Package ‘BiCausality’

May 26, 2022

Title Binary Causality Inference Framework

Version 0.1.1

Maintainer Chainarong Amornbunchornvej <grandca@gmail.com>

Description A framework to infer causality on binary data using techniques in frequent pattern mining and estimation statistics. Given a set of individual vectors S={x} where x(i) is a realization value of binary variable i, the framework infers empirical causal relations of binary variables i,j from S in a form of causal graph G=(V,E) where V is a set of nodes representing binary variables and there is an edge from i to j in E if the variable i causes j. The framework determines dependency among variables as well as analyzing confounding factors before deciding whether i causes j. The publication of this package is at <arXiv:2205.06131>.

URL https://github.com/DarkEyes/BiCausality

BugReports https://github.com/DarkEyes/BiCausality/issues

Depends R (>= 3.5.0)

Encoding UTF-8

LazyData TRUE

Suggests knitr, rmarkdown, markdown, igraph

VignetteBuilder knitr

RoxygenNote 7.1.2

NeedsCompilation no

Author Chainarong Amornbunchornvej [aut, cre]
(<https://orcid.org/0000-0003-3131-0370>)

Repository CRAN

Date/Publication 2022-05-26 13:30:15 UTC

R topics documented:

`adjustmentProb` *adjustmentProb function*

Description

This function evaluates the P(Y=yflag|do(X=xflag)) given only marginal distributions using parent adjustment method.

Usage

```
adjustmentProb(EValHat, mat, yflag = 1, xflag = 1)
```

Arguments

`EValHat`	is an adjacency matrix of weighted directed causal graph where edge weights are P(Y=yflag|X=xflag) or probabilities of effect being 1 given cause being either 1 for positive association or 0 for negative association.
`mat`	is a matrix n by d where n is a number of transactions or samples and d is a number of dimensions.
`yflag`	is value set for Y in P(Y=yflag|X=xflag,z) for the adjustment method.
`xflag`	is value set for X in P(Y=yflag|X=xflag,z) for the adjustment method.

Value

This function returns an adjacency matrix of weighted directed causal graph where the edge weights are P(Y=yflag|do(X=xflag)).

Examples

```
adjustmentProb(resC$CausalGRes$EValHat,mat)
```

`assocSignTest` *indpFunc function*

Description

This function provides association signs (positive/negative association) inference between i and j. If there is a positive association, it implies i and j trend to have a similar value. For a negative association, however, i and j trend to have an opposite value.

Usage

```
assocSignTest(mat, i, j, z = c(), alpha = 0.05, IndpThs = 0.05, nboot = 100)
```

Arguments

`mat`	is a matrix n by d where n is a number of transactions or samples and d is a number of dimensions.
`i`	is an ith dimension in `mat`.
`j`	is an jth dimension in `mat`.
`z`	is a conditioning d-dimensional vector on `mat`. Given k non-negative-bit positions of `z`, all k bit positions of samples in the subset of `mat` must have similar values with these bits.
`alpha`	is a significance threshold for hypothesis tests (Mann Whitney) that deploys for testing degrees of dependency, association direction, and causal direction. The default is 0.5.
`IndpThs`	is a threshold for the degree of dependency. In the independence test, to claim that any variables are dependent, the dependency degree must greater than this value significantly. The default is 0.05.
`nboot`	is a number of bootstrap replicates for bootstrapping deployed to infer confidence intervals and distributions for hypothesis tests. The default is 100.

Value

This function returns results of inference of association signs (positive/negative association) between i and j.

`bmean`	A mean of sign dependency degrees between variables i and j.
`confInv`	An `alpha`*100th percentile confidence interval of sign dependency degrees between variables i and j.
`testRes`	A Mann-Whitney hypothesis test result for an independence test between variables i and j. The null hypothesis is that the distributions of dependency degrees of i,j differ by a location shift of `IndpThs` and the alternative is that distributions of dependency degrees of i,j is shifted greater than `IndpThs`.

