described in Kontkanen and Myllymäki (2007) and returns the cutpoints found to give the best model according to the MDL principle.

Usage

```
discretizeMDL(x = NULL, max_bins = 20)
```

Arguments

x	[a vector] A vector that contains the distribution to be discretized.
max_bins	[an int] The maximum number of bins allowed by the algorithm.

Value

A list containing the cutpoints of the best discretization.

References

- Kontkanen P, Myllymäki P. MDL histogram density estimation. Artificial Intelligence and Statistics 2007 Mar 11 (pp. 219-226).

Examples

```
library(miic)
# Bimodal normal distribution
N <- 300
modes <- sample(1:2, size = N, replace = TRUE)
x <- as.numeric(modes == 1) * rnorm(N, mean = 0, sd = 1) +
     as.numeric(modes == 2) * rnorm(N, mean = 5, sd = 2)
MDL_disc <- discretizeMDL(x)
hist(x, breaks = MDL_disc$cutpoints)

N <- 2000
modes <- sample(1:2, size = N, replace = TRUE)
x <- as.numeric(modes == 1) * rnorm(N, mean = 0, sd = 1) +
     as.numeric(modes == 2) * rnorm(N, mean = 5, sd = 2)
MDL_disc <- discretizeMDL(x)
hist(x, breaks = MDL_disc$cutpoints)
```

---

discretizeMutual *Iterative dynamic programming for (conditional) mutual information through optimized discretization.*

Description

This function chooses cutpoints in the input distributions by maximizing the mutual information minus a complexity cost (computed as BIC or with the Normalized Maximum Likelihood). The (conditional) mutual information computed on the optimized discretized distributions effectively estimates the mutual information of the original continuous variables.

Usage

```
discretizeMutual(
  x,
  y,
  matrix_u = NULL,
  maxbins = NULL,
  cplx = "nml",
  n_eff = NULL,
  sample_weights = NULL,
  is_continuous = NULL,
  plot = TRUE
)
```

Arguments

x	[a vector] The X vector that contains the observational data of the first variable.
y	[a vector] The Y vector that contains the observational data of the second variable.

`matrix_u` [a numeric matrix] The matrix with the observations of as many columns as conditioning variables.

`maxbins` [an int] The maximum number of bins desired in the discretization. A lower number makes the computation faster, a higher number allows finer discretization (by default : 5 * cubic root of N).

`cplx` [a string] The complexity used in the dynamic programming:

- ["bic"] Bayesian Information Criterion
- ["nml"] Normalized Maximum Likelihood, more accurate complexity cost compared to BIC, especially on small sample size.

`n_eff` [an integer] The effective number of samples. When there is significant autocorrelation between successive samples, you may want to specify an effective number of samples that is lower than the total number of samples.

`sample_weights` [a vector of floats] Individual weights for each sample, used for the same reason as the effective number of samples but with individual weights.

`is_continuous` [a vector of booleans] Specify if each variable is to be treated as continuous (TRUE) or discrete (FALSE) in a logical vector of length ncol(matrix_u) + 2, in the order [X, Y, U1, U2...]. By default, factors and character vectors are treated as discrete, and numerical vectors as continuous.

`plot` [a boolean] Specify whether the resulting XY optimum discretization is to be plotted (requires ‘ggplot2‘ and ‘gridExtra‘).

Details

For a pair of continuous variables X and Y, the algorithm will iteratively choose cutpoints on X then on Y, maximizing $I(X_d; Y_d) - cplx(X_d; Y_d)$ where $cplx(X_d; Y_d)$ is the complexity cost of the considered discretizations of X and Y (see Cabeli 2020). Upon convergence, the discretization scheme of X_d and Y_d is returned as well as $I(X_d; Y_d)$ and $I(X_d; Y_d) - cplx(X_d; Y_d)$.

With a set of conditioning variables U, the discretization scheme maximizes each term of the sum $I(X; Y|U) \sim 0.5 * (I(X_d; Y_d, U_d) - I(X_d; U_d) + I(Y_d; X_d, U_d) - I(Y_d; U_d))$.

Discrete variables can be passed as factors and will be used "as is" to maximize each term.

Value

A list that contains :

- two vectors containing the cutpoints for each variable : *cutpoints1* corresponds to *x*, *cutpoints2* corresponds to *y*.
- *n_iterations* is the number of iterations performed before convergence of the (C)MI estimation.
- *iteration1, iteration2, ...*, lists containing the cutpoint vectors for each iteration.
- *info* and *infok*, the estimated (C)MI value and (C)MI minus the complexity cost.
- if *plot* == TRUE, a plot object (requires ggplot2 and gridExtra).

References

- Cabeli *et al.*, PLoS Comput. Biol. 2020, Learning clinical networks from medical records based on information estimates in mixed-type data

Examples

```
library(miic)
N <- 1000
# Dependence, conditional independence : X <- Z -> Y
Z <- runif(N)
X <- Z * 2 + rnorm(N, sd = 0.2)
Y <- Z * 2 + rnorm(N, sd = 0.2)
res <- discretizeMutual(X, Y, plot = FALSE)
message("I(X;Y) = ", res$info)
res <- discretizeMutual(X, Y, matrix_u = matrix(Z, ncol = 1), plot = FALSE)
message("I(X;Y|Z) = ", res$info)


# Conditional independence with categorical conditioning variable : X <- Z -> Y
Z <- sample(1:3, N, replace = TRUE)
X <- -as.numeric(Z == 1) + as.numeric(Z == 2) + 0.2 * rnorm(N)
Y <- as.numeric(Z == 1) + as.numeric(Z == 2) + 0.2 * rnorm(N)
res <- miic::discretizeMutual(X, Y, cplx = "nml")
message("I(X;Y) = ", res$info)
res <- miic::discretizeMutual(X, Y, matrix(Z, ncol = 1), is_continuous = c(TRUE, TRUE, FALSE))
message("I(X;Y|Z) = ", res$info)


# Independence, conditional dependence : X -> Z <- Y
X <- runif(N)
Y <- runif(N)
Z <- X + Y + rnorm(N, sd = 0.1)
res <- discretizeMutual(X, Y, plot = TRUE)
message("I(X;Y) = ", res$info)
res <- discretizeMutual(X, Y, matrix_u = matrix(Z, ncol = 1), plot = TRUE)
message("I(X;Y|Z) = ", res$info)
```

`estimateTemporalDynamic` *Estimation of the temporal causal discovery parameters*

Description

This function estimates the number of layers and number of time steps between each layer that are needed to cover the dynamic of a temporal dataset when reconstructing a temporal causal graph. Using autocorrelation decay, the function computes the average relaxation time of the variables and, based on a maximum number of nodes, deduces the number of layers and number of time steps between each layer to be used.

