R documentation

of all in '.'

December 23, 2014

R topics documented:

`BN_Data_Generator-package`

What the package does (short line) ~~ package title ~~

Description

More about what it does (maybe more than one line) ~~ A concise (1-5 lines) description of the package ~~

Details

Package:	BN_Data_Generator
Type:	Package
Version:	1.0
Date:	2014-12-23
License:	What license is it under?

~~ An overview of how to use the package, including the most important ~~ ~~ functions ~~

Author(s)

Who wrote it

Maintainer: Who to complain to <yourfault@somewhere.net> ~~ The author and/or maintainer of the package ~~

References

~~ Literature or other references for background information ~~

See Also

~~ Optional links to other man pages, e.g. ~~ ~~ <pkg> ~~

Examples

```
~~ simple examples of the most important functions ~~
```

---

big_letters

Usage

```
big_letters(len)
```

Arguments

```
len
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (len)
{
    letters_list = list()
```

```
    letters_list[[1]] = letters
    while (TRUE) {
        num_of_letters = 0
        len_letters_list = length(letters_list)
        for (i in 1:len_letters_list) {
            num_of_letters = num_of_letters + length(letters_list[[i]])
        }
        if (num_of_letters < len) {
            merge_mat = merge(letters, letters_list[[len_letters_list]])
            letters_list[[len_letters_list + 1]] = sort(paste(merge_mat[,
                1], merge_mat[, 2], sep = ""))
        }
        else {
            break
        }
    }
    result = NULL
    for (i in 1:length(letters_list)) {
        result = c(result, letters_list[[i]])
    }
    result = result
    return(result[1:len])
}
```

BN_Data_Generator

Usage

```
BN_Data_Generator(arcs_mat, input_Probs, n, node_names = NULL, cardinalities = NULL)
```

Arguments

arcs_mat
input_Probs
n
node_names
cardinalities

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (arcs_mat, input_Probs, n, node_names = NULL, cardinalities = NULL)
{
    if (n <= 0) {
```

```
    stop("Sample size 'n' must be greater than 0.")
}
if (n < 10000) {
    temp_n = 1000
}
else {
    temp_n = n
}
num_of_nodes = dim(arcs_mat)[1]
result_mat = matrix(0, temp_n, num_of_nodes)
dimnames(result_mat)[[2]] = node_names
checker = check_cardinalities(arcs_mat = arcs_mat, node_names = node_names,
    cardinalities = cardinalities)
cardinalities = checker$cardinalities
node_names = checker$node_names
list_parent_nodes = checker$list_parent_nodes
num_of_probs = checker$num_of_probs
num_of_parent_nodes = checker$num_of_parent_nodes
num_of_root_nodes = checker$num_of_root_nodes
input_prob_len = length(input_Probs)
for (i in 1:input_prob_len) {
    if (as.numeric(length(input_Probs[[i]])) != as.numeric(num_of_probs[i])) {
        stop("Input Probs != num_of_probs!")
    }
}
for (i in 1:num_of_root_nodes) {
    p = input_Probs[[i]]
    mat_values = merge("Value", c(1:cardinalities[i]))
    mat_values = paste(mat_values[, 1], mat_values[, 2],
        sep = "")
    result_mat[, i] = sample(mat_values, temp_n, prob = c(p,
        1 - sum(p)), rep = T)
}
init = num_of_root_nodes + 1
mat = NULL
for (i in init:num_of_nodes) {
    p = input_Probs[[i]]
    temp_list_of_pn = as.numeric(list_parent_nodes[[i]])
    num_of_c_cases = prod(cardinalities[temp_list_of_pn])
    temp_cases = list()
    cases = NULL
    for (j in 1:length(temp_list_of_pn)) {
        temp_cases[[j]] = toss_value(1, cardinalities[temp_list_of_pn[j]])
        if (is.null(cases)) {
            cases = temp_cases[[j]]
            names(cases) = 1
        }
        else {
            cases = merge(cases, temp_cases[[j]])
            names(cases) = c(1:dim(cases)[2])
        }
    }
    cases = as.matrix(cases)
```