Examples

```
assocSignTest(mat=mat,i=1,j=2)
```

`bin2dec` *bin2dec function*

Description

This function convertes a binary vector into its decimal value.

Usage

```
bin2dec(X)
```

Arguments

`X`	is a binary vector where `X[i]` is the ith bit of vector.

Value

This function returns a decimal value of `X`.

Examples

```
bin2dec(X=c(1,1,1,0))
```

`bIndpTest` *bIndpTest function*

Description

This function infers dependency for a pair of variables i,j with bootstrapping.

Usage

```
bIndpTest(
  mat,
  i,
  j,
  z = c(),
  alpha = 0.05,
  IndpThs = 0.05,
  nboot = 100,
  pflag = FALSE
)
```

Arguments

`mat`	is a matrix n by d where n is a number of transactions or samples and d is a number of dimensions.
`i`	is an ith dimension in `mat`.
`j`	is an jth dimension in `mat`.
`z`	is a conditioning d-dimensional vector on `mat`. Given k non-negative-bit positions of `z`, all k bit positions of samples in the subset of `mat` must have similar values with these bits.
`alpha`	is a significance threshold for hypothesis tests (Mann Whitney) that deploys for testing degrees of dependency, association direction, and causal direction. The default is 0.5.
`IndpThs`	is a threshold for the degree of dependency. In the independence test, to claim that any variables are dependent, the dependency degree must greater than this value significantly. The default is 0.05.
`nboot`	is a number of bootstrap replicates for bootstrapping deployed to infer confidence intervals and distributions for hypothesis tests. The default is 100.
`pflag`	is a flag for printing progress message (TRUE). The default is FALSE (no printing).

Value

This function returns results of dependency inference between i and j.

`bmean`	A mean of dependency degrees between variables i and j.

confInv — An alpha*100th percentile confidence interval of dependency degrees between variables i and j.

testRes — A Mann-Whitney hypothesis test result for an independence test between variables i and j. The null hypothesis is that the distributions of dependency degrees of i,j differ by a location shift of IndpThs and the alternative is that distributions of dependency degrees of i,j is shifted greater than IndpThs.

Examples

```
bIndpTest(mat=mat,i=1,j=2)
```

---

bSCMCausalGraphFunc *bSCMCausalGraphFunc function*

Description

This function infers a causal graph from a result of confounding factor filtering by bSCMdeConfoundingGraphFunc().

Usage

```
bSCMCausalGraphFunc(E1, Dboot, alpha = 0.05, SignThs = 0.05, CausalThs = 0.25)
```

Arguments

E1 — is an adjacency matrix of undirected graph after filtering associations without true causal directions from any confounding factor.

Dboot — is a list of Ds (aligned list of transactions) that are generated from sampling with replacement on input samples (mat) nboot times.

alpha — is a significance threshold for hypothesis tests (Mann Whitney) that deploys for testing degrees of dependency, association direction, and causal direction. The default is 0.5.

SignThs — is a threshold for the degree of dependency for association direction inference. In the independence test of sign direction, to claim that any variables are dependent, the dependency degree must greater than this value significantly. The default is 0.05.

CausalThs — is a threshold for the degree of causal direction In the causal-direction test, to claim that any variables have causal relations, the degree of causal direction must greater than this value significantly. The default is 0.1.

Value

This function returns causal inference results from E1 matrix that is an output of `bSCMdeConfoundingGraphFunc`.

`Ehat` An adjacency matrix of directed causal graph where `CausalGRes$Ehat[i,j]=1` implies i causes j.

`EValHat` An adjacency matrix of weighted directed causal graph where edge weights are estimated means of probabilities of effect being 1 given cause being either 1 for positive association or 0 for negative association using CondProb() and bootstrapping to estimate.

