Package ‘gRain’

February 19, 2015

Version 1.2-3
Title Graphical Independence Networks
Author Søren Højsgaard <sorenh@math.aau.dk>
Maintainer Søren Højsgaard <sorenh@math.aau.dk>
Description A package for probability propagation in graphical independence networks, also known as probabilistic expert systems or Bayesian networks.
License GPL (>= 2)
Depends R (>= 3.0.2), methods, gRbase (>= 1.7-0)
Imports igraph, graph
URL http://people.math.aau.dk/~sorenh/software/gR/
Encoding latin1
Suggests Rgraphviz
ByteCompile Yes
NeedsCompilation no
Repository CRAN
Date/Publication 2014-03-25 05:41:30

R topics documented:

gRain-package ......................................................... 2
andtable .............................................................. 2
compile.grain .......................................................... 3
compileCPT,compilePOT .......................................... 4
cptable ............................................................... 5
extractCPT,extractPOT .......................................... 7
grain ..................................................................... 8
loadHuginNet ........................................................... 10
predict.grain ........................................................... 12
propagate.grain ....................................................... 13
querygrain ............................................................... 14
repeatPattern ......................................................... 16
Overview - the gRain package for graphical independence networks

Description
Probability propagation in graphical independence networks, (also known as probabilistic expert or Bayesian networks).

Details
This package implements graphical independence networks.
The main function for building networks is `grain`.
The function `querygrain` is used for querying independence networks.
Functions `simulate.grain` and `predict.grain` are available.
There is a small vignette which illustrates the use of gRain.

Author(s)
Søren Højsgaard <sorenh@math.aau.dk>
Maintainer: Søren Højsgaard <sorenh@math.aau.dk>

References

Conditional probability tables based on logical dependencies

Description
Generate conditional probability tables based on the logical expressions AND and OR.

Usage
```r
andtable(v, pa1 = c(TRUE, FALSE), pa2 = c(TRUE, FALSE), levels)
ornable(v, pa1 = c(TRUE, FALSE), pa2 = c(TRUE, FALSE), levels)
```
Arguments

\begin{itemize}
\item \texttt{v} Specifications of the names in \( P(v | p_{a1}, \ldots, p_{ak}) \). See section 'details' for information about the form of the argument.
\item \texttt{pa1,pa2} The coding of the logical parents
\item \texttt{levels} The levels (or rather labels) of \( v \), see 'examples' below
\end{itemize}

Details

Regarding the form of the argument \( v \): To specify \( P(a | b, c) \) one may write \( \sim a | b + c \) or \( \sim a + b + c \) or \( c("a","b","c") \). Internally, the last form is used. Notice that the + operator is used as a separator only. The order of the variables is important so + does not commute.

Value

A cptable.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References


See Also

\begin{itemize}
\item \texttt{cptable}
\end{itemize}

Examples

\begin{verbatim}
ortable(c("v","A","B"), levels=c("yes","no"))
\end{verbatim}


code

```
compile.grain
```

Compile a graphical independence network (a Bayesian network)

Description

Compiles a Bayesian network. This means creating a junction tree and establishing clique potentials.

Usage

```
# S3 method for class 'grain'
compile(object, propagate = FALSE, root = NULL, control = object$control,
        details = 0, ...)
```
Arguments

object: A grain object.
propagate: If TRUE the network is also propagated meaning that the cliques of the junction tree are calibrated to each other.
root: A set of variables which must be in the root of the junction tree
control: Controlling the compilation process.
details: For debugging info. Do not use.
... Currently not used.

Value

A compiled Bayesian network; an object of class grain.

Author(s)
Søren Højsgaard, <sorenh@math.aau.dk>

References

See Also
grain, propagate, triangulate, rip, junctionTree

---

compileCPT, compilePOT  Compile conditional probability tables / cliques potentials.

Description

Compile conditional probability tables / cliques potentials as a preprocessing step for creating a graphical independence network

Usage

compileCPT(x, forceCheck=TRUE, details=0)
compilePOT(x)

Arguments

x: To compileCPT x is a list of conditional probability tables; to compilePOT, x is a list of clique potentials
forceCheck: Controls if consistency checks of the probability tables should be made.
details: Controls amount of print out. Mainly for debugging purposes
**Value**

compileCPT returns a list of class 'cptspec'
compilePOT returns a list of class 'potspec'

**Author(s)**

Søren Højsgaard, <sorenh@math.aau.dk>

**References**


**See Also**

extractCPT, extractPOT

---

### cptable

Create conditional probability tables (CPTs)

**Description**

Creates conditional probability tables of the form \( p(v|pa(v)) \).