Usage

```
estimateTemporalDynamic(
  input_data,
```

```
  state_order = NULL,
  mov_avg = NULL,
  max_nodes = 50,
  verbose_level = 1
)
```

Arguments

input_data	[a data frame] A data frame containing the observational data. The expected data frame layout is variables as columns and time series/time steps as rows. The time step information must be supplied in the first column and, for each time series, be consecutive and in ascending order (increment of 1). Multiple trajectories can be provided, the function will consider that a new trajectory starts each time a smaller time step than the one of the previous row is encountered.
state_order	[a data frame] An optional data frame providing extra information about variables. It must have d rows where d is the number of input variables, excluding the time step one. For optional columns, if they are not provided or contain missing values, default values suitable for *input_data* will be used. The following structure (named columns) is expected: "var_names" (required) contains the name of each variable as specified by colnames(input_data), excluding the time steps column. "var_type" (optional) contains a binary value that specifies if each variable is to be considered as discrete (0) or continuous (1). Discrete variables will be excluded from the temporal dynamic estimation. "is_contextual" (optional) contains a binary value that specifies if a variable is to be considered as a contextual variable (1) or not (0). Contextual variables will be excluded from the temporal dynamic estimation. "mov_avg" (optional) contains an integer value that specifies the size of the moving average window to be applied to the variable. Note that if "mov_avg" column is present in the *state_order*, its values will overwrite the function parameter.
mov_avg	[an integer] Optional, NULL by default. When an integer>= 2 is supplied, a moving average operation is applied to all the non discrete and not contextual variables. If no *state_order* is provided, the discrete/continuous variables are deduced from the input data. If you want to apply a moving average only on specific columns, consider to use a *mov_avg* column in the *state_order* parameter.
max_nodes	[a positive integer] The maximum number of nodes in the final time-unfolded causal graph. The more nodes allowed in the temporal causal discovery, the more precise will be the discovery but at the cost of longer execution time. The default is set to 50 for fast causal discovery. On recent computers, values up to 200 or 300 nodes are usually possible (depending on the number of trajectories and time steps in the input data).
verbose_level	[an integer value in the range [0,2], 1 by default] The level of verbosity: 0 = no display, 1 = summary display, 2 = full display.

Value

A named list with two items:

- *n_layers*: the number of layers
- *delta_t*: the number of time steps between the layers

`export` *Export miic result for plotting (with igraph)*

Description

This function creates an object built from the result returned by `miic` that is ready to be fed to the plotting method.

Usage

```
export(
  miic_obj,
  method = "igraph",
  pcor_palette = NULL,
  display = "compact",
  show_self_loops = TRUE
)
```

Arguments

`miic_obj`	[a miic object, required] The object returned by the `miic` execution.
`method`	[a string, optional, default value "igraph"] The plotting method, currently only "igraph" is supported.
`pcor_palette`	[a color palette, optional, default value grDevices::colorRampPalette(c("blue", "darkgrey", "red")] Used to represent the partial correlations (the color of the edges). The palette must be able to handle 201 shades to cover the correlation range from -100 to +100.
`display`	[a string, optional, default value "compact"] Used only when exporting object returned by miic in temporal mode. It allows different representations of the temporal graph. Possible values are *"raw"*, *"lagged"*, *"compact"*, *"combine"*, *"unique"*, *"drop"*: • When *display* = *"raw"*, the export function will use the tmiic graph object as it, leading to the return of a lagged graph. • When *display* = *"lagged"*, the export function will repeat the edges over history assuming stationarity and return a lagged graph.

- When *display* = *"compact"*, the default, nodes and edges are converted into a flattened version to produce a compact view of the temporal network whilst still presenting all the information in the export.
  e.g. X_lag1->Y_lag0, X_lag2<-Y_lag0 become respectively X->Y lag=1, X<-Y lag=2.
- When *display* = *"combine"*, prior to the export, a pre-processing will be applied to kept only one edge per pair of nodes. The info_shifted will be the highest one of the summarized edges whilst the lag and orientation of the summarized edge will be an aggregation.
  e.g. X_lag2->Y_lag0, X_lag0<-Y_lag1 will become X<->Y lag=1-2 with the info_shifted of X_lag2->Y_lag0 if info_shifted of X_lag2->Y_lag0 > X_lag0<-Y_lag1.
- When *display* = *"unique"*, prior to the export, a pre-processing will be applied to kept only the edges having the highest info_shifted for a pair of nodes. If several edges between the sames nodes have the same info_shifted, then the edge kept is the one with the minimum lag.
  e.g. X_lag1->Y_lag0, X_lag0<-Y_lag2 with info_shifted of X_lag1->Y_lag0 > X_lag0<-Y_lag2 become X->Y lag=1.
- When *display* = *"drop"*, the same pre-processing as *"unique"* will be applied, then the lag information will be dropped before the export.

`show_self_loops`
: [a boolean, optional, TRUE by default]
  Used only when exporting object returned by miic in temporal mode. When TRUE, the lagged edges starting and ending on the same node are included in the igraph object. When FALSE, only edges having different nodes are present in the igraph object.

Details

The behavior depends on the method used for the export.

For igraph, edge attributes are passed to the igraph graph and can be accessed with e.g. `E(g)$partial_correlation`. See `miic` for more details on edge parameters. By default, edges are colored according to the partial correlation between two nodes conditioned on the conditioning set (negative is blue, null is gray and positive is red) and their width is based on the conditional mutual information minus the complexity cost.

Value

A graph object adapted to the method.