`i` An index

`j` An index

`causalInfo[['i,j']]$CDirConfValInv`
An `alpha`*100th percentile confidence interval of estimated conditional probability of effect j being 1/0 given cause i's value being either the same (positive association) or opposite (negative association).

`causalInfo[['i,j']]$CDirConfInv`
An `alpha`*100th percentile confidence interval of estimated causal direction degree of i cause j.

`causalInfo[['i,j']]$CDirmean`
A mean-estimated-causal-direction degree of i cause j.

`causalInfo[['i,j']]$testRes2`
A Mann-Whitney hypothesis test result for existence of causal direction. The null hypothesis is that the distributions of causal-direction degrees of i,j differ by a location shift of `CausalThs` and the alternative is that distributions of causal-direction degrees of i,j is shifted greater than `CausalThs`.

`causalInfo[['i,j']]$testRes1`
A Mann-Whitney hypothesis test result for existence of association by odd differences from `oddDiffFunc()`. The null hypothesis is that the distributions of absolute odd difference of i,j differ by a location shift of `IndpThs` and the alternative is that distributions of absolute odd difference of i,j is shifted greater than `IndpThs`.

`causalInfo[['i,j']]$sign`
A direction of i,j association: 1 for positive, 0 for negative, and -1 for no association.

`causalInfo[['i,j']]$SignConfInv`
An `alpha`*100th percentile confidence interval of i,j odd difference from bootstrapping.

`causalInfo[['i,j']]$Signmean`
A mean of i,j odd difference from bootstrapping.

Examples

```
bSCMCausalGraphFunc(resC$ConfoundRes$E1,resC$depRes$Dboot)
```

bSCMdeConfoundingGraphFunc

bSCMdeConfoundingGraphFunc function

Description

This function removes any association/dependency of variables i,j that have any confounding factor k s.t. given k, i and j are independent.

Usage

```
bSCMdeConfoundingGraphFunc(dat, IndpThs = 0.05, alpha = 0.05)
```

Arguments

`dat`	is the result of inferring dependencies between all pairs of variables from `bSCMDepndentGraphFunc()`.
`IndpThs`	is a threshold for the degree of dependency. In the independence test, to claim that any variables are dependent, the dependency degree must greater than this value significantly. The default is 0.05.
`alpha`	is a significance threshold for hypothesis tests (Mann Whitney) that deploys for testing degrees of dependency, association direction, and causal direction. The default is 0.5.

Value

This function returns an adjacency matrix of dependencies that have no confounding factors.

`E1`	An adjacency matrix of undirected graph after filtering associations without true causal directions from any confounding factor.
`E2`	A matrix of associations that have confounding factors where `E2[i,j]=0` if no confounding factor and `E2[i,j]=k` if k is a confounding factor of i and j.

Examples

```
bSCMdeConfoundingGraphFunc(resC$depRes)
```

bSCMDepndentGraphFastFunc — *bSCMDepndentGraphFastFunc function*

Description

This function infers dependencies for all pairs of variables without bootstrapping.

Usage

```
bSCMDepndentGraphFastFunc(mat, IndpThs = 0.05)
```

Arguments

`mat`	is a matrix n by d where n is a number of transactions or samples and d is a number of dimensions.
`IndpThs`	is a threshold for the degree of dependency. In the independence test, to claim that any variables are dependent, the dependency degree must greater than this value significantly. The default is 0.05.

Value

This function returns results of dependency inference among variables.

`E0`	An adjacency matrix of undirected graph where there is an edge between any pair of variables if they are dependent.
`E0raw`	A matrix of the degree of dependency of variable pairs.

Examples

```
bSCMDepndentGraphFastFunc(mat)
```

bSCMDepndentGraphFunc — *bSCMDepndentGraphFunc function*

Description

This function infers dependencies for all pairs of variables with bootstrapping.

