Package 'PNetica'

March 31, 2019

Version 0.3-6

Date 2018/11/04

Title Parameterized Bayesian Networks Netica Interface

Author Russell Almond

Maintainer Russell Almond <ralmond@fsu.edu>

Depends R (>= 3.0), RNetica (>= 0.5), CPTtools (>= 0.4), Peanut (>= 0.3-4)

Description This package provides RNetica implementation of Peanut interface.

License Artistic-2.0

URL http://pluto.coe.fsu.edu/RNetica

R topics documented:

`PNetica-package` *Parameterized Bayesian Networks Netica Interface*

Description

This package provides RNetica implementation of Peanut interface.

Details

The DESCRIPTION file: This package was not yet installed at build time.

The `Peanut` package provides a set of generic functions for manipulation parameterized networks, in particular, for the abstract `Pnet` and `Pnode` classes. This package provides concrete implementations of those classes using the built in classes of `RNetica`. In particular, `Pnet.NeticaBN` extends `NeticaBN` and `Pnode.NeticaNode` extends `NeticaNode`.

The properties of the `Pnet` and `Pnode` objects are stored as serialized Netica user fields (see `NetworkUserObj` and `NodeUserObj`).

The `as.Pnet` (`as.Pnode`) method for a `NeticaBN` (`NeticaNode`) merely adds "Pnet" ("Pnode") to `class(net)` (`class(node)`). All of the methods in the `PNetica` are defined for either the `NeticaBN` or `NeticaNode` object, so strictly speaking, adding the "Pnet" or "Pnode" class is not necessary, but it is recommended in case this is used in the future.

PNetica Specific Implementation Details

Here are some Netica specific details which may not be apparent from the description of the generic functions in the `Peanut` package.

1. The `cases` argument to `calcPnetLLike.NeticaBN`, `calcExpTables.NeticaBN` and `GEMfit` all expect the pathname of a Netica case file (see `write.CaseFile`).
2. The methods `calcPnetLLike.NeticaBN`, `calcExpTables.NeticaBN`, and therefore `GEMfit` when called with a `Pnet.NeticaBN` as the first argument, expect that there exists a node set (see `NetworkNodesInSet`) called "onodes" corresponding to the observable variables in the case file `cases`.
3. The function `CompileNetwork` needs to be called before calls to `calcPnetLLike.NeticaBN`, `calcExpTables.NeticaBN` and `GEMfit`.

4. The method PnetPnodes.NeticaBN stores its value in a nodeset called "pnodes". It is recommended that the accessor function be used for modifying this field.
5. The PnetPriorWeight.NeticaBN field of the Pnet.NeticaBN object and all of the fields of the Pnode.NeticaNode are stored in serialized user fields with somewhat obvious names (see NetworkUserObj and NodeUserObj). These fields should not be used for other purposes.

Creating and Restoring Pnet.NeticaBN objects

As both the nodesets and and user fields are serialized when Netica serializes a network (WriteNetworks) the fields of the Pnet.NeticaBN and Pnode.NeticaNode objects should be properly saved and restored. The only thing which will not be restored is the code "Pnet" or "Pnode" class marker. These can be restored by calling as.Pnet on the restored network and as.Pnode on each of the restored Pnodes (see Examples).

The first time the network and nodes are created, it is recommended that Pnet.default and Pnode.NeticaNode (or simply the generic functions Pnet and Pnode. Note that calling Pnode.NeticaNode will calculate defaults for the PnodeLnAlphas and PnodeBetas based on the current value of NodeParents(node), so this should be set before calling this function. (See examples).

Index

Index: This package was not yet installed at build time.

Legal Stuff

Netica and Norsys are registered trademarks of Norsys, LLC (http://www.norsys.com/), used by permission.

Extensive use of PNetica will require a Netica API license from Norsys. This is basically a requirement of the RNetica package, and details are described more fully there. Without a license, RNetica and PNetica will work in a student/demonstration mode which limits the size of the network.

Although Norsys is generally supportive of the RNetica project, it does not officially support RNetica, and all questions should be sent to the package maintainers.

Author(s)

Russell Almond

Maintainer: Russell Almond <ralmond@fsu.edu>

References

Almond, R. G. (2015) An IRT-based Parameterization for Conditional Probability Tables. Paper presented at the 2015 Bayesian Application Workshop at the Uncertainty in Artificial Intelligence Conference.