```
        mat_values = merge("Value", c(1:cardinalities[i]))
        mat_values = sort(paste(mat_values[, 1], mat_values[,
            2], sep = ""))
        stack = 1
        for (j in 1:dim(cases)[1]) {
            mat = t(t(as.matrix(result_mat[, temp_list_of_pn])) ==
                cases[j, ])
            mat = (apply(mat, 1, sum) == dim(mat)[2])
            if (cardinalities[i] == 2) {
                temp_p = p[j]
            }
            else {
                temp_p = p[stack:(stack + cardinalities[i] -
                  2)]
            }
            len = length(which(mat))
            result_mat[which(mat), i] = sample(mat_values, len,
                prob = c(temp_p, 1 - sum(temp_p)), rep = T)
            stack = stack + (cardinalities[i] - 1)
        }
    }
    if (n < 1000) {
        result_mat = result_mat[sample(c(1:1000), size = n),
            ]
    }
    res = list(data = data.frame(result_mat), node_names = node_names,
        num_of_nodes = num_of_nodes, num_of_parent_nodes = num_of_parent_nodes,
        list_parent_nodes = list_parent_nodes)
    return(res)
}
```

---

check_cardinalities

Usage

```
check_cardinalities(arcs_mat, node_names = NULL, cardinalities = NULL)
```

Arguments

arcs_mat	
node_names	
cardinalities	

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.
```

```
## The function is currently defined as
function (arcs_mat, node_names = NULL, cardinalities = NULL)
{
    check_dag_arcs = as.matrix(arcs_mat)
    if (is_DAG(check_dag_arcs) == FALSE) {
        stop("arcs_mat must a DAG")
    }
    num_of_nodes = dim(arcs_mat)[1]
    if (is.null(node_names)) {
        node_names = big_letters(num_of_nodes)
    }
    if (is.null(cardinalities)) {
        cardinalities = rep(2, num_of_nodes)
    }
    else if (sum(cardinalities < 2) > 0) {
        stop("All cardinality must be at least 2.")
    }
    else if (num_of_nodes != length(cardinalities)) {
        stop("Wrong length of cardinalities")
    }
    num_of_parent_nodes = apply(arcs_mat, 2, sum)
    list_parent_nodes = list()
    for (m in 1:num_of_nodes) {
        if (length(which(arcs_mat[, m] == 1)) == 0) {
            list_parent_nodes[[m]] = NULL
        }
        else {
            list_parent_nodes[[m]] = which(arcs_mat[, m] == 1)
        }
    }
    num_of_root_nodes = sum(num_of_parent_nodes == 0)
    num_of_probs = NULL
    for (k in 1:num_of_nodes) {
      num_of_probs[k] = (cardinalities[k] - 1) * prod(cardinalities[list_parent_nodes[[k]]])
    }
    text_of_probs = list()
    for (i in 1:length(num_of_parent_nodes)) {
        temp_text = NULL
        present_cardinality = as.matrix(toss_value(1, cardinalities[i]))
        if (num_of_parent_nodes[i] == 0) {
            for (j in 1:(cardinalities[i] - 1)) {
                temp_text = c(temp_text, paste("P(", node_names[i],
                  " = ", present_cardinality[j, 1], ")", sep = ""))
            }
        }
        else {
            temp_list_of_pn = as.numeric(list_parent_nodes[[i]])
            for (j in 1:(cardinalities[i] - 1)) {
                temp_cases = list()
                cases = NULL
                for (k in 1:length(temp_list_of_pn)) {
                  temp_cases[[k]] = toss_value(1, cardinalities[temp_list_of_pn[k]])
```

```
                    if (is.null(cases)) {
                      cases = temp_cases[[k]]
                      names(cases) = 1
                    }
                    else {
                      cases = merge(cases, temp_cases[[k]])
                      names(cases) = c(1:dim(cases)[2])
                    }
                  }
                  cases = as.matrix(cases)
                  for (k in 1:dim(cases)[1]) {
                    temp_text_conditional = NULL
                    for (m in 1:dim(cases)[2]) {
                      case_value = paste(node_names[temp_list_of_pn[m]],
                        " = ", cases[k, m], sep = "")
                      if (m == 1) {
                        temp_text_conditional = case_value
                      }
                      else {
                        temp_text_conditional = paste(temp_text_conditional,
                          paste(", ", case_value), sep = "")
                      }
                    }
                    temp_text = c(temp_text, paste("P(", node_names[i],
                      " = ", present_cardinality[j, 1], "|", temp_text_conditional,
                      ")", sep = ""))
                  }
              }
          }
          text_of_probs[[i]] = temp_text
      }
      res = list(cardinalities = cardinalities, node_names = node_names,
          num_of_root_nodes = num_of_root_nodes, num_of_probs = num_of_probs,
         num_of_parent_nodes = num_of_parent_nodes, list_parent_nodes = list_parent_nodes,
          list_of_probs = text_of_probs)
      return(res)
    }
```