Usage

```
bSCMDepndentGraphFunc(
  mat,
  nboot = 100,
  alpha = 0.05,
  IndpThs = 0.05,
  pflag = FALSE
)
```

Arguments

`mat`	is a matrix n by d where n is a number of transactions or samples and d is a number of dimensions.
`nboot`	is a number of bootstrap replicates for bootstrapping deployed to infer confidence intervals and distributions for hypothesis tests. The default is 100.
`alpha`	is a significance threshold for hypothesis tests (Mann Whitney) that deploys for testing degrees of dependency, association direction, and causal direction. The default is 0.5.
`IndpThs`	is a threshold for the degree of dependency. In the independence test, to claim that any variables are dependent, the dependency degree must greater than this value significantly. The default is 0.05.
`pflag`	is a flag for printing progress message (TRUE). The default is FALSE (no printing).

Value

This function returns results of dependency inference among variables.

`E0`	An adjacency matrix of undirected graph where there is an edge between any pair of variables if they are dependent.
`E0pval`	A matrix of p-values from independence test of pairs of variables.
`E0mean`	A matrix of means of dependency degrees between variables.
`E0lowbound`	A matrix of lower bounds of dependency-degree confidence intervals between variables.
`depInfo[['i,j']]$bmean`	A mean of dependency degrees between variables i and j.
`depInfo[['i,j']]$confInv`	An `alpha`*100th percentile confidence interval of dependency degrees between variables i and j.
`depInfo[['i,j']]$testRes`	A Mann-Whitney hypothesis test result for an independence test between variables i and j. The null hypothesis is that the distributions of dependency degrees of i,j differ by a location shift of `IndpThs` and the alternative is that distributions of dependency degrees of i,j is shifted greater than `IndpThs`.
`depInfo[['i,j']]$indices`	A pair of indices of i and j in a numeric vector.

Dboot	A list of Ds (aligned list of transactions) that are generated from sampling with replacement on input samples (mat) nboot times.

Examples

```
bSCMDepndentGraphFunc(mat, nboot=50)
```

CausalGraphInferMainFunc
CausalGraphInferMainFunc function

Description

A framework to infer causality on binary data using techniques in frequent pattern mining and estimation statistics. Given a set of individual vectors S=x where x(i) is a realization value of binary variable i, the framework infers empirical causal relations of binary variables i,j from S in a form of causal graph G=(V,E) where V is a set of nodes representing binary variables and there is an edge from i to j in E if the variable i causes j. The framework determines dependency among variables as well as analyzing confounding factors before deciding whether i causes j.

Note that all statistics (e.g. means) and confidence intervals as well as hypothesis testing are inferred by bootstrapping.

Usage

```
CausalGraphInferMainFunc(
  mat,
  alpha = 0.05,
  nboot = 100,
  IndpThs = 0.05,
  CausalThs = 0.1
)
```

Arguments

mat	is a matrix n by d where n is a number of transactions or samples and d is a number of dimensions.
alpha	is a significance threshold for hypothesis tests (Mann Whitney) that deploys for testing degrees of dependency, association direction, and causal direction. The default is 0.5.
nboot	is a number of bootstrap replicates for bootstrapping deployed to infer confidence intervals and distributions for hypothesis tests. The default is 100.
IndpThs	is a threshold for the degree of dependency. In the independence test, to claim that any variables are dependent, the dependency degree must greater than this value significantly. The default is 0.05.
CausalThs	is a threshold for the degree of causal direction In the causal-direction test, to claim that any variables have causal relations, the degree of causal direction must greater than this value significantly. The default is 0.1.

Value

This function returns causal inference results. #TODO: provide list of results.

depRes
: The result of inferring dependencies between all pairs of variables.

ConfoundRes
: The result of filtering associations without true causal directions from any confounding factor.