**Usage**

```r
cptable(vpar, levels=NULL, values=NULL, normalize=TRUE, smooth=0)
```

**Arguments**

- `vpar`: Specifications of the names in \( p(v|pa1,...,pak) \). See section 'details' for information about the form of the argument.
- `values`: Probabilities; recycled if necessary. Regarding the order, please see section 'details' and the examples.
- `normalize`: See 'details' below.
- `smooth`: See 'details' below.
- `levels`: See 'details' below.

**Details**

If `normalize=TRUE` then for each configuration of the parents the probabilities are normalized to sum to one.

If `smooth` is non–zero then zero entries of `values` are replaced with `smooth` before normalization takes place.

Regarding the form of the argument `vpar`: To specify \( P(a|b,c) \) one may write \( \sim a|b\cdot c \), \( \sim a:b\cdot c \), \( \sim a|b+c \), \( \sim a+b+c \) or \( c(\text{"a"},\text{"b"},\text{"c"}) \). Internally, the last form is used. Notice that the + and :
operator is used as a separator only. The order of the variables is important so the operators do not commute.

If a has levels a1, a2 and likewise for b and c then the order of values corresponds to the configurations (a1,b1,c1), (a2,b1,c1) (a1,b2,c1), (a2,b2,c1) etc. That is, the first variable varies fastest. Hence the first two elements in values will be the conditional probabilities of a given b=b1, c=c1.

Value

A cptable object (a list).

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References


See Also

andtable, ortable, extractCPT, compileCPT, extractPOT, compilePOT, grain

Examples

yn <- c("yes","no")
ynm <- c("yes","no","maybe")
a <- cptable(~ asia, values=c(1,99), levels=yn)
t.a <- cptable(~ tub : asia, values=c(5,95,1,99,1,999), levels=ynm)
d.a <- cptable(~ dia : asia, values=c(5,5,1,99,100,999), levels=ynm)
cptlist <- compileCPT(list(a,t.a,d.a))
grain(cptlist)

## Example: Specifying conditional probabilities as a matrix
bayes.levels <- c("Enzyme", "Keratine", "unknown")
root.node <- cptable(~R, values=c(1, 1, 1 ), levels=bayes.levels)
cond.prob.tbl <- t(matrix( c( 1, 0, 0, 0, 1, 0, 0.5, 0.5, 0 ),
                           nrow=3, ncol=3, byrow=TRUE, dimnames=list(bayes.levels, bayes.levels)))
cond.prob.tbl

## Notice above: Columns represent parent states; rows represent child states
query.node <- cptable(~ Q | R, values=cond.prob.tbl, levels=bayes.levels )
sister.node <- cptable(~ S | R, values=cond.prob.tbl, levels=bayes.levels )

## Testing
compile(grain(compileCPT(list( root.node, query.node, sister.node ))), propagate=TRUE)
extractCPT, extractPOT

Extract conditional probabilities and clique potentials from data

Description

Extract list of conditional probability tables and list of clique potentials from data

Usage

extractCPT(x, graph, smooth = 0)
extractPOT(x, graph, smooth = 0)

Arguments

x
A table or dataframe

graph
A graph represented as a graphNEL object. For extractCPT, graph must be a DAG while for extractPOT, graph must be undirected triangulated graph.

smooth
See 'details' below

Details

If smooth is non–zero then smooth is added to all cell counts before normalization takes place.

Value

extractCPT: A list of conditional probability tables
extractPOT: A list of clique potentials

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References


See Also

compileCPT, compilePOT, grain

Examples

## Asia (chest clinic) example:

## Version 1) Specify conditional probability tables.
yn <- c("yes","no")
a <- cptable(~asia, values=c(1,99), levels=yn)
The `grain` builds a graphical independence network.

**Usage**

```r
grain(x, data, control=list(), smooth=0, details=0, ...)
```
Arguments

x An argument to build an independence network from. Typically a list of conditional probability tables, a DAG or an undirected graph. In the two latter cases, data must also be provided.

data An optional data set (currently must be an array/table)

control A list defining controls, see 'details' below.

smooth A (usually small) number to add to the counts of a table if the grain is built from a graph plus a dataset.

details Debugging information.

Details

If 'smooth' is non-zero then entries of 'values' which a zero are replaced by the value of 'smooth' - BEFORE any normalization takes place.

