

Package ‘gRain’

November 22, 2009

Version 0.8.2

Title Graphical Independence Networks

Author Søren Højsgaard <sorenh@agrsci.dk>

Maintainer Søren Højsgaard <sorenh@agrsci.dk>

Description A package for probability propagation in graphical independence networks, also known as probabilistic expert systems (which includes Bayesian networks as a special case).

License GPL (>= 2)

Depends R (>= 2.1.0),methods,gRbase,graph,RBGL

Imports graph

Suggests Rgraphviz

URL <http://genetics.agrsci.dk/~sorenh/public/R/gRainweb/>

Encoding latin1

Repository CRAN

Date/Publication 2009-11-22 16:11:40

R topics documented:

gRain-package	2
andtable	2
compile,propagate	3
compile.grain	4
compileCPT,compilePOT	5
cptable	5
extractCPT,extractPOT	6
grain	7
loadHuginNet	9
predict.grain	10
propagate.grain	11
querygrain	12
simulate.grain	13

Index**15**

gRain-package	<i>Overview - the gRain package for graphical independence networks</i>
---------------	---

Description

Probability propagation in graphical independence networks, also known as probabilistic expert systems (which includes Bayesian networks).

Details

This package implements graphical independence networks.

The package is still at an experimental stage.

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.

A paper describing gRain is under preparation

Author(s)

Søren Højsgaard <sorenh@agrsci.dk>

Maintainer: Søren Højsgaard <sorenh@agrsci.dk>

andtable	<i>Conditional probability tables based on logical dependencies</i>
----------	---

Description

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

Usage

```
andtable(v, pa = NULL, pa1 = c(TRUE, FALSE), pa2 = c(TRUE, FALSE), levels)
ortable(v, pa = NULL, pa1 = c(TRUE, FALSE), pa2 = c(TRUE, FALSE), levels)
```

Arguments

v	A vertex
pa	The parents of vertex v
pa1, pa2	The coding of the logical parents
levels	The levels (or rather labels) of v, see 'examples' below

Value

A ctable.

Author(s)

Søren Højsgaard, sorenh@agrsci.dk

See Also

[cetable](#)

Examples

```
ortable(c("v", "A", "B"), levels=c("yes", "no"))
```

compile,propagate *Compile and propagate functions*

Description

`compile` and `propagate` are generic functions which invoke particular methods which depend on the class of the first argument

Usage

```
compile(object, ...)  
propagate(object, ...)
```

Arguments

<code>object</code>	An object to be compiled or propagated
<code>...</code>	Additional arguments which depends on the class of the object

Value

The value returned depends on the class of the first argument.

Author(s)

Søren Højsgaard, sorenh@agrsci.dk

See Also

[compile.grain,propagate.grain](#)

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

```
compile.grain(object, method = "mcwh", propagate = FALSE, root = NULL, smooth = 0,
```

Arguments

object	A grain object
method	Triangulation method. Currently only "mcwh" is available
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
smooth	If non-zero this value is added to all potentials (to avoid zeros in the joint distribution)
control	Controlling the compilation process.
details	For debugging info. Do not use.
...	Currently not used.

Value

A compiled Bayesian network; an object of class `compgrain`.

Author(s)

Søren Højsgaard, sorenh@agrsci.dk

See Also

[grain](#), [propagate](#)

```
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)
compilePOT(x)
```

Arguments

`x` To `compileCPT` `x` is a list of conditional probability tables; to `compilePOT`, `x` is a list of clique potentials

Value

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

Author(s)

~~who you are~~

See Also

[extractCPT](#), [extractPOT](#)

```
cptable
```

Create conditional probability tables (CPTs)

Description

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

Usage

```
cptable(v, pa=NULL, levels=NULL, values = NULL, normalize = TRUE, smooth = 0)
```

Arguments

<code>v</code>	A vertex
<code>pa</code>	The parents of vertex <code>v</code>
<code>values</code>	Probabilities; recycled if necessary
<code>normalize</code>	See 'details' below.
<code>smooth</code>	See 'details' below.
<code>levels</code>	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.

Value

If a `gmData` object is given, then `'cpt'` returns an object of class `'ctab'` (which is a general representation of a table).

If no `gmData` object is given, then `'cpt'` returns an object of class `'cptTemplate'`.

`'compileCPT'` returns an object of class `'cptspec'` (which is just a list with a special class attribute).