CausalGRes
: The result of inferring causal directions between all pairs of dependent variables that have no confounding factors.

depRes$E0
: An adjacency matrix of undirected graph where there is an edge between any pair of variables if they are dependent.

depRes$E0pval
: A matrix of p-values from independence test of pairs of variables.

depRes$E0mean
: A matrix of means of dependency degrees between variables.

depRes$E0lowbound
: A matrix of lower bounds of dependency-degree confidence intervals between variables.

depRes$depInfo[['i,j']]$bmean
: A mean of dependency degrees between variables i and j.

depRes$depInfo[['i,j']]$confInv
: An alpha*100th percentile confidence interval of dependency degrees between variables i and j.

depRes$depInfo[['i,j']]$testRes
: A Mann-Whitney hypothesis test result for an independence test between variables i and j. The null hypothesis is that the distributions of dependency degrees of i,j differ by a location shift of IndpThs and the alternative is that distributions of dependency degrees of i,j is shifted greater than IndpThs.

depRes$depInfo[['i,j']]$indices
: A pair of indices of i and j in a numeric vector.

depRes$Dboot
: A list of Ds (aligned list of transactions) that are generated from sampling with replacement on input samples (mat) nboot times.

ConfoundRes$E1
: An adjacency matrix of undirected graph after filtering associations without true causal directions from any confounding factor.

ConfoundRes$E2
: A matrix of associations that have confounding factors where E2[i,j]=0 if no confounding factor and E2[i,j]=k if k is a confounding factor of i and j.

CausalGRes$Ehat
: An adjacency matrix of directed causal graph where CausalGRes$Ehat[i,j]=1 implies i causes j.

CausalGRes$EValHat
: An adjacency matrix of weighted directed causal graph where edge weights are estimated means of probabilities of effect being 1 given cause being either 1 for positive association or 0 for negative association using CondProb() and bootstrapping to estimate

CausalGRes$causalInfo[['i,j']]$CDirConfValInv
: An alpha*100th percentile confidence interval of estimated conditional probability of effect j being 1/0 given cause i's value being either the same (positive association) or opposite (negative association).

`CausalGRes$causalInfo[['i,j']]$CDirConfInv`
: An `alpha`*100th percentile confidence interval of estimated causal direction degree of i cause j.

`CausalGRes$causalInfo[['i,j']]$CDirmean`
: A mean-estimated-causal-direction degree of i cause j.

`CausalGRes$causalInfo[['i,j']]$testRes2`
: A Mann-Whitney hypothesis test result for existence of causal direction. The null hypothesis is that the distributions of causal-direction degrees of i,j differ by a location shift of `CausalThs` and the alternative is that distributions of causal-direction degrees of i,j is shifted greater than `CausalThs`.

`CausalGRes$causalInfo[['i,j']]$testRes1`
: A Mann-Whitney hypothesis test result for existence of association by odd differences from `oddDiffFunc()`. The null hypothesis is that the distributions of absolute odd difference of i,j differ by a location shift of `IndpThs` and the alternative is that distributions of absolute odd difference of i,j is shifted greater than `IndpThs`.

`CausalGRes$causalInfo[['i,j']]$sign`
: A direction of i,j association: 1 for positive, 0 for negative, and -1 for no association.

`CausalGRes$causalInfo[['i,j']]$SignConfInv`
: An `alpha`*100th percentile confidence interval of i,j odd difference from bootstrapping.

`CausalGRes$causalInfo[['i,j']]$Signmean`
: A mean of i,j odd difference from bootstrapping.

Examples

```
resC<-CausalGraphInferMainFunc(mat = mat, nboot =50)
```

`comparePredAdjMatrix2TrueAdjMat` *comparePredAdjMatrix2TrueAdjMat*

Description

comparePredAdjMatrix2TrueAdjMat is a support function that can compare two adjacency matrices: ground-truth and inferred matrices.

Usage

```
comparePredAdjMatrix2TrueAdjMat(trueAdjMat, adjMat)
```

Arguments

`trueAdjMat`	a ground-truth matrix.
`adjMat`	an inferred matrix.