Examples

```
library(miic)
data(hematoData)

# execute MIIC (reconstruct graph)
miic_obj <- miic(
  input_data = hematoData, latent = "yes",
  n_shuffles = 10, conf_threshold = 0.001
```

```
)

# Using igraph
if(require(igraph)) {
g = export(miic_obj, "igraph")
plot(g) # Default visualisation, calls igraph::plot.igraph()

# Specifying layout (see ?igraph::layout_)
l <-layout_with_kk(g)
plot(g, layout=l)

# Override some graphical parameters
plot(g, edge.curved = .2)
plot(g, vertex.shape="none", edge.color="gray85", vertex.label.color="gray10")
}

# In temporal mode, execute MIIC
data(covidCases)
tmiic_obj <- miic(input_data = covidCases, mode = "TS", n_layers = 3, delta_t = 1, mov_avg = 14)

# Plot by default the compact display of the temporal network using igraph
if(require(igraph)) {
g = export (tmiic_obj)
plot(g)

# Plot the raw temporal network using igraph
g = export(tmiic_obj, display="raw")
plot(g)

# Plot the complete temporal network using igraph (completed by stationarity)
g = export(tmiic_obj, display="lagged")
plot(g)

# Specifying layout (see ?igraph::layout_)
l <- layout_on_grid(g, width = 5, height = 3, dim = 2)
plot(g, layout=l)

# For compact temporal display, please be aware that the rendering of
# igraph::plot.igraph() is not optimal when the graph contains
# multiple edges between the same nodes.
# So, the recommend way to plot a compact graph is to use tmiic plotting:
plot(tmiic_obj)
}
```

---

hematoData *Early blood development: single cell binary gene expression data*

Description

Binarized expression data of 33 transcription factors involved in early differentiation of primitive erythroid and endothelial cells (3934 cells).

Usage

```
data(hematoData)
```

Format

A data.frame object.

References

Moignard et al. (2015) Nat Biotechnol 33(3):269-76 (doi:10.1038/nbt.3154).

`miic` *MIIC, causal network learning algorithm including latent variables*

Description

MIIC (Multivariate Information-based Inductive Causation) combines constraint-based and information-theoretic approaches to disentangle direct from indirect effects amongst correlated variables, including cause-effect relationships and the effect of unobserved latent causes.

Usage

```
miic(
  input_data,
  state_order = NULL,
  true_edges = NULL,
  black_box = NULL,
  n_threads = 1,
  cplx = "nml",
  orientation = TRUE,
  ort_proba_ratio = 1,
  ort_consensus_ratio = NULL,
  propagation = FALSE,
  latent = "orientation",
  n_eff = -1,
  n_shuffles = 0,
  conf_threshold = 0,
  sample_weights = NULL,
  test_mar = TRUE,
  consistent = "no",
  max_iteration = 100,
  consensus_threshold = 0.8,
```

```
  negative_info = FALSE,
  mode = "S",
  n_layers = NULL,
  delta_t = NULL,
  mov_avg = NULL,
  keep_max_data = FALSE,
  max_nodes = 50,
  verbose = FALSE
)
```

Arguments

`input_data` [a data frame, required]

A n*d data frame (n samples, d variables) that contains the observational data.

In standard mode, each column corresponds to one variable and each row is a sample that gives the values for all the observed variables. The column names correspond to the names of the observed variables. Numeric columns with at least 5 distinct values will be treated as continuous by default whilst numeric columns with less than 5 distinct values, factors and characters will be considered as categorical.

In temporal mode, the expected data frame layout is variables as columns and time series/time steps as rows. The time step information must be supplied in the first column and, for each time series, be consecutive and in ascending order (increment of 1). Multiple trajectories can be provided, miic will consider that a new trajectory starts each time a smaller time step than the one of the previous row is encountered.

`state_order` [a data frame, optional, NULL by default]

A data frame providing extra information for variables. It must have d rows where d is the number of input variables and possible columns are described below. For optional columns, if they are not provided or contain missing values, default values suitable for *input_data* will be used.

"var_names" (required) contains the name of each variable as specified by colnames(input_data). In temporal mode, the time steps column should not be mentioned in the variables list.

"var_type" (optional) contains a binary value that specifies if each variable is to be considered as discrete (0) or continuous (1).

"levels_increasing_order" (optional) contains a single character string with all of the unique levels of the ordinal variable in increasing order, delimited by comma ','. It will be used during the post-processing to compute the sign of an edge using Spearman's rank correlation. If a variable is continuous or is categorical but not ordinal, this column should be NA.

"is_contextual" (optional) contains a binary value that specifies if a variable is to be considered as a contextual variable (1) or not (0). Contextual variables cannot be the child node of any other variable (cannot have edge with arrowhead pointing to them).

"is_consequence" (optional) contains a binary value that specifies if a variable is to be considered as a consequence variable (1) or not (0). Edges between

consequence variables are ignored, consequence variables cannot be the parent node of any other variable and cannot be used as contributors. Edges between a non consequence and consequence variables are pre-oriented toward the consequence.

Several other columns are possible in temporal mode:

"n_layers" (optional) contains an integer value that specifies the number of layers to be considered for the variable. Note that if a *"n_layers"* column is present in the *state_order*, its values will overwrite the function parameter.

"delta_t" (optional) contains an integer value that specifies the number of time steps between each layer for the variable. Note that if a *"delta_t"* column is present in the *state_order*, its values will overwrite the function parameter.

"mov_avg" (optional) contains an integer value that specifies the size of the moving average window to be applied to the variable. Note that if *"mov_avg"* column is present in the *state_order*, its values will overwrite the function parameter.

`true_edges` [a data frame, optional, NULL by default]

A data frame containing the edges of the true graph for computing performance after the run.
In standard mode, the expected layout is a two columns data frame, each row representing a true edge with in each column, the variable names. Variables names must exist in the *input_data* data frame.
In temporal mode, the expected layout is a three columns data frame, with the first two columns being variable names and the third the lag. Variables names must exist in the *input_data* data frame and the lag must be valid in the time unfolded graph. e.g. a row var1, var2, 3 is valid with *n_layers* = 4 + *delta_t* = 1 or *n_layers* = 2 + *delta_t* = 3 but not for *n_layers* = 2 + *delta_t* = 2 as there is no matching edge in the time unfolded graph.
Please note that the order is important: in standard mode, "var1 var2" will be interpreted as var1 -> var2 and in temporal mode, "var1 var2 3" is interpreted as var1_lag3 -> var2_lag0. Please note also that, in temporal mode, for contextual variables that are not lagged, the expected value in the third column for the time lag is NA.