Author(s)

Søren Højsgaard, sorenh@agrsci.dk

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)
compileCPT(list(a,t.a,d.a))
```

`extractCPT,extractPOT`

Extract conditional probabilities and clique potentials from data

Description

Extract conditional probabilities and clique potentials from data

Usage

```
extractCPT(x, graph, V = nodes(graph), smooth = 0)
extractPOT(x, graph, smooth = 0)
```

Arguments

x	A contingency table (an array)
graph	To extractCPT graph is a DAG while to extractPOT graph is an undirected triangulated graph.
V	The set of vertices for which CPTs should be extracted
smooth	See 'details' below

Details

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

Value

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

Author(s)

Søren Højsgaard, sorenh@agrsci.dk

See Also

[compileCPT](#), [compilePOT](#), [grain](#)

grain

Graphical Independence Network

Description

The 'grain' builds a graphical independence network.

Usage

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

Arguments

<code>x</code>	An argument to build an independence network from.
<code>data</code>	An optional data set (currently must be an array/table)
<code>control</code>	A list defining controls, see 'details' below.
<code>smooth</code>	A (usuall small) number to add to the counts of a table if the grain is built from a graph plus a dataset.
<code>details</code>	Debugging information.
<code>...</code>	Additional arguments, currently not used.

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@agrsci.dk

See Also

[cptable](#), [setFinding](#), [getFinding](#), [pFinding](#), [retractFinding](#)

Examples

```

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(1,0,1,0,1,0,0,1), 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))
pn <- grain(plist)
pn

summary(pn)
plot(pn)

```

```
data(HairEyeColor)
d <- HairEyeColor
dag <- dagList(list(~Hair, ~Eye+Hair, ~Sex+Hair))
class(dag)
ug <- ugList(list(~Eye+Hair, ~Sex+Hair))
class(ug)

b1 <- grain(dag,d)
class(b1)

b3 <- grain(ug,d)
class(b3)

plist <- compileCPT(list(a, t.a))
pn <- grain(plist)
querygrain(pn)

plist <- compileCPT(list(a, t.a, s))
pn <- grain(plist)
querygrain(pn)
```

loadHuginNet *Load and save Hugin net files*

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

```
loadHuginNet(file, description, details = 0)
saveHuginNet(gin, file, details = 0)
```

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@agrsci.dk

See Also

[grain](#)

Examples

```
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)
```

```
saveHuginNet(golf, paste(td, "/golf.net", sep=''))
golf2 <- loadHuginNet(paste(td, "/golf.net", sep=''))
```

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 an 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 Højsgaard, sorenh@agrsci.dk

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

```
propagate.grain(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@agrsci.dk

See Also

[grain,compile](#)

querygrain

Query an independence 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 = nodeName(object), normalize = TRUE, type =
c("marginal", "joint", "conditional"), return="array", details = 0)

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.
propagate	Should the network be propagated?
normalize	Should the results be normalized to sum to one.
type	Should marginals (for each node), the joint for all nodes, or the conditional of the first node given the rest be returned.
return	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

Author(s)

Søren Højsgaard, sorenh@agrsci.dk

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)
```

simulate.grain	<i>Simulate from an independence network</i>
----------------	--

Description

Simulate data from an independence network.

Usage

```
## 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@agrsci.dk

Examples

```
## 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)
```

Index

*Topic **models**

- compile.grain, 4
- cptable, 5
- grain, 7
- predict.grain, 10
- propagate.grain, 11
- querygrain, 12
- simulate.grain, 13

*Topic **package**

- gRain-package, 2

*Topic **utilities**

- andtable, 2
- compile, propagate, 3
- compile.grain, 4
- compileCPT, compilePOT, 5
- extractCPT, extractPOT, 6
- loadHuginNet, 9
- propagate.grain, 11
- querygrain, 12

andtable, 2

compile, 12

compile(*compile*, *propagate*), 3

compile, propagate, 3

compile.cpt-grain
(*compile.grain*), 4

compile.grain, 3, 4

compile.pot-grain
(*compile.grain*), 4

compileCPT, 7

compileCPT
(*compileCPT*, *compilePOT*), 5

compileCPT, compilePOT, 5

compilePOT, 7

compilePOT
(*compileCPT*, *compilePOT*), 5

cptable, 3, 5, 8, 13

extractCPT, 5

extractCPT

(*extractCPT*, *extractPOT*), 6

extractCPT, extractPOT, 6

extractPOT, 5

extractPOT
(*extractCPT*, *extractPOT*), 6

getFinding, 8

getFinding(*querygrain*), 12

gRain(*gRain-package*), 2

grain, 2, 4, 7, 7, 10–12

gRain-package, 2

loadHuginNet, 9

nodeName(*grain*), 7

nodeStates(*grain*), 7

ortable(*andtable*), 2

pFinding, 8

pFinding(*querygrain*), 12

plot.grain(*grain*), 7

predict.grain, 2, 10

print.cptable(*cptable*), 5

print.CPTspec
(*compileCPT*, *compilePOT*), 5

print.grain(*grain*), 7

print.grainFinding(*querygrain*),
12

print.POTspec
(*compileCPT*, *compilePOT*), 5

propagate, 4

propagate(*compile*, *propagate*), 3

propagate.grain, 3, 11

querygrain, 2, 12

retractFinding, 8

retractFinding(*querygrain*), 12

`saveHuginNet (loadHuginNet), 9`
`setFinding, 8`
`setFinding (querygrain), 12`
`simulate.grain, 2, 13`