Value

An object of class "grain"

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References


See Also

cptable, compile.grain, propagate.grain, setFinding, setEvidence, getFinding, pFinding, retractFinding

Examples

```r
## Asia (chest clinic) example:
yn <- c("yes","no")
a <- cptable(~asia, values=c(1,99), levels=yn)
t.a <- cptable(~tub+asia, values=c(5,95,1,99), levels=yn)
s <- cptable(~smoke, values=c(5,5), levels=yn)
l.s <- cptable(~lung+smoke, values=c(1,9,1,99), levels=yn)
b.s <- cptable(~bronc+smoke, values=c(6,4,3,7), levels=yn)
e.lt <- cptable(~either+lung+tub, values=c(1,0,1,0,1,0,1), levels=yn)
x.e <- cptable(~xray+either, values=c(98,2,5,95), levels=yn)
d.be <- cptable(~dysp+bronc+either, values=c(9,1,7,3,8,2,1,9), levels=yn)
plist <- compileCPT(list(a, t.a, s, l.s, b.s, e.lt, x.e, d.be))
np <- grain(plist)
```
pn
summary(pn)
plot(pn)
pnc <- compile(pn, propagate=TRUE)

## If we want to query the joint distribution of the disease nodes,
## computations can be speeded up by forcing these nodes to be in
## the same clique of the junction tree:
pnc2 <- compile(pn, root=c("lung", "brone", "tub"), propagate=TRUE)

system.time(
  for (i in 1:200)
    querygrain(pnc, nodes=c("lung","brone", "tub"), type="joint")))

## Create network from gmData (with data) and graph specification.
## There are different ways:
data(HairEyeColor)
d <- HairEyeColor
daG <- dagList(list(~Hair, ~Eye:Hair, ~Sex:Hair))
class( daG )
uG <- ugList(list(~Eye:Hair, ~Sex:Hair))
class( uG )

## Create directly from dag:
b1 <- grain( daG, d )
class( b1 )

## Build model from undirected (decomposable) graph
b3 <- grain( uG, d )
class( b3 )

## Simple example - one clique only in triangulated graph:
plist <- compileCPT( list(a, t.a) )
pn <- grain( plist )
querygrain(pn)

## Simple example - disconnected network:
plist <- compileCPT( list(a, t.a, s) )
pn <- grain( plist )
querygrain(pn)
**loadHuginNet**

**Description**

These functions can load a net file saved in the 'Hugin format' into R and save a network in R as a file in the 'Hugin format'.

**Usage**

```r
loadHuginNet(file, description, details = FALSE)
saveHuginNet(gin, file, details = FALSE)
```

**Arguments**

- `gin`: An independence network
- `file`: Name of HUGIN net file. Convenient to give the file the extension '.net'
- `description`: A text describing the network, defaults to `file`
- `details`: Debugging information

**Value**

An object (a list) of class "huginNet".

**Author(s)**

Søren Højsgaard, <sorenh@math.aau.dk>

**References**


**See Also**

- `grain`

**Examples**

```r
tf <- system.file("huginex", "chest.clinic.net", package = "gRain")
chest <- loadHuginNet(tf, details=1)
chest

td <- tempdir()
saveHuginNet(chest, paste(td,"/chest.net",sep=''))

chest2 <- loadHuginNet(paste(td,"/chest.net",sep=''))

tf <- system.file("huginex", "golf.net", package = "gRain")
golf <- loadHuginNet(tf, details=1)
```
predict.grain

Make predictions from a probabilistic network

Description

Makes predictions (either as the most likely state or as the conditional distributions) of variables conditional on finding (evidence) on other variables in a independence network.

Usage

## S3 method for class 'grain'
predict(object, response, predictors, newdata, type = "class", ...)

Arguments

- object: A grain object
- response: A vector of response variables to make predictions on
- predictors: A vector of predictor variables to make predictions from. Defaults to all variables that are not responses.
- newdata: A data frame
- type: If "class", the most probable class is returned; if "distribution" the conditional distribution is returned.
- ... Not used

Value

A list with components

- pred: A list with the predictions
- pFinding: A vector with the probability of the finding (evidence) on which the prediction is based

Author(s)

Søren Höjsgaard, <sorenh@math.aau.dk>

References

propagate.grain

See Also

grain

propagate.grain  Propagate a graphical independence network (a Bayesian network)

Description

Propagation refers to calibrating the cliques of the junction tree so that the clique potentials are consistent on their intersections.