`black_box` [a data frame, optional, NULL by default]

A data frame containing pairs of variables that will be considered as independent during the network reconstruction. In practice, these edges will not be included in the skeleton initialization and cannot be part of the final result.
In standard mode, the expected layout is a two columns data frame, each row representing a forbidden edge with in each column, the variable names. Variables names must exist in the *input_data* data frame.
In temporal mode, the expected layout is a three columns data frame, with the first two columns being variable names and the third the lag. Variables names must exist in the *input_data* data frame and the lag must be valid in the time unfolded graph. e.g. a row var1, var2, 3 is valid with *n_layers* = 4 + *delta_t* = 1 or *n_layers* = 2 + *delta_t* = 3 but not for *n_layers* = 2 + *delta_t* = 2 as there is no matching edge in the time unfolded graph. Please note that the order is important: var1, var2, 3 is interpreted as var1_lag3 - var2_lag0. Please note also that, for contextual variables that are not lagged, the expected value in the third column for the time lag is NA.

`n_threads` [a positive integer, optional, 1 by default]

When set greater than 1, n_threads parallel threads will be used for computation. Make sure your compiler is compatible with openmp if you wish to use multithreading.

`cplx` [a string, optional, "nml" by default, possible values: "nml", "bic"]

In practice, the finite size of the input dataset requires that the 2-point and 3-point information measures should be *shifted* by a *complexity* term. The finite size corrections can be based on the Bayesian Information Criterion (BIC). However, the BIC complexity term tends to underestimate the relevance of edges connecting variables with many different categories, leading to the removal of false negative edges. To avoid such biases with finite datasets, the (universal) Normalized Maximum Likelihood (NML) criterion can be used (see Affeldt 2015).

`orientation` [a boolean value, optional, TRUE by default]

The miic network skeleton can be partially directed by orienting edge directions, based on the sign and magnitude of the conditional 3-point information of unshielded triples and, in temporal mode, using time. If set to FALSE, the orientation step is not performed.

`ort_proba_ratio`

[a floating point between 0 and 1, optional, 1 by default]

The threshold when deducing the type of an edge tip (head/tail) from the probability of orientation. For a given edge tip, denote by p the probability of it being a head, the orientation is accepted if (1 - p) / p < *ort_proba_ratio*. 0 means reject all orientations, 1 means accept all orientations.

`ort_consensus_ratio`

[a floating point between 0 and 1, optional, NULL by default] Used to determine if orientations correspond to genuine causal edges and, when consistency is activated, to deduce the orientations in the consensus graph.
Oriented edges will be marked as genuine causal when: $(1 - p_{head})/p_{head} <$ *ort_consensus_ratio* and $p_{tail}/(1 - p_{tail}) <$ *ort_consensus_ratio*.
When consistency is activated, *ort_consensus_ratio* is used as threshold when deducing the type of an consensus edge tip (head/tail) from the average probability of orientations over the cycle of graphs. For a given edge tip, denote by p the average probability of it being a head, the orientation is accepted if (1 - p) / p < *ort_consensus_ratio*.
If not supplied, the *ort_consensus_ratio* will be initialized with the *ort_proba_ratio* value.

`propagation` [a boolean value, optional, FALSE by default]

If set to FALSE, the skeleton is partially oriented with only the v-structure orientations. Otherwise, the v-structure orientations are propagated to downstream un-directed edges in unshielded triples following the propagation procedure, relying on probabilities (for more details, see Verny 2017).

`latent` [a string, optional, "orientation" by default, possible values: "orientation", "no", "yes"]

When set to "yes", the network reconstruction is taking into account hidden (latent) variables. When set to "orientation", latent variables are not considered

during the skeleton reconstruction but allows bi-directed edges during the orientation. Dependence between two observed variables due to a latent variable is indicated with a '6' in the adjacency matrix and in the network edges.summary and by a bi-directed edge in the (partially) oriented graph.

`n_eff` [a positive integer, optional, -1 by default]

In standard mode, the n samples given in the *input_data* data frame are expected to be independent. In case of correlated samples such as in Monte Carlo sampling approaches, the effective number of independent samples *n_eff* can be estimated using the decay of the autocorrelation function (see Verny 2017). This effective number *n_eff* of independent samples can be provided using this parameter.

`n_shuffles` [a positive integer, optional, 0 by default]

The number of shufflings of the original dataset in order to evaluate the edge specific confidence ratio of all retained edges. Default is 0: no confidence cut is applied. If the number of shufflings is set to an integer > 0, the confidence threshold must also be > 0 (e.g. *n_shuffles* = 100 and *conf_threshold* = 0.01).

`conf_threshold` [a positive floating point, optional, 0 by default]

The threshold used to filter the less probable edges following the skeleton step (see Verny 2017). Default is 0: no confidence cut is applied. If the confidence threshold is set > 0, the number of shufflings must also be > 0 (e.g. *n_shuffles* = 100 and *conf_threshold* = 0.01).

`sample_weights` [a numeric vector, optional, NULL by default]

An vector containing the weight of each observation. If defined, it must be a vector of floats in the range [0,1] of size equal to the number of samples.

`test_mar` [a boolean value, optional, TRUE by default]

If set to TRUE, distributions with missing values will be tested with Kullback-Leibler divergence: conditioning variables for the given link $X - Y$, Z will be considered only if the divergence between the full distribution and the non-missing distribution $KL(P(X,Y)|P(X,Y)_{!NA})$ is low enough (with $P(X,Y)_{!NA}$ as the joint distribution of X and Y on samples which are not missing on Z. This is a way to ensure that data are missing at random for the considered interaction and detect bias due to values not missing at random.

`consistent` [a string, optional, "no" by default, possible values: "no", "orientation", "skeleton"]

If set to "orientation": iterate over skeleton and orientation steps to ensure consistency of the separating sets and all disconnected pairs in the final network. If set to "skeleton": iterate over skeleton step to get a consistent skeleton, then orient edges including inconsistent orientations (see Li 2019 for details).

`max_iteration` [a positive integer, optional, 100 by default]

When the *consistent* parameter is set to "skeleton" or "orientation", the maximum number of iterations allowed when trying to find a consistent graph.