Value

This function returns a list of precision `prec`, recall `rec`, and F1 score `F1` of inferred vs. groundtruth matrices.

Examples

```
# Generate simulation data
G<-matrix(FALSE,10,10) # groundtruth
G[1,c(4,7,8,10)]<-TRUE
G[2,c(5,7,9,10)]<-TRUE
G[3,c(6,8,9,10)]<-TRUE
comparePredAdjMatrix2TrueAdjMat(trueAdjMat=G,adjMat=G)
```

`CondProb` *CondProb function*

Description

This function computes a confidence value of `y` given `c` or `conf(y|z)` from an aligned list `D`. For any`y[i],z[j]`, their values are -1 by default. The function computes the numbers of transactions that satisfy the following conditions.

1. All transactions must have values at any k position equal to `z[k]` for any `z[k]` that is not -1. Let `count` be the number of these transactions in `D`.
2. All transactions must have values at any k position equal to either `z[k]` or `y[k]` that is not -1. Let `countTotal` be the number of these transactions in `D`.

Usage

```
CondProb(D, y, z)
```

Arguments

`D`	is an aligned list of transactions that was converted from any matrix n by d `mat` using `D<-VecAlignment(mat)` where n is a number of transactions or samples and d is a number of dimensions for each sample.
`y`	is a d-dimensional vector.
`z`	is a d-dimensional vector.

Value

This function returns the ratio `condP=count/countTotal`, which is the confidence of `y` given `z`.

`condP`	The confidence of `y` given `z` in `D`.
`nD`	The subset of `D` such that all transactions have values at any position similar to `z[k]` when `z[k]` is not -1.

count — A number of transactions that have values at any position similar to either z[k] or y[k] that is not -1.

countTotal — A number of transactions in nD

Examples

```
d=10 # dimensions of example vectors
z<-numeric(d)-1
y<-numeric(d)-1
y[1]<-c(1)
z[c(2,3)]<-c(1,1)
CondProb(BiCausality::D,y=y,z=z)$condP # conf(inx1 is 1 |inx 2,3 are 1 ) y|z
```

---

confNetFunc *confNetFunc function*

Description

This function Computes a confidence network in data mining. Given a set of n transactions or samples in mat s.t. each transaction has d binary items. The conf(mat[,j]=1|mat[,i]=1) is a ratio of a number of samples in jth and ith dimensions that have values equal to one divided by a number of samples in the ith dimension that has a value equal to one. The confNetFunc computes the network where the nodes are dimensions and the edge weights are conf(mat[,j]=1|mat[,i]=1) for any directed edge from i to j.

Usage

```
confNetFunc(mat, ths = 0.1)
```

Arguments

mat — is a matrix n by d where n is a number of transactions or samples and d is a number of dimensions.

ths — is a threshold parameter for cutting of the edge weights. There exists the directed edge from i to j if its edge weight if above or equal ths.

Value

This function returns a binary adjacency matrix confNet and the weighted adjacency matrix confValMat.

confNet — A binary adjacency matrix that has confNet[i,j]=1 if confValMat[i,j]>=ths. Otherwise, it is zero.

confValMat — A weighted adjacency matrix where confValMat[i,j] is conf(mat[,j]=1|mat[,i]=1).

Examples

```
res<-confNetFunc(mat)
```

D — *An example of aligned list of transactions*

Description

A dataset containing simulated data that is used for examples in the package.

The `D` is an aligned list of transactions that was converted by using `D<-VecAlignment(mat)`.

Usage

```
D
```

Format

An aligned list of a matrix with 200 samples and 10 dimensions generated from Bernoulli distribution.

D It is an aligned list of transactions that was converted from `mat`.

getReachableNodes — *getReachableNodes function*

Description

getReachableNodes is a support function for inferring reachable nodes that have some directed path to a node `targetNode`. This function uses Breadth-first search (BFS) algorithm.