Usage

## S3 method for class 'grain'
propagate(object, details = object$details,...)

Arguments

- **object**: A grain object
- **details**: For debugging info
- **...**: Currently not used

Value

A compiled and propagated grain object.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References


See Also

grain.compile
querygrain  

Set evidence and query a network

Description

Query an independence network, i.e. obtain the conditional distribution of a set of variables given finding (evidence) on other variables.

Usage

querygrain(object, nodes = nodeNames(object), normalize = TRUE,
  type = "marginal", result="array", details = 0)

setEvidence(object, nodes=NULL, states=NULL, nslist=NULL, propagate=TRUE)
retractEvidence(object, nodes=NULL, propagate=TRUE)
getEvidence(object)
pEvidence(object)

setFinding(object, nodes=NULL, states=NULL, flist=NULL, propagate=TRUE)
retractFinding(object, nodes=NULL, propagate=TRUE)
getFinding(object)
pFinding(object)

Arguments

object  A "grain" object

nodes   A vector of nodes

states  A vector of states (of the nodes given by 'nodes')

flist   An alternative way of specifying findings (evidence), see examples below.

nslist  An alternative way of specifying findings (evidence), see examples below.

propagate Should the network be propagated?

normalize Should the results be normalized to sum to one.

type    Valid choices are "marginal" which gives the marginal distribution for each node in nodes; "joint" which gives the joint distribution for nodes and "conditional" which gives the conditional distribution for the first variable in nodes given the other variables in nodes.

result  If "data.frame" the result is returned as a data frame (or possibly as a list of dataframes).

details Debugging information

Value

A list of tables with potentials.
Note

`setEvidence()` is an improvement of `setFinding()` (and as such `setFinding` is obsolete). Users are recommended to use `setEvidence()` in the future.

`setEvidence()` allows to specification of "hard evidence" (specific values for variables) and likelihood evidence (also known as virtual evidence) for variables.

The syntax of `setEvidence()` may change in the future.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References


See Also

cptable

Examples

testfile <- system.file("huginex", "chest_clinic.net", package = "gRain")
chest <- loadHuginNet(testfile, details=0)
qb <- querygrain(chest)
qb

lapply(qb, as.numeric)
sapply(qb, as.numeric)

## setFinding / setEvidence

yn <- c("yes","no")
a <- cptable(~asia, values=c(1,99), levels=yn)
t.a <- cptable(~tub+asia, values=c(5,95,1,99), levels=yn)
s <- cptable(~smoke, values=c(5,5), levels=yn)
l.s <- cptable(~lung+smoke, values=c(1,9,1,99), levels=yn)
b.s <- cptable(~bronc+smoke, values=c(0,4,3,7), levels=yn)
e.lt <- cptable(~either+lung+tub, values=c(1,0,1,0,1,0,0,0,1), levels=yn)
x.e <- cptable(~xray+either, values=c(98,2,5,95), levels=yn)
d.be <- cptable(~dysp+bronc+either, values=c(9,1,7,3,8,2,1,9), levels=yn)
plist <- compileCPT(list(a, t.a, s, l.s, b.s, e.lt, x.e, d.be))
chest <- grain(plist)

## 1) These two forms are identical
setEvidence(chest, c("asia","xray"), c("yes", "yes"))
setFinding(chest, c("asia","xray"), c("yes", "yes"))

## 2) Suppose we do not know with certainty whether a patient has
## recently been to Asia. We can then introduce a new variable
repeatPattern

Create repeated patterns in Bayesian networks

Description

Repeated patterns is a useful model specification short cut for Bayesian networks

Usage

repeatPattern(plist, instances, unlist = TRUE)

Arguments

plist A list of conditional probability tables. The variable names must have the form name[i] and the i will be substituted by the values given in instances below.
instances A vector of distinct integers
unlist If FALSE the result is a list in which each element is a copy of plist in which name[i] are substituted. If TRUE the result is the result of applying unlist().

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References


See Also

grain, compileCPT
Examples

```r
## Specify hidden markov models. The x[i]'s are unobserved, the
## y[i]'s can be observed.

yn <- c("yes","no")

## Specify p(x0)
x.0 <- cptable(~x0, values=c(1,1), levels=yn)

## Specify transition density
x.x <- cptable(~x[i]|x[i-1], values=c(1,99,2,98), levels=yn)

## Specify emission density
y.x <- cptable(~y[i]|x[i], values=c(1,99,2,98), levels=yn)

## The pattern to be repeated
pp <- list(x.x, y.x)

## Repeat pattern and create network
ppp <- repeatPattern(pp, instances=1:10)
qqq <- compileCPT(c(list(x.0), ppp))
rrr <- grain(qqq)
```

simulate.grain

Simulate data from an independence network.