`consensus_threshold`

[a floating point between 0.5 and 1.0, optional, 0.8 by default]

When the *consistent* parameter is set to "skeleton" or "orientation" and when the result graph is inconsistent or is a union of more than one inconsistent graphs,

a consensus graph will be produced based on a pool of graphs. If the result graph is inconsistent, then the pool is made of *max_iteration* graphs from the iterations, otherwise it is made of those graphs in the union. In the consensus graph, an edge is present when the proportion of non-zero status in the pool is above the threshold. For example, if the pool contains [A, B, B, 0, 0], where "A", "B" are different status of the edge and "0" indicates the absence of the edge. Then the edge is set to connected ("1") if the proportion of non-zero status (0.6 in the example) is equal to or higher than *consensus_threshold*. (When set to connected, the orientation of the edge will be further determined by the average probability of orientation.)

`negative_info` [a boolean value, optional, FALSE by default]

If TRUE, negative shifted mutual information is allowed during the computation when mutual information is inferior to the complexity term. For small dataset with complicated structures, e.g. discrete variables with many levels, allowing for negative shifted mutual information may help identifying weak v-structures related to those discrete variables, as the negative three-point information in those cases will come from the difference between two negative shifted mutual information terms (expected to be negative due to the small sample size). However, under this setting, a v-structure (X -> Z <- Y) in the final graph does not necessarily imply that X is dependent on Y conditioning on Z, As a consequence, the reliability of certain orientations is not guaranteed. By contrast, keeping this parameter as FALSE is more conservative and leads to more reliable orientations (see Cabeli 2021 and Ribeiro-Dantas 2024).

`mode` [a string, optional, "S" by default, possible values are "S": Standard (non temporal data) or "TS": Temporal Stationary data]

When temporal mode is activated, the time information must be provided in the first column of *input_data*. For more details about temporal stationary mode (see Simon 2024).

`n_layers` [an integer, optional, NULL by default, must be >= 2 if supplied]

Used only in temporal mode, *n_layers* defines the number of layers that will be considered for the variables in the time unfolded graph. The layers will be distant of *delta_t* time steps. If not supplied, the number of layers is estimated from the dynamic of the dataset and the maximum number of nodes *max_nodes* allowed in the final lagged graph.

`delta_t` [an integer, optional, NULL by default, must be >= 1 if supplied]

Used only in temporal mode, *delta_t* defines the number of time steps between each layer. i.e. on 1000 time steps with *n_layers* = 3 and *delta_t* = 7, the time steps kept for the samples conversion will be 1, 8, 15 for the first sample, the next sample will use 2, 9, 16 and so on. If not supplied, the number of time steps between layers is estimated from the dynamic of the dataset and the number of layers.

`mov_avg` [an integer, optional, NULL by default, must be >= 2 if supplied]

Used only in temporal mode. When supplied, a moving average operation is applied to all integer and numeric variables that are not contextual variables.

`keep_max_data` [a boolean value, optional, FALSE by default]

Used only in temporal mode. If TRUE, rows where some NAs have been introduced during the moving averages and lagging will be kept whilst they will be dropped if FALSE.

`max_nodes` [an integer, optional, 50 by default]

Used only in temporal mode and if the *n_layers* or *delta_t* parameters are not supplied. *max_nodes* is used as the maximum number of nodes in the final time-unfolded graph to compute *n_layers* and/or *delta_t*. The default is 50 to produce quick runs and can be increased up to 200 or 300 on recent computers to produce more precise results.

`verbose` [a boolean value, optional, FALSE by default]

If TRUE, debugging output is printed.

Details

Starting from a complete graph, the method iteratively removes dispensable edges, by uncovering significant information contributions from indirect paths, and assesses edge-specific confidences from randomization of available data. The remaining edges are then oriented based on the signature of causality in observational data. Miic distinguishes genuine causal edges (with both reliable arrow heads and tails) from putative causal edges (with one reliable arrow head only) and latent causal edges (with both reliable arrow heads). (see Ribeiro-Dantas 2024)

In temporal mode, miic reorganizes the dataset using the *n_layers* and *delta_t* parameters to transform the time steps into lagged samples. As starting point, a lagged graph is created with only edges having at least one node laying on the last time step. Then, miic standard algorithm is applied to remove dispensable edges. The remaining edges are then duplicated to ensure time invariance (stationary dynamic) and oriented using the temporality and the signature of causality in observational data. The use of temporal mode is presented in Simon 2024.

The method relies on information theoretic principles which replace (conditional) independence tests as described in Affeldt 2015, Cabeli 2020, Cabeli 2021 and Ribeiro-Dantas 2024. It deals with both categorical and continuous variables by performing optimal context-dependent discretization. As such, the input data frame may contain both numerical columns which will be treated as continuous, or character / factor columns which will be treated as categorical. For further details on the optimal discretization method and the conditional independence test, see the function discretizeMutual. The user may also choose to run miic with scheme presented in Li 2019 and Ribeiro-Dantas 2024 to improve the end result's interpretability by ensuring consistent separating sets.

Value

A *miic-like* object that contains:

- *summary:* a data frame with information about the relationship between relevant pair of variables.

  As returning the information on all possible pairs of variables could lead to an huge data frame, by convention, the summary does not include pair of variables not sharing information at all (*I'(x,y) <= 0*). However, as exception to this convention, when a ground truth is supplied (using the *true_edges* parameter), the edges that are not retained by MIIC because the variables does not share information at all but are present in the true edges will be included in the summary to report correctly all the false negative edges.

  So, the summary contains these categories of edges:

- edges retained
- edges not retained after conditioning on some contributor(s)
- edges not retained without conditioning but present in true edges

while these edges are not considered as relevant and are not included:

- edges not retained without conditioning and not in true edges

Information available in the summary are:

- *x:* X node name
- *y:* Y node name
- *type:* contains 'N' if the edge has been removed or 'P' for retained edges. If the true graph is supplied in the *true_edges* parameter, 'P' becomes 'TP' (True Positive) or 'FP' (False Positive), while 'N' becomes 'TN' (True Negative) or 'FN' (False Negative). Note that, as the *summary* does not contain all the removed edges, edges not present have to be considered as 'N' and, if the true graph is supplied, as 'TN'.
- *ai:* the contributing nodes found by the method which contribute to the mutual information between *x* and *y*, and possibly separate them.
- *raw_contributions:* describes the share of total mutual information between *x* and *y* explained by each contributor, measured by I'(x;y;ai|{aj}) / I'(x;y), where {aj} is the separating set before adding ai.
- *contributions:* describes the share of remaining mutual information between *x* and *y* explained by each successive contributors, measured by I'(x;y;ai|{aj}) / I'(x;y|{aj}), where {aj} is the separating set before adding ai.
- *info:* the mutual information *I(x;y)* times *n_xy*, the number of samples without missing or NA values for both *x* and *y*.
- *n_xy:* gives the number of samples on which the information without conditioning has been computed. If the input dataset has no missing value, the number of samples is the same for all pairs and corresponds to the total number of samples.
- *info_cond:* the conditional mutual information *I(x;y|ai)* times the number of samples without NA *n_xy_ai* used in the computation. *info_cond* is equal to *info* when *ai* is an empty set.
- *cplx:* the complexity term for the pair (*x*, *y*) taking into account the contributing nodes *ai*.
- *n_xy_ai:* the number of samples without NA in *x*, *y* and all nodes in *ai* on which the information and the complexity terms are computed. If the input dataset has no missing value, the number of samples is the same for all pairs and corresponds to the total number of samples.
- *info_shifted:* value equal to *info_cond* - *cplx*. Used to decide whether the edge is retained (when positive), or removed (when zero or possibly negative when the parameter *negative_info* is set to TRUE).
- *ort_inferred:* the orientation of the edge (*x*, *y*). 0: edge removed, 1: un-directed, 2: directed from X to Y, -2: directed from Y to X, 6: bi-directed.
  When the *consistent* option is turned on and there is more than one graph in the consistent cycle, this is the inferred orientation of the edge in the last graph in the cycle.
- *ort_ground_truth:* the orientation of the edge (*x*, *y*) in the ground truth graph when true edges are provided.
- *is_inference_correct:* indicates if the inferred orientation agrees with the provided ground truth. TRUE: agrees, FALSE: disagrees and set to NA when no ground truth is supplied.

  - *is_causal:* boolean value indicating the causal nature of the arrow tips of an edge, based on the probabilities given in the columns *p_y2x* and *p_x2y*. TRUE: when the edges is directed and both the head and the tail are set with high confidence (adjustable with the *ort_consensus_ratio* parameter), FALSE otherwise or NA if the edge is not retained. More formally, an oriented edge is marked as genuine causal when $(1 - p_{head})/p_{head} <$ *ort_consensus_ratio* and $p_{tail}/(1 - p_{tail}) <$ *ort_consensus_ratio*.
    A directed edge not marked as genuine causal indicates that only the head is set with high confidence, while the tail remains uncertain. This corresponds to a putative causal edge, which could either be a genuine causal edge or a bi-directed edge from a latent confounder.
    Note that the genuine causality is deducible only when latent variables are allowed and propagation is not allowed.
  - *ort_consensus:* Not computed (NAs) when consistency is not activated or, when consistency is on, if there is only one graph returned (no cycle). When computed, indicates the consensus orientation of the edge determined from the consensus skeleton and the *ort_consensus_ratio* threshold on averaged orientation probabilities over the cycle of graphs. Possible values are 0: not connected, 1: un-directed, -2 or 2: directed and 6: bi-directed (latent variable).
  - *is_causal_consensus:* Not computed (NAs) when consistency is not activated or, when consistency is on, if there is only one graph returned (no cycle). When computed, work in the same way as *is_causal* but on the consensus graph.
  - *edge_stats:* Not computed (NAs) when consistency is not activated or, when consistency is on, if there is only one graph returned (no cycle). When computed, contains the frequencies of all *ort_inferred* values present in the cycle of graphs for the edge (*x, y*), in the format [percentage(orientation)], separated by ";". e.g. In a cycle of 4 graphs, if an edge is three times marked as 2 (directed) and one time marked as 1 (un-directed), edge_stats will contain "75%(2);25%(1)".
  - *sign:* the sign of the partial correlation between variables *x* and *y*, conditioned on the contributing nodes *ai*.
  - *partial_correlation:* value of the partial correlation for the edge (*x, y*) conditioned on the contributing nodes *ai*.
  - *p_y2x:* probability of the arrowhead from *y* to *x*, of the inferred orientation, derived from the three-point mutual information (see Verny 2017 and Ribeiro-Dantas 2024). NA if the edge is removed.
  - *p_x2y:* probability of the arrowhead from *x* to *y*, of the inferred orientation, derived from the three-point mutual information (see Verny 2017 and Ribeiro-Dantas 2024). NA if the edge is removed.
  - *confidence:* computed only when the confidence cut is activated, NA otherwise. When computed, it corresponds to a measure of the strength of the retained edges: it is the ratio between the probability to reject the edge *exp(-info_shifted(x;y|ai))* in the original dataset and the mean probability to do the same in *n_shuffles* number of randomized datasets. Edges with *confidence* > *conf_threshold* will be filtered out from the graph. (see parameters *n_shuffles* and *conf_threshold*)

- *edges:* a data frame with the raw edges output coming from the C++ core function. This data frame is used internally by MIIC to produce the summary and contains all pairs of variables (*x, y*).