See Also

PNetica depends on the following other packages.

RNetica A binding of the Netica C API into R.

Peanut An the generic functions for which this package provides implementations.

CPTtools A collection of implementation independent Bayes net utilities.

Examples

```
sess <- NeticaSession()
startSession(sess)

## Building CPTs
tNet <- CreateNetwork("TestNet", session=sess)


theta1 <- NewDiscreteNode(tNet,"theta1",
                          c("VH","High","Mid","Low","VL"))
NodeLevels(theta1) <- effectiveThetas(NodeNumStates(theta1))
NodeProbs(theta1) <- rep(1/NodeNumStates(theta1),NodeNumStates(theta1))
theta2 <- NewDiscreteNode(tNet,"theta2",
                          c("VH","High","Mid","Low","VL"))
NodeLevels(theta2) <- effectiveThetas(NodeNumStates(theta2))
NodeProbs(theta2) <- rep(1/NodeNumStates(theta2),NodeNumStates(theta2))

partial3 <- NewDiscreteNode(tNet,"partial3",
                            c("FullCredit","PartialCredit","NoCredit"))
NodeParents(partial3) <- list(theta1,theta2)

partial3 <- Pnode(partial3,Q=TRUE, link="partialCredit")
PnodePriorWeight(partial3) <- 10
BuildTable(partial3)

## Set up so that first skill only needed for first transition, second
## skill for second transition; adjust alphas to match
PnodeQ(partial3) <- matrix(c(TRUE,TRUE,
                             TRUE,FALSE), 2,2, byrow=TRUE)
PnodeLnAlphas(partial3) <- list(FullCredit=c(-.25,.25),
                                PartialCredit=0)
BuildTable(partial3)
partial4 <- NewDiscreteNode(tNet,"partial4",
                            c("Score4","Score3","Score2","Score1"))
NodeParents(partial4) <- list(theta1,theta2)
partial4 <- Pnode(partial4, link="partialCredit")
PnodePriorWeight(partial4) <- 10

## Skill 1 used for first transition, Skill 2 used for second
## transition, both skills used for the 3rd.

PnodeQ(partial4) <- matrix(c(TRUE,TRUE,
                             FALSE,TRUE,
                             TRUE,FALSE), 3,2, byrow=TRUE)
```

```
PnodeLnAlphas(partial4) <- list(Score4=c(.25,.25),
                                Score3=0,
                                Score2=-.25)
BuildTable(partial4)

## Fitting Model to data

irt10.base <- ReadNetworks(file.path(library(help="PNetica")$path,
                           "testnets","IRT10.2PL.base.dne"), session=sess)
irt10.base <- as.Pnet(irt10.base)  ## Flag as Pnet, fields already set.
irt10.theta <- NetworkFindNode(irt10.base,"theta")
irt10.items <- PnetPnodes(irt10.base)
## Flag items as Pnodes
for (i in 1:length(irt10.items)) {
  irt10.items[[i]] <- as.Pnode(irt10.items[[i]])

}


casepath <- file.path(library(help="PNetica")$path,
                           "testdat","IRT10.2PL.200.items.cas")
## Record which nodes in the casefile we should pay attention to
NetworkNodesInSet(irt10.base,"onodes") <-
   NetworkNodesInSet(irt10.base,"observables")


BuildAllTables(irt10.base)
CompileNetwork(irt10.base) ## Netica requirement

item1 <- irt10.items[[1]]
priB <- PnodeBetas(item1)
priA <- PnodeAlphas(item1)
priCPT <- NodeProbs(item1)

gemout <- GEMfit(irt10.base,casepath)


DeleteNetwork(irt10.base)
DeleteNetwork(tNet)
stopSession(sess)
```

BNWarehouse *A warehouse for Bayesian networks.*

Usage

```
BNWarehouse(...)
```

Arguments

...

`BuildTable.NeticaNode` *Builds the conditional probability table for a Pnode*

Description

The function `BuildTable` calls `calcDPCFrame` to calculate the conditional probability for a `Pnode` object, and sets the current conditional probability table of `node` to the resulting value. It also sets the `NodeExperience(node)` to the current value of `GetPriorWeight(node)`.