C_M_WO_WC

Usage

```
C_M_WO_WC(target_arcs_mat, learnt_arcs_mat)
```

Arguments

target_arcs_mat	
learnt_arcs_mat	

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (target_arcs_mat, learnt_arcs_mat)
{
    nodes = dim(target_arcs_mat)[2]
    C = 0
    M = 0
    WO = 0
    WC = 0
    for (i in 1:nodes) {
        C = C + abs(sum((target_arcs_mat[, i] == 1) & (learnt_arcs_mat[,
            i] == 1) & (target_arcs_mat[, i] == learnt_arcs_mat[,
            i])))
        M = M + abs(sum((target_arcs_mat[, i] == 1) & (learnt_arcs_mat[,
            i] == 0) & (target_arcs_mat[, i] != learnt_arcs_mat[,
            i])) - sum((target_arcs_mat[, i] == 1) & (learnt_arcs_mat[i,
            ] == 1) & (target_arcs_mat[, i] == learnt_arcs_mat[i,
            ])))
        WO = WO + abs(sum((target_arcs_mat[, i] == 1) & (learnt_arcs_mat[i,
            ] == 1) & (target_arcs_mat[, i] == learnt_arcs_mat[i,
            ])))
        WC = WC + abs(sum((target_arcs_mat[, i] == 0) & (learnt_arcs_mat[,
            i] == 1) & (target_arcs_mat[, i] != learnt_arcs_mat[,
            i])) - sum((target_arcs_mat[i, ] == 0) & (learnt_arcs_mat[i,
            ] == 1) & (target_arcs_mat[i, ] != learnt_arcs_mat[i,
            ])))
    }
    result = t(as.matrix(c(C, M, WO, WC)))
    dimnames(result)[[2]] = c("C", "M", "WO", "WC")
    return(result)
  }
```

---

fromto_to_mat

Usage

```
fromto_to_mat(fromto, node_names)
```

Arguments

fromto	
node_names	

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (fromto, node_names)
{
    if (dim(fromto)[1] == 0) {
        stop("It has not any arc")
    }
    num_of_nodes = length(node_names)
    arcs_mat = matrix(0, num_of_nodes, num_of_nodes)
    arcs_order_mat = cbind(node_names, c(1:length(node_names)))
    temp_arcs = cbind(match(fromto[, 1], arcs_order_mat), match(fromto[,
        2], arcs_order_mat))
    if (length(temp_arcs) > 0) {
        for (i in 1:dim(temp_arcs)[1]) {
            from = as.numeric(temp_arcs[i, 1])
            to = as.numeric(temp_arcs[i, 2])
            arcs_mat[from, to] = arcs_mat[from, to] + 1
        }
    }
    dimnames(arcs_mat)[[1]] = node_names
    dimnames(arcs_mat)[[2]] = node_names
    return(arcs_mat)
  }
```