- *triples:* this data frame lists the orientation probabilities of the two edges of all unshielded triples of the reconstructed network with the structure: node1 – mid-node – node2:
  - *node1:* node at the end of the unshielded triplet
  - *p1:* probability of the arrowhead node1 <- mid-node
  - *p2:* probability of the arrowhead node1 -> mid-node
  - *mid-node:* node at the center of the unshielded triplet
  - *p3:* probability of the arrowhead mid-node <- node2
  - *p4:* probability of the arrowhead mid-node -> node2
  - *node2:* node at the end of the unshielded triplet
  - *ni3:* 3 point (conditional) mutual information * N
  - *conflict:* indicates if there is a conflict between the computed probabilities and the *ni3* value
- *adj_matrix:* the adjacency matrix is a square matrix used to represent the inferred graph. The entries of the matrix indicate whether pairs of vertices are adjacent or not in the graph. The matrix can be read as a (row, column) set of couples where the row represents the source node and the column the target node. Since miic can reconstruct mixed networks (including directed, un-directed and bi-directed edges), we will have a different digit for each case:
  - 1: (x, y) edge is un-directed
  - 2: (x, y) edge is directed as x -> y
  - -2: (x, y) edge is directed as x <- y
  - 6: (x, y) edge is bi-directed
- *proba_adj_matrix:* the probability adjacency matrix is a square matrix used to represent the orientation probabilities associated to the edges tips. The value at ("row", "column") is the probability, for the edge between "row" and "column" nodes, of the edge tip on the "row" side. A probability less than 0.5 is an indication of a possible tail (cause) and a probability greater than 0.5 a possible head (effect).
- *adj_matrices:* present only when consistency is activated. The list of the adjacency matrices, one for each graph which is part of the resulting cycle of graphs. Each item is a square matrix with the same layout as *adj_matrix*.
- *proba_adj_matrices:* present only when consistency is activated. The list of the probability adjacency matrices, one for each graph which is part of the resulting cycle of graphs. Each item is a square matrix with the same layout as *proba_adj_matrix*.
- *proba_adj_average:* present only when consistency is activated. The average probability adjacency matrix is a square matrix used to represent the orientation probabilities associated to the edges tips of the consensus graph. Its layout is the same as *proba_adj_matrix* and it contains the averaged probability of edges tips over the resulting cycle of graphs.
- *is_consistent:* present only when consistency is activated. TRUE if the returned graph is consistent, FALSE otherwise.
- *time:* execution time of the different steps and total run-time of the causal graph reconstruction by MIIC.
- *interrupted:* TRUE if causal graph reconstruction has been interrupted, FALSE otherwise.
- *scores:* present only when true edges have been supplied. Contains the scores of the returned graph in regard of the ground truth:

  - *tp*: number of edges marked as True Positive
  - *fp*: number of edges marked as False Positive
  - *fn*: number of edges marked as False Negative
  - *precision*: Precision
  - *recall*: Recall
  - *fscore*: F1-Score

- *params:* the list of parameters used for the network reconstruction. The parameters not supplied are initialized to their default values. Otherwise, the parameters are checked and corrected if necessary.
- *state_order:* the state order used for the network reconstruction. If no state order is supplied, it is generated by using default values. Otherwise, it is the state order checked and corrected if necessary.
- *black_box:* present only if a black box has been supplied, the black box, checked and corrected if necessary, used for the network reconstruction.
- *true_edges:* present only if the true edges have been supplied, the true edges, checked and corrected if necessary, used for the network evaluation.
- *tmiic:* present only in temporal mode. Named list containing the full list of edges completed by stationarity, the lagged state order and, if a black box or true edges have been supplied, the lagged versions of these inputs.

References

- Simon *et al.*, eLife 2024, CausalXtract: a flexible pipeline to extract causal effects from live-cell time-lapse imaging data
- Ribeiro-Dantas *et al.*, iScience 2024, Learning interpretable causal networks from very large datasets, application to 400,000 medical records of breast cancer patients
- Cabeli *et al.*, NeurIPS 2021, Reliable causal discovery based on mutual information supremum principle for finite dataset
- Cabeli *et al.*, PLoS Comput. Biol. 2020, Learning clinical networks from medical records based on information estimates in mixed-type data
- Li *et al.*, NeurIPS 2019, Constraint-based causal structure learning with consistent separating sets
- Verny *et al.*, PLoS Comput. Biol. 2017, Learning causal networks with latent variables from multivariate information in genomic data
- Affeldt *et al.*, UAI 2015, Robust Reconstruction of Causal Graphical Models based on Conditional 2-point and 3-point Information

See Also

`discretizeMutual` for optimal discretization and (conditional) independence test.

Examples

```
library(miic)

# EXAMPLE HEMATOPOIESIS
data(hematoData)

# execute MIIC (reconstruct graph)
miic_obj <- miic(
  input_data = hematoData[1:1000,], latent = "yes",
  n_shuffles = 10, conf_threshold = 0.001
)

# plot graph
if(require(igraph)) {
 plot(miic_obj, method="igraph")
}


# write graph to graphml format. Note that to correctly visualize
# the network we created the miic style for Cytoscape (http://www.cytoscape.org/).

writeCytoscapeNetwork(miic_obj, file = file.path(tempdir(), "temp"))

# EXAMPLE CANCER
data(cosmicCancer)
data(cosmicCancer_stateOrder)
# execute MIIC (reconstruct graph)
miic_obj <- miic(
  input_data = cosmicCancer, state_order = cosmicCancer_stateOrder, latent = "yes",
  n_shuffles = 100, conf_threshold = 0.001
)

# plot graph
if(require(igraph)) {
 plot(miic_obj)
}

# write graph to graphml format. Note that to correctly visualize
# the network we created the miic style for Cytoscape (http://www.cytoscape.org/).
writeCytoscapeNetwork(miic_obj, file = file.path(tempdir(), "temp"))

# EXAMPLE COVID CASES (time series demo)
data(covidCases)
# execute MIIC (reconstruct graph in temporal mode)
tmiic_obj <- miic(input_data = covidCases, mode = "TS", n_layers = 3, delta_t = 1, mov_avg = 14)

# to plot the default graph (compact)
if(require(igraph)) {
 plot(tmiic_obj)
}

# to plot the raw temporal network
```

```
if(require(igraph)) {
  plot(tmiic_obj, display="raw")
}

# to plot the full temporal network
if(require(igraph)) {
  plot(tmiic_obj, display="lagged")
}
```

`plot.miic` *Basic plot function of a miic network inference result*

Description

This function calls `export` to build a plottable object from the result returned by `miic` and plot it.

Usage

```
## S3 method for class 'miic'
plot(x, method = "igraph", pcor_palette = NULL, ...)
```

Arguments

`x`	[a miic object, required] The object returned by `miic` execution.
`method`	[a string, optional, default value "igraph"] The plotting method, currently only "igraph" is supported.
`pcor_palette`	[a color palette, optional, default value grDevices::colorRampPalette(c("blue", "darkgrey", "red")] Used to represent the partial correlations (the color of the edges). The palette must be able to handle 201 shades to cover the correlation range from -100 to +100.
`...`	Additional plotting parameters. See the corresponding plot function for the complete list. For igraph, see `igraph.plotting`.

Details

See the documentation of `export` for further details.

See Also

`export` for graphical exports, `igraph.plotting`

`plot.tmiic` *Basic plot function of a temporal miic (tmiic) network inference result*

Description

This function calls `export` to build a plottable object from the result returned by `miic` in temporal mode and plot it.