Usage

```
## S3 method for class 'NeticaNode'
BuildTable(node)
```

Arguments

`node`	A `Pnode.NeticaNode` object whose table is to be built.

Details

The fields of the `Pnode` object correspond to the arguments of the `calcDPCTable` function. The output conditional probability table is then set in the node object in using the `[]` (Extract.NeticaNode) operator.

In addition to setting the CPT, the weight given to the nodes in the EM algorithm are set to `GetPriorWeight(node)`, which will extract the value of `PnodePriorWeight(node)` or if that is null, the value of `PnetPriorWeight(NodeParents(node))` and set `NodeExperience(node)` to the resulting value.

Value

The `node` argument is returned invisibly. As a side effect the conditional probability table and experience of `node` is modified.

Author(s)

Russell Almond

References

Almond, R. G. (2015) An IRT-based Parameterization for Conditional Probability Tables. Paper presented at the 2015 Bayesian Application Workshop at the Uncertainty in Artificial Intelligence Conference.

See Also

Pnode.NeticaNode, Pnode, PnodeQ, PnodePriorWeight, PnodeRules, PnodeLink, PnodeLnAlphas, PnodeAlphas, PnodeBetas, PnodeLinkScale,GetPriorWeight, calcDPCTable, NodeExperience(node), [.NeticaNode

---

calcExpTables.NeticaBN

Calculate expected tables for a Pnet.NeticaBN

Description

The performs the E-step of the GEM algorithm by running the Netica EM algorithm (see LearnCPTs) using the data in cases. After this is run, the conditional probability table for each Pnode.NeticaNode should be the mean of the Dirichlet distribution and the scale parameter should be the value of NodeExperience(node).

Usage

```
## S3 method for class 'NeticaBN'
calcExpTables(net, cases, Estepit = 1,
                          tol = sqrt(.Machine$double.eps))
```

Arguments

net	A Pnet.NeticaBN object representing a parameterized network.
cases	A character scalar giving the file name of a Netica case file (see write.CaseFile).
Estepit	An integer scalar describing the number of steps the Netica should take in the internal EM algorithm.
tol	A numeric scalar giving the stopping tolerance for the internal Netica EM algorithm.

Details

The key to this method is realizing that the EM algorithm built into the Netica (see LearnCPTs) can perform the E-step of the outer GEMfit generalized EM algorithm. It does this in every iteration of the algorithm, so one can stop after the first iteration of the internal EM algorithm.

This method expects the cases argument to be a pathname pointing to a Netica cases file containing the training or test data (see write.CaseFile). Also, it expects that there is a nodeset (see NetworkNodesInSet) attached to the network called "onodes" which references the observable variables in the case file.

Before calling this method, the function BuildTable needs to be called on each Pnode to both ensure that the conditional probability table is at a value reflecting the current parameters and to reset the value of NodeExperience(node) to the starting value of GetPriorWeight(node).

Note that Netica does allow NodeExperience(node) to have a different value for each row the the conditional probability table. However, in this case, each node must have its own prior weight (or

exactly the same number of parents). The prior weight counts as a number of cases, and should be scaled appropriately for the number of cases in `cases`.

The parameters `Estepit` and `tol` are passed `LearnCPTs`. Note that the outer EM algorithm assumes that the expected table counts given the current values of the parameters, so the default value of one is sufficient. (It is possible that a higher value will speed up convergence, the parameter is left open for experimentation.) The tolerance is largely irrelevant as the outer EM algorithm does the tolerance test.

Value

The `net` argument is returned invisibly.

As a side effect, the internal conditional probability tables in the network are updated as are the internal weights given to each row of the conditional probability tables.

Author(s)

Russell Almond

References

Almond, R. G. (2015) An IRT-based Parameterization for Conditional Probability Tables. Paper presented at the 2015 Bayesian Application Workshop at the Uncertainty in Artificial Intelligence Conference.