Usage

```
getReachableNodes(adjMat, targetNode)
```

Arguments

`adjMat`	is an adjacency matrix of a directed graph of which its elements are binary: zero for no edge, and one for having an edge.
`targetNode`	is a node in a graph that we want to find a set of nodes that can reach this target node via some paths.

Value

This function returns a set of node IDs that have some directed path to a node `targetNode`.

Examples

```
# Given an example of adjacency matrix
A<-matrix(FALSE,5,5)
A[2,1]<-TRUE
A[c(3,4),2]<-TRUE
A[5,3]<-TRUE
# Get a set of reachable nodes of targetNode.

followers<-getReachableNodes(adjMat=A,targetNode=1)
```

---

getTransitiveClosureMat *getTransitiveClosureMat function*

Description

getTransitiveClosureMat is a support function for inferring a transitive-closure adjacency matrix.

Usage

```
getTransitiveClosureMat(adjMat)
```

Arguments

adjMat	is an adjacency matrix of a directed graph of which its elements are binary: zero for no edge, and one for having an edge.

Value

This function returns a transitive-closure adjacency matrix.

Examples

```
# Given an example of adjacency matrix
A<-matrix(FALSE,5,5)
A[2,1]<-TRUE
A[c(3,4),2]<-TRUE
A[5,3]<-TRUE
# Get a set of reachable nodes of targetNode.

trsClosureMat<-getTransitiveClosureMat(adjMat=A)
```

indpFunc *indpFunc function*

Description

This function computes the degree of dependency between variables. Let i and j be variables, if they are independent, then |p(i,j) -p(i)*p(j)| should be zero. Given the samples in the n by d matrix `mat` where n is a number of samples and d is a number of dimensions, an aligned list of transactions `D` is computed by `D<-VecAlignment(mat)`.

Usage

```
indpFunc(D, i, j, z = c())
```

Arguments

`D`	is an aligned list of transactions that was converted from `mat`.
`i`	is an ith dimension in `mat`.
`j`	is an jth dimension in `mat`.
`z`	is a conditioning d-dimensional vector on `D`. Given k non-negative-bit positions of `z`, all k bit positions of samples in the subset of `D` must have similar values with these bits.

Value

This function returns the degree of dependency between variables: zero implies both variables are independent, and non-zero value implies the degree of dependency (higher implies more dependent degree).

Examples

```
indpFunc(D,i=1,j=2)
```

mat *A simulation dataset*

Description

A dataset containing simulated data that is used for examples in the package. The matrix `mat` is generated by the following code.

`seedN<-2022`

`n<-200` # 200 individuals

`d<-10` # 10 variables

`mat<-matrix(nrow=n,ncol=d)` # the input of framework

#Simulate binary data from Bernoulli distribution distribution where the probability of value being 1 is 0.5.

`for(i in seq(n)) { set.seed(seedN+i)`

`mat[i,] <- rbinom(n=d, size=1, prob=0.5) }`

`mat[,1]<-mat[,2] | mat[,3]` # 1 causes by 2 and 3

`mat[,4] <-mat[,2] | mat[,5]` # 4 causes by 2 and 5

`mat[,6] <- mat[,1] | mat[,4]` # 6 causes by 1 and 4

Usage

```
mat
```

Format

A matrix with 200 samples and 10 dimensions generated from Bernoulli distribution.

mat It is a 200 by 10 matrix where n is a number of transactions or samples and d is a number of dimensions. ...

`num2Bits` *num2Bits function*

Description

Given a natural number and number of bits, the function provides an n-dimensional vector of bits that represents `num`. The ith bits of binary vector represents the ith bit of `num`. For example, if `vec<-num2Bits(num=2,n=4)`, the first bit `vec[1]` is 0 and the second bit `vec[2]` is 1.

Usage

```
num2Bits(num, n = 32)
```

Arguments

`num`	is a natural number.
`n`	is a number of bits representing `num`.