---

gen_asia

Usage

```
gen_asia()
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function ()
{
    arcs_mat = rbind(c(0, 0, 1, 0, 0, 0, 0, 0), c(0, 0, 0, 1,
        1, 0, 0, 0), c(0, 0, 0, 0, 0, 1, 0, 0), c(0, 0, 0, 0,
        0, 1, 0, 0), c(0, 0, 0, 0, 0, 0, 0, 1), c(0, 0, 0, 0,
        0, 0, 1, 1), c(0, 0, 0, 0, 0, 0, 0, 0), c(0, 0, 0, 0,
        0, 0, 0, 0))
    node_names = c("A", "S", "T", "L", "B", "E", "X", "D")
```

```
  dimnames(arcs_mat)[[1]] = node_names
  dimnames(arcs_mat)[[2]] = node_names
  input_Probs = list(c(0.01), c(0.5), c(0.05, 0.01), c(0.1,
      0.01), c(0.6, 0.3), c(1, 1, 1, 0), c(0.98, 0.05), c(0.9,
      0.7, 0.8, 0.1))
  num_of_nodes = length(node_names)
  cardinalities = rep(2, num_of_nodes)
  result = list(arcs_mat = arcs_mat, Probs = input_Probs, node_names = node_names,
      cardinalities = cardinalities, num_of_nodes = num_of_nodes)
  return(result)
}
```

---

is_acyclic

Usage

```
is_acyclic(arcs_mat)
```

Arguments

```
arcs_mat
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (arcs_mat)
{
    transClos = function(arcs_mat) {
        if (nrow(arcs_mat) == 1)
            return(arcs_mat)
        A = arcs_mat
        diag(A) = 1
        repeat {
            B = sign(A %*% A)
            if (all(B == A))
                break
            else A = B
        }
        diag(A) = 0
        A
    }
    B = transClos(arcs_mat)
    l = B[lower.tri(B)]
    u = t(B)[lower.tri(t(B))]
    com = (l & u)
```

```
    return(all(!com))
  }
```

---

is_DAG

Usage

```
is_DAG(arcs_mat)
```

Arguments

```
arcs_mat
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (arcs_mat)
{
    unmakeMG = function(arcs_mat) {
        d = nrow(arcs_mat)
        ug = dg = bg = arcs_mat
        M = expand.grid(dg = 0:1, ug = 0:1, bg = 0:1)
        i = strtoi(as.character(arcs_mat), 2)
        GG = M[i + 1, ]
        ug[, ] = GG[, 2]
        dg[, ] = GG[, 1]
        bg[, ] = GG[, 3]
        if (any(ug != t(ug)))
            stop("Undirected edges are wrongly coded.")
        if (any(bg != t(bg)))
            stop("Undirected edges are wrongly coded.")
        return(list(dg = dg, ug = ug, bg = bg))
    }
    comp = unmakeMG(arcs_mat)
    ug = comp$ug
    dag = comp$dg
    bg = comp$bg
    out = TRUE
    if (any(arcs_mat > 100)) {
        warning("There are double edges.")
        out = FALSE
    }
    if (!is_acyclic(dag)) {
        warning("Not acyclic.")
        out = FALSE
```

```
  }
  return(out)
}
```

---

make_topology

Usage

```
make_topology(nodes, topology = "Collapse", input_Probs = NULL, node_names = NULL, cardinalities = NUL
```

Arguments

nodes
topology
input_Probs
node_names
cardinalities

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (nodes, topology = "Collapse", input_Probs = NULL, node_names = NULL,
    cardinalities = NULL)
{
    NeedMoreNodes = function(num_of_nodes) {
        if (nodes < num_of_nodes)
            stop("Need More Nodes!")
    }
    switch(topology, Collapse = {
        NeedMoreNodes(3)
    }, Line = {
        NeedMoreNodes(3)
    }, Star = {
        NeedMoreNodes(3)
    }, PseudoLoop = {
        NeedMoreNodes(3)
    }, Diamond = {
        NeedMoreNodes(4)
    }, Rhombus = {
        NeedMoreNodes(4)
    }, )
    arcs_mat = matrix(0, nodes, nodes)
    switch(topology, Collapse = {
```