See Also

`Pnet`, `Pnet.NeticaBN`, `GEMfit`, `calcPnetLLike`, `maxAllTableParams`, `calcExpTables`, `NetworkNodesInSet` `write.CaseFile`, `LearnCPTs`

Examples

```
sess <- NeticaSession()
startSession(sess)

irt10.base <- ReadNetworks(file.path(library(help="PNetica")$path,
                           "testnets","IRT10.2PL.base.dne"), session=sess)
irt10.base <- as.Pnet(irt10.base)  ## Flag as Pnet, fields already set.
irt10.theta <- NetworkFindNode(irt10.base,"theta")
irt10.items <- PnetPnodes(irt10.base)
## Flag items as Pnodes
for (i in 1:length(irt10.items)) {
  irt10.items[[i]] <- as.Pnode(irt10.items[[i]])
}
CompileNetwork(irt10.base) ## Netica requirement

casepath <- file.path(library(help="PNetica")$path,
                           "testdat","IRT10.2PL.200.items.cas")
## Record which nodes in the casefile we should pay attention to
NetworkNodesInSet(irt10.base,"onodes") <-
   NetworkNodesInSet(irt10.base,"observables")
```

```
item1 <- irt10.items[[1]]

priorcounts <- sweep(NodeProbs(item1),1,NodeExperience(item1),"*")

calcExpTables(irt10.base,casepath)

postcounts <- sweep(NodeProbs(item1),1,NodeExperience(item1),"*")

## Posterior row sums should always be larger.
stopifnot(
  all(apply(postcounts,1,sum) >= apply(priorcounts,1,sum))
)

DeleteNetwork(irt10.base)
stopSession(sess)
```

`calcPnetLLike.NeticaBN` *Calculates the log likelihood for a set of data under a Pnet.NeticaBN model*

Description

The method `calcPnetLLike.NeticaBN` calculates the log likelihood for a set of data contained in `cases` using the current conditional probability tables in a `Pnet.NeticaBN`. Here `cases` should be the filename of a Netica case file (see `write.CaseFile`).

Usage

```
## S3 method for class 'NeticaBN'
calcPnetLLike(net, cases)
```

Arguments

`net`	A `Pnet.NeticaBN` object representing a parameterized network.
`cases`	A character scalar giving the file name of a Netica case file (see `write.CaseFile`).

Details

This function provides the convergence test for the `GEMfit` algorithm. The `Pnet.NeticaBN` represents a model (with parameters set to the value used in the current iteration of the EM algorithm) and `cases` a set of data. This function gives the log likelihood of the data.

This method expects the `cases` argument to be a pathname pointing to a Netica cases file containing the training or test data (see `write.CaseFile`). Also, it expects that there is a nodeset (see `NetworkNodesInSet`) attached to the network called "onodes" which references the observable variables in the case file.

As Netica does not have an API function to directly calculate the log-likelihood of a set of cases, this method loops through the cases in the case set and calls `FindingsProbability(net)` for each one. Note that if there are frequencies in the case file, each case is weighted by its frequency.

Value

A numeric scalar giving the log likelihood of the data in the case file.

Author(s)

Russell Almond

References

Almond, R. G. (2015) An IRT-based Parameterization for Conditional Probability Tables. Paper presented at the 2015 Bayesian Application Workshop at the Uncertainty in Artificial Intelligence Conference.

See Also

`Pnet`, `Pnet.NeticaBN`, `GEMfit`, `calcExpTables`, `BuildAllTables`, `maxAllTableParams` `NetworkNodesInSet`, `FindingsProbability`, `write.CaseFile`

Examples

```
sess <- NeticaSession()
startSession(sess)

irt10.base <- ReadNetworks(file.path(library(help="PNetica")$path,
                           "testnets","IRT10.2PL.base.dne"), session=sess)
irt10.base <- as.Pnet(irt10.base)  ## Flag as Pnet, fields already set.
irt10.theta <- NetworkFindNode(irt10.base,"theta")
irt10.items <- PnetPnodes(irt10.base)
## Flag items as Pnodes
for (i in 1:length(irt10.items)) {
  irt10.items[[i]] <- as.Pnode(irt10.items[[i]])
}
CompileNetwork(irt10.base) ## Netica requirement

casepath <- file.path(library(help="PNetica")$path,
                           "testdat","IRT10.2PL.200.items.cas")
## Record which nodes in the casefile we should pay attention to
NetworkNodesInSet(irt10.base,"onodes") <-
   NetworkNodesInSet(irt10.base,"observables")

llike <- calcPnetLLike(irt10.base,casepath)

DeleteNetwork(irt10.base)
stopSession(sess)
```