Value

This function returns an n-dimensional vector of bits that represents `num`.

Examples

```
num2Bits(num=10,n=4)
```

---

oddDiffFunc *oddDiffFunc function*

Description

Given the samples in the n by d matrix `mat` where n is a number of samples and d is a number of dimensions. This function computes an odd difference value of variables of ith and jth dimensions from a given an aligned list of transactions `D` (compute by `D<-VecAlignment(mat)`).

Usage

```
oddDiffFunc(D, i, j, z = c())
```

Arguments

`D`	is an aligned list of transactions that was converted from `mat`.
`i`	is an ith dimension in `mat` for computing the odd difference with.
`j`	is an jth dimension in `mat` for computing compute the odd difference with.
`z`	is a conditioning d-dimensional vector on `D`. Given k non-negative-bit positions of `z`, all k bit positions of samples in the subset of `D` must have similar values with these bits.

Value

This function returns an odd difference value of variables of ith and jth dimensions from `D`.

Examples

```
oddDiffFunc(D,i=1,j=2)
```

`oddRatioFunc` *oddRatioFunc function*

Description

Given the samples in the n by d matrix `mat` where n is a number of samples and d is a number of dimensions. This function computes an odd ratio value of variables of ith and jth dimensions from a given an aligned list of transactions `D` (compute by `D<-VecAlignment(mat)`).

Usage

```
oddRatioFunc(D, i, j, z = c(), slack = 0.001)
```

Arguments

`D`	is an aligned list of transactions that was converted from `mat`.
`i`	is an ith dimension in `mat` for computing the odd ratio with.
`j`	is an jth dimension in `mat` for computing compute the odd ratio with.
`z`	is a conditioning d-dimensional vector on `D`. Given k non-negative-bit positions of `z`, all k bit positions of samples in the subset of `D` must have similar values with these bits.
`slack`	is a parameter to prevent the issue of division by zero.

Value

This function returns an odd ratio value of variables of ith and jth dimensions from `D`.

Examples

```
oddRatioFunc(D,i=1,j=2)
```

`resC` *An example of causal inference result*

Description

A dataset containing a result of causal inference from simulated data that is used for examples in the package.

Usage

```
resC
```

Format

A result of causal inference using `mat` as an input.

resC It is a result of causal inference using simData$mat as an input by running `resC<-BiCausality::CausalGraphInferM` `= mat,CausalThs=0.1, nboot =50, IndpThs=0.05)`..

---

supp *supp function*

Description

This function computes a support value from a matrix `X` given a `values`.

Usage

```
supp(X, values)
```

Arguments

`X`	is a matrix n by d where n is a number of transactions or samples and d is a number of dimensions for each sample.
`values`	is a d-dimensional vector we use to count how many of it within `X`.

Value

This function returns the support of `values` in `X` by counting the ratio of how many samples in `X` are similar to `values`

Examples

```
x <- rbinom(n=100, size=1, prob=0.5)
ny<-rbinom(n=100, size=1, prob=0.25)
y <- x |  ny
supp(X=cbind(x,y),values=c(1,1) )
```

VecAlignment *VecAlignment function*

Description

This function rearranges the samples in the `mat` into an aligned list of transactions, which is mainly used by other functions in the package. Suppose `mat[i,]` is a binary vector we are interested, we use `A<-bin2dec(mat[i,])` to store the decimal value of `mat[i,]` in `A`. Then, we call `D[[A]]$count` to get number of samples in `mat` that are similar to `mat[i,]` and the `D[[A]]$name` is `mat[i,]`.

Usage

```
VecAlignment(mat)
```

Arguments

`mat`	is a matrix n by d where n is a number of transactions or samples and d is a number of dimensions.

Value

This function returns an aligned list of transactions `D`, is an aligned list of transactions that was converted from any matrix n by d `mat`.

Examples

```
VecAlignment(mat=mat)
```

Index