```
    arcs_mat[, nodes] = 1
    arcs_mat[nodes, nodes] = 0
}, Line = {
    for (i in 1:(nodes - 1)) {
        arcs_mat[i, (i + 1)] = 1
    }
}, Star = {
    arcs_mat[1, ] = 1
    arcs_mat[1, 1] = 0
}, PseudoLoop = {
    arcs_mat[1, nodes] = 1
    for (i in 1:(nodes - 1)) {
        arcs_mat[i, (i + 1)] = 1
    }
}, Diamond = {
    arcs_mat[1, ] = 1
    arcs_mat[1, 1] = 0
    arcs_mat[, nodes] = 1
    arcs_mat[1, nodes] = 0
    arcs_mat[nodes, nodes] = 0
}, Rhombus = {
    arcs_mat[1, ] = 1
    arcs_mat[2, ] = 1
    arcs_mat[(1:2), (1:2)] = 0
    arcs_mat[nodes, nodes] = 0
}, )
checker = check_cardinalities(arcs_mat = arcs_mat, node_names = node_names,
    cardinalities = cardinalities)
cardinalities = checker$cardinalities
num_of_probs = checker$num_of_probs
node_names = checker$node_names
if (is.null(input_Probs) & is.null(cardinalities)) {
    input_Probs = list()
    switch(topology, Collapse = {
        for (i in 1:(nodes - 1)) {
            input_Probs[[i]] = runif(1)
        }
        input_Probs[[nodes]] = runif(2^(nodes - 1))
    }, Line = {
        input_Probs[[1]] = runif(1)
        for (i in 2:nodes) {
            input_Probs[[i]] = runif(2)
        }
    }, Star = {
        input_Probs[[1]] = runif(1)
        for (i in 2:nodes) {
            input_Probs[[i]] = runif(2)
        }
    }, PseudoLoop = {
        input_Probs[[1]] = runif(1)
        for (i in 2:(nodes - 1)) {
            input_Probs[[i]] = runif(2)
        }
```

```
            input_Probs[[nodes]] = runif(4)
        }, Diamond = {
            input_Probs[[1]] = runif(1)
            for (i in 2:(nodes - 1)) {
                input_Probs[[i]] = runif(2)
            }
            input_Probs[[nodes]] = runif(2^(nodes - 2))
        }, Rhombus = {
            input_Probs[[1]] = runif(1)
            input_Probs[[2]] = runif(1)
            for (i in 3:nodes) {
                input_Probs[[i]] = runif(2^2)
            }
        }, )
    }
    else if (is.null(input_Probs)) {
        input_Probs = list()
        for (i in 1:length(num_of_probs)) {
            input_Probs[[i]] = runif(num_of_probs[i])
        }
    }
    result = list(arcs_mat = arcs_mat, Probs = input_Probs, node_names = node_names,
        cardinalities = cardinalities, num_of_nodes = nodes)
    return(result)
}
```

---

mat_to_fromto

Usage

```
mat_to_fromto(arcs_mat)
```

Arguments

```
arcs_mat
```

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (arcs_mat)
{
    check_dag_arcs = as.matrix(arcs_mat)
    if (is_DAG(check_dag_arcs) == FALSE) {
        stop("arcs_mat must a DAG")
    }
```

```
    node_names = dimnames(arcs_mat)[[2]]
    num_of_nodes = length(node_names)
    result_mat = NULL
    for (i in 1:num_of_nodes) {
        where = which(arcs_mat[i, ] == 1)
        len = length(where)
        if (len > 0) {
            for (j in 1:len) {
                temp = c(node_names[i], node_names[where[j]])
                result_mat = rbind(result_mat, temp)
            }
        }
    }
    dimnames(result_mat)[[1]] = NULL
    dimnames(result_mat)[[2]] = c("from", "to")
    return(result_mat)
  }
```

---

real_alarm

Usage

```
real_alarm(n, rep = T)
```

Arguments

n

rep

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (n, rep = T)
{
    packages = c("bnlearn")
    if (length(setdiff(packages, rownames(installed.packages()))) >
        0) {
        install.packages(setdiff(packages, rownames(installed.packages())))
    }
    data(alarm, package = "bnlearn")
    data = alarm[sample(c(1:20000), n, rep = rep), ]
    res = empty.graph(names(alarm))
    modelstring(res) = paste("[HIST|LVF][CVP|LVV][PCWP|LVV][HYP][LVV|HYP:LVF]",
        "[LVF][STKV|HYP:LVF][ERLO][HRBP|ERLO:HR][HREK|ERCA:HR][ERCA]",
        "[HRSA|ERCA:HR][ANES][APL][TPR|APL][ECO2|ACO2:VLNG][KINK]",
```