ClearWarehouse *Removes entries from a Warehouse*

Usage

```
ClearWarehouse(warehouse)
```

Arguments

```
warehouse
```

is.PnetWarehouse *Tests if an object is a PnetWarehouse or not*

Usage

```
is.PnetWarehouse(obj)
```

Arguments

```
obj
```

is.PnodeWarehouse *Test is an object is a PnodeWarehouse or not*

Usage

```
is.PnodeWarehouse(obj)
```

Arguments

```
obj
```

MakePnet.NeticaBN *Creates a NeticaBN object which is also a Pnet*

Usage

```
MakePnet.NeticaBN(sess, name, data)
```

Arguments

```
sess
name
data
```

MakePnode.NeticaNode *Makes a Pnode which is also a Netica Node*

Usage

```
MakePnode.NeticaNode(net, name, data)
```

Arguments

net

name

data

maxCPTParam.NeticaNode *Find optimal parameters of a Pnode.NeticaNode to match expected tables*

Description

These function assumes that an expected count contingency table can be built from the network; i.e., that LearnCPTs has been recently called. They then try to find the set of parameters maximizes the probability of the expected contingency table with repeated calls to mapDPC. This describes the method for maxCPTParam when the Pnode is a NeticaNode.

Usage

```
## S3 method for class 'NeticaNode'
maxCPTParam(node, Mstepit = 5, tol = sqrt(.Machine$double.eps))
```

Arguments

node	A Pnode object giving the parameterized node.
Mstepit	A numeric scalar giving the number of maximization steps to take. Note that the maximization does not need to be run to convergence.
tol	A numeric scalar giving the stopping tolerance for the maximizer.

Details

This method is called on on a Pnode.NeticaNode object during the M-step of the EM algorithm (see GEMfit and maxAllTableParams for details). Its purpose is to extract the expected contingency table from Netica and pass it along to mapDPC.

When doing EM learning with Netica, the resulting conditional probability table (CPT) is the mean of the Dirichlet posterior. Going from the mean to the parameter requires multiplying the CPT by row counts for the number of virtual observations. In Netica, these are call NodeExperience. Thus, the expected counts are calculated with this expression: `sweep(node[[]], 1, NodeExperience(node), "*")`.

What remains is to take the table of expected counts and feed it into mapDPC and then take the output of that routine and update the parameters.

The parameters `Mstepit` and `tol` are passed to mapDPC to control the gradient decent algorithm used for maximization. Note that for a generalized EM algorithm, the M-step does not need to be run to convergence, a couple of iterations are sufficient. The value of `Mstepit` may influence the speed of convergence, so the optimal value may vary by application. The tolerance is largely irrelevant (if `Mstepit` is small) as the outer EM algorithm does the tolerance test.

Value

The expression `maxCPTParam(node)` returns `node` invisibly. As a side effect the PnodeLnAlphas and PnodeBetas fields of `node` (or all nodes in `PnetPnodes(net)`) are updated to better fit the expected tables.

Author(s)

Russell Almond

References

Almond, R. G. (2015) An IRT-based Parameterization for Conditional Probability Tables. Paper presented at the 2015 Bayesian Application Workshop at the Uncertainty in Artificial Intelligence Conference.

See Also

Pnode, Pnode.NeticaNode, GEMfit, maxAllTableParams mapDPC

`NNWarehouse` *Creates a Warehouse for Netica Nodes*

Usage

```
NNWarehouse(...)
```

Arguments

`...`

PnetFindNode *Finds nodes in a Netica network.*

Description

The function PnetFindNode finds a node in a Pnet with the given name. If no node with the specified name found, it will return NULL. The function PnetAllNodes() returns a list of all nodes in the network.

Usage

```
PnetFindNode(net, name)
PnetAllNodes(net)
```

Arguments

net	The Pnet to search.
name	A character vector giving the name or names of the desired nodes. Names must follow the IDname protocol.

Details

Although each PnetNode belongs to a single network, a network contains many nodes. Within a network, a node is uniquely identified by its name. However, nodes can be renamed (see NodeName()).

The function PnetAllNodes() returns all the nodes in the network, however, the order of the nodes in the network could be different in different calls to this function.

Value

The PnetNode object or list of PnetNode objects corresponding to names, or a list of all node objects for PnetAllNodes(). In the latter case, the names will be set to the node names.

Note

PnetNode objects do not survive the life of a Netica session (or by implication an R session). So the safest way to "save" a PnetNode object