Description

Simulate data from an independence network.

Usage

```r
## S3 method for class 'grain'
simulate(object, nsim = 1, seed = NULL, ...)
```

Arguments

- `object`: An independence network
- `nsim`: Number of cases to simulate
- `seed`: An optional integer controlling the random number generation
- `...`: Not used...

Value

A data frame
Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References


Examples

```r
## Not run:

tf <- system.file("huginex", "chest_clinic.net", package = "gRain")
chest <- loadHuginNet(tf, details=1)
simulate(chest,n=10)

chest2 <- setFinding(chest, c("VisitToAsia", "Dyspnoea"), c("yes", "yes"))
simulate(chest2,n=10)

## End(Not run)
```

update.CPTgrain  

Update a Bayesian network

Description

Update a Bayesian network

Usage

```r
## S3 method for class 'CPTgrain'
update(object, ...)
```

Arguments

- `object`: A Bayesian network of class CPTgrain
- `...`: If CPTlist is a name in the dotted list, then the object will be update with this value (which is assumed to be a list of conditional probabilities).

Value

A new Bayesian network.
Note

There is NO checking that the input matches the settings in the Bayesian network.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References


Examples

```r
## Network for Bernulli experiment; two nodes: X and thetaX
yn <- c("yes", "no")  # Values for X
thX.val <- c(.3, .5, .7)  # Values for thetaX
prX.val <- rep(1, length(thX.val))  # Probabilities for thetaX values

thX <- cptable(~thetaX, values=prX.val, levels=thX.val)
X <- cptable(~X|thetaX, values=rbind(thX.val, 1-thX.val), levels=yn)

cptlist <- compileCPT( list(thX, X) )
bn <- compile( grain( cptlist ) )
querygrain( setEvidence(bn, nodes="X", states="yes") )

## To insert a new prior distribution we may do as follows
## (where we can omit the process of recompiling the network)
prX.val2 <- c(.2, .3, .5)
thX2 <- cptable(~thetaX, values=prX.val2, levels=thX.val)
bn2 <- update(bn, CPTlist=compileCPT( list(thX2, X) )
querygrain( setEvidence(bn2, nodes="X", states="yes") )
```
Index

*Topic models
  compile.grain, 3
cptable, 5
grain, 8
predict.grain, 12
propagate.grain, 13
querygrain, 14
simulate.grain, 17

*Topic package
  gRain-package, 2

*Topic utilities
  andtable, 2
  compile.grain, 3
  compileCPT, compilePOT, 4
  extractCPT, extractPOT, 7
  loadHuginNet, 10
  propagate.grain, 13
  querygrain, 14
  update.CPTgrain, 18

*Topic utils
  repeatPattern, 16

andtable, 2, 6
compile, 13
compile.CPTgrain (compile.grain), 3
compile.grain, 3, 9
compilePOTgrain (compile.grain), 3
compileCPT, 6, 7, 16
compileCPT (compileCPT, compilePOT), 4
compileCPT, compilePOT, 4
compilePOT, 6, 7
compilePOT (compileCPT, compilePOT), 4
cptable, 3, 5, 9, 15
extractCPT, 5, 6
extractCPT (extractCPT, extractPOT), 7
extractCPT, extractPOT, 7
extractPOT, 5, 6
extractPOT (extractCPT, extractPOT), 7
getEvidence (querygrain), 14
gEvidence, 9
gFinding (querygrain), 14
gRain (gRain-package), 2
gRain-package, 2
iplot.grain (grain), 8
junctionTree, 4
loadHuginNet, 10
nodeNames (grain), 8
nodeStates (grain), 8
ortable, 6
ortable (andtable), 2
pEvidence (querygrain), 14
pFinding, 9
pFinding (querygrain), 14
plot.grain (grain), 8
predict.grain, 2, 12
propagate, 4
propagate.grain, 9, 13
propagateLS (propagate.grain), 13
querygrain, 2, 14
repeatPattern, 16
retractEvidence (querygrain), 14
retractFinding, 9
retractFinding (querygrain), 14
rip, 4
saveHuginNet (loadHuginNet), 10
setEvidence, 9
setEvidence (querygrain), 14
setFinding, 9
setFinding (querygrain), 14
INDEX

simulate grain, 2, 17
summary CPTspec
    (compileCPT, compilePOT), 4

triangulate, 4

update CPT grain, 18