```
        "[MINV|INT:VLNG][FIO2][PVS|FIO2:VALV][SAO2|PVS:SHNT][PAP|PMB][PMB]",
        "[SHNT|INT:PMB][INT][PRSS|INT:KINK:VTUB][DISC][MVS][VMCH|MVS]",
        "[VTUB|DISC:VMCH][VLNG|INT:KINK:VTUB][VALV|INT:VLNG][ACO2|VALV]",
        "[CCHL|ACO2:ANES:SAO2:TPR][HR|CCHL][CO|HR:STKV][BP|CO:TPR]",
        sep = "")
    arcs_mat = fromto_to_mat(temp$res$arcs, dimnames(temp$data)[[2]])
    result = list(arcs_mat = arcs_mat, node_names = dimnames(data)[[2]],
        data = data, res = res)
    return(result)
  }
```

---

real_asia

Usage

```
real_asia(n, rep = T)
```

Arguments

n

rep

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (n, rep = T)
{
    packages = c("bnlearn")
    if (length(setdiff(packages, rownames(installed.packages()))) >
        0) {
        install.packages(setdiff(packages, rownames(installed.packages())))
    }
    data(asia, package = "bnlearn")
    data = asia[sample(c(1:5000), n, rep = rep), ]
    res = empty.graph(names(asia))
    modelstring(res) = "[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]"
    arcs_mat = fromto_to_mat(temp$res$arcs, dimnames(temp$data)[[2]])
    result = list(arcs_mat = arcs_mat, node_names = dimnames(data)[[2]],
        data = data, res = res)
    return(result)
  }
```

real_hailfinder

Usage

```
real_hailfinder(n, rep = T)
```

Arguments

n

rep

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (n, rep = T)
{
    packages = c("bnlearn")
    if (length(setdiff(packages, rownames(installed.packages()))) >
        0) {
        install.packages(setdiff(packages, rownames(installed.packages())))
    }
    data(hailfinder, package = "bnlearn")
    data = hailfinder[sample(c(1:20000), n, rep = rep), ]
    res = empty.graph(names(hailfinder))
   modelstring(res) = paste("[N07muVerMo][SubjVertMo][QGVertMotion][SatContMoist][RaoContMoist]",
      "[VISCloudCov][IRCloudCover][AMInstabMt][WndHodograph][MorningBound][LoLevMoistAd][Date]",
       "[MorningCIN][LIfr12ZDENSd][AMDewptCalPl][LatestCIN][LLIW]",
      "[CombVerMo|N07muVerMo:SubjVertMo:QGVertMotion][CombMoisture|SatContMoist:RaoContMoist]",
      "[CombClouds|VISCloudCov:IRCloudCover][Scenario|Date][CurPropConv|LatestCIN:LLIW]",
      "[AreaMesoALS|CombVerMo][ScenRelAMCIN|Scenario][ScenRelAMIns|Scenario][ScenRel34|Scenario]",
      "[ScnRelPlFcst|Scenario][Dewpoints|Scenario][LowLLapse|Scenario][MeanRH|Scenario]",
      "[MidLLapse|Scenario][MvmtFeatures|Scenario][RHRatio|Scenario][SfcWndShfDis|Scenario]",
      "[SynForcng|Scenario][TempDis|Scenario][WindAloft|Scenario][WindFieldMt|Scenario]",
       "[WindFieldPln|Scenario][AreaMoDryAir|AreaMesoALS:CombMoisture]",
      "[AMCINInScen|ScenRelAMCIN:MorningCIN][AMInsWliScen|ScenRelAMIns:LIfr12ZDENSd:AMDewptCalPl]",
      "[CldShadeOth|AreaMesoALS:AreaMoDryAir:CombClouds][InsInMt|CldShadeOth:AMInstabMt]",
      "[OutflowFrMt|InsInMt:WndHodograph][CldShadeConv|InsInMt:WndHodograph][MountainFcst|InsInMt]",
      "[Boundaries|WndHodograph:OutflowFrMt:MorningBound][N34StarFcst|ScenRel34:PlainsFcst]",
      "[CompPlFcst|AreaMesoALS:CldShadeOth:Boundaries:CldShadeConv][CapChange|CompPlFcst]",
       "[InsChange|CompPlFcst:LoLevMoistAd][CapInScen|CapChange:AMCINInScen]",
       "[InsSclInScen|InsChange:AMInsWliScen][R5Fcst|MountainFcst:N34StarFcst]",
       "[PlainsFcst|CapInScen:InsSclInScen:CurPropConv:ScnRelPlFcst]",
       sep = "")
    arcs_mat = fromto_to_mat(temp$res$arcs, dimnames(temp$data)[[2]])
    result = list(arcs_mat = arcs_mat, node_names = dimnames(data)[[2]],
```