Usage

```
## S3 method for class 'tmiic'
plot(
  x,
  method = "igraph",
  pcor_palette = NULL,
  display = "compact",
  show_self_loops = TRUE,
  positioning_for_grid = "greedy",
  orientation_for_grid = "L",
  ...
)
```

Arguments

`x`	[a tmiic object, required] The object returned by `miic` in temporal mode.
`method`	[a string, optional, default value "igraph"] The plotting method, currently only "igraph" is supported.
`pcor_palette`	[a color palette, optional, default value grDevices::colorRampPalette(c("blue", "darkgrey", "red")] Used to represent the partial correlations (the color of the edges). The palette must be able to handle 201 shades to cover the correlation range from -100 to +100.
`display`	[a string, optional, default value "compact"] Possible values are *"raw"*, *"lagged"*, *"compact"*, *"combine"*, *"unique"*, *"drop"*: • When *display* = *"raw"*, the plot function will use the tmiic graph object as it, leading to the display of a lagged graph. Unless a specific layout is specified, nodes will be positioned on a grid. • When *display* = *"lagged"*, the function will repeat the edges over history assuming stationarity and plot a lagged graph. Unless a specific layout is specified, nodes will be positioned on a grid. • When *display* = *"compact"*, the default, nodes and edges are converted into a flattened version to produce a compact view of the temporal network whilst still presenting all the information in the plotting. e.g. X_lag1->Y_lag0, X_lag2<-Y_lag0 become respectively X->Y lag=1, X<-Y lag=2.

- When *display* = *"combine"*, prior to the plotting, a pre-processing will be applied to kept only one edge per pair of nodes. The info_shifted will be the highest one of the summarized edges whilst the lag and orientation of the summarized edge will be an aggregation.
  e.g. X_lag1->Y_lag0, X_lag2<-Y_lag0 will become X<->Y lag=1,2 with the info_shifted of X_lag1->Y_lag0 if info_shifted of X_lag1->Y_lag0 > X_lag2<-Y_lag0.
- When *display* = *"unique"*, prior to the plotting, a pre-processing will be applied to kept only the edges having the highest info_shifted for a pair of nodes. If several edges between the sames nodes have the same info_shifted, then the edge kept is the one with the minimum lag.
  e.g. X_lag1->Y_lag0, X_lag2<-Y_lag0 with info_shifted of X_lag1->Y_lag0 > X_lag2<-Y_lag0 become X->Y lag=1.
- When *display* = *"drop"*, the same pre-processing as *"unique"* will be applied, then the lag information will be dropped and will not be displayed on the final plotting.

`show_self_loops`

[a boolean, optional, TRUE by default]

When TRUE, the lagged edges starting and ending on the same node are included in the igraph object. When FALSE, only edges having different nodes are present in the igraph object.

`positioning_for_grid`

[a string, optional, "greedy" by default]

Used only when the display is "raw" or "lagged" and no layout is supplied. Possible values are *"none"*, *"alphabetical"*, *"layers"*, *"greedy"* and *"sugiyama"*

- When *positioning_for_grid* = *"none"* The nodes are positioned as they appear in the miic result
- When *positioning_for_grid* = *"alphabetical"* The nodes are positioned alphabetically in ascending order
- When *positioning_for_grid* = *"layers"* The nodes with the less lags will be placed on the exteriors while the nodes having the most lags are in the center
- When *positioning_for_grid* = *"greedy"* A greedy algorithm will be used to placed the nodes in a way minimizing the crossing edges
- When *positioning_for_grid* = *"sugiyama"* The sugiyama algorithm will be used to placed the nodes in a way minimizing the crossing edges

`orientation_for_grid`

[a string, optional, "L" by default]

Used only when the display is "raw" or "lagged and no layout is supplied. Indicates the orientation of the draw, possible values are landscape: "L" or portrait: "P".

`...`

Additional plotting parameters. See the corresponding plot function for the complete list.

For igraph, see `igraph.plotting`.

Details

See the documentation of `export` for further details.

See Also

`export` for graphical exports, `igraph.plotting`

Examples

```
library(miic)

#' # EXAMPLE COVID CASES (time series demo)
data(covidCases)
# execute MIIC (reconstruct graph in temporal mode)
tmiic_obj <- miic(input_data = covidCases, mode = "TS", n_layers = 3, delta_t = 1, mov_avg = 14)

# to plot the default compact graph
if(require(igraph)) {
  plot(tmiic_obj)
}

# to plot the raw temporal network
if(require(igraph)) {
  plot(tmiic_obj, display="raw")
}

# to plot the full temporal network
if(require(igraph)) {
  plot(tmiic_obj, display="lagged")
}
```

---

writeCytoscapeNetwork *GraphML converting function for miic graph*

Description

Convert miic graph to GraphML format.

Usage

```
writeCytoscapeNetwork(miic_obj, file, layout = NULL)
```

Arguments

`miic_obj`	A miic object. The object returned by the `miic` execution.
`file`	A string. Path to the output file containing file name without extension (.graphml will be appended).

layout	An optional data frame of 2 (or 3) columns containing the coordinate x and y for each node. The optional first column can contain node names. If node names is not given, the order of the input file will be assigned to the list of positions.

Value

None

`writeCytoscapeStyle` *Style writing function for the miic network*

Description

This function writes the miic style for a correct visualization using the cytoscape tool (http://www.cytoscape.org/).

Usage

```
writeCytoscapeStyle(file)
```

Arguments

file	[a string] The file path of the output file (containing the file name without extension).

Details

The style is written in the xml file format.

Value

None

Index

∗ **datasets**
cosmicCancer, 7
cosmicCancer_stateOrder, 8
covidCases, 9
hematoData, 16
∗ **data**
cosmicCancer, 7
cosmicCancer_stateOrder, 8
covidCases, 9
hematoData, 16

computeMutualInfo, 3, *6*
computeThreePointInfo, 5
cosmicCancer, 7
cosmicCancer_stateOrder, 8
covidCases, 9

discretizeMDL, 9
discretizeMutual, 10, *27*

estimateTemporalDynamic, 12
export, 14, *29, 30*, *32*

hematoData, 16

igraph.plotting, *29*, *31, 32*

miic, *14, 15*, 17, *29, 30*, *32*

plot.miic, 29
plot.tmiic, 30

writeCytoscapeNetwork, 32
writeCytoscapeStyle, 33