```
        data = data, res = res)
    return(result)
  }
```

---

real_insurance

Usage

```
real_insurance(n, rep = T)
```

Arguments

n	
rep	

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (n, rep = T)
{
    packages = c("bnlearn")
    if (length(setdiff(packages, rownames(installed.packages()))) >
        0) {
        install.packages(setdiff(packages, rownames(installed.packages())))
    }
    data(insurance, package = "bnlearn")
    data = insurance[sample(c(1:20000), n, rep = rep), ]
    res = empty.graph(names(insurance))
    modelstring(res) = paste("[Age][Mileage][SocioEcon|Age][GoodStudent|Age:SocioEcon]",
       "[RiskAversion|Age:SocioEcon][OtherCar|SocioEcon][VehicleYear|SocioEcon:RiskAversion]",
        "[MakeModel|SocioEcon:RiskAversion][SeniorTrain|Age:RiskAversion]",
        "[HomeBase|SocioEcon:RiskAversion][AntiTheft|SocioEcon:RiskAversion]",
        "[RuggedAuto|VehicleYear:MakeModel][Antilock|VehicleYear:MakeModel]",
        "[DrivingSkill|Age:SeniorTrain][CarValue|VehicleYear:MakeModel:Mileage]",
        "[Airbag|VehicleYear:MakeModel][DrivQuality|RiskAversion:DrivingSkill]",
        "[Theft|CarValue:HomeBase:AntiTheft][Cushioning|RuggedAuto:Airbag]",
        "[DrivHist|RiskAversion:DrivingSkill][Accident|DrivQuality:Mileage:Antilock]",
        "[ThisCarDam|RuggedAuto:Accident][OtherCarCost|RuggedAuto:Accident]",
        "[MedCost|Age:Accident:Cushioning][ILiCost|Accident]",
        "[ThisCarCost|ThisCarDam:Theft:CarValue][PropCost|ThisCarCost:OtherCarCost]",
        sep = "")
    arcs_mat = fromto_to_mat(temp$res$arcs, dimnames(temp$data)[[2]])
    result = list(arcs_mat = arcs_mat, node_names = dimnames(data)[[2]],
        data = data, res = res)
    return(result)
  }
```

real_lizards

Usage

```
real_lizards(n, rep = T)
```

Arguments

n

rep

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (n, rep = T)
{
    packages = c("bnlearn")
    if (length(setdiff(packages, rownames(installed.packages()))) >
        0) {
        install.packages(setdiff(packages, rownames(installed.packages())))
    }
    data(lizards, package = "bnlearn")
    data = lizards[sample(c(1:409), n, rep = rep), ]
    res = empty.graph(names(lizards))
    modelstring(res) = "[Species][Diameter|Species][Height|Species]"
    arcs_mat = fromto_to_mat(temp$res$arcs, dimnames(temp$data)[[2]])
    result = list(arcs_mat = arcs_mat, node_names = dimnames(data)[[2]],
        data = data, res = res)
    return(result)
  }
```

toss_value

Usage

```
toss_value(times, num_of_cases, makespace = FALSE)
```

Arguments

times

num_of_cases

makespace

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (times, num_of_cases, makespace = FALSE)
{
    mat_values = merge("Value", c(1:num_of_cases))
    temp = list()
    for (i in 1:times) {
        temp[[i]] = paste(mat_values[, 1], mat_values[, 2], sep = "")
    }
    res = expand.grid(temp, KEEP.OUT.ATTRS = FALSE)
    names(res) = c(paste(rep("toss", times), 1:times, sep = ""))
    if (makespace)
        res$probs = rep(1, 2^times)/2^times
    return(res)
  }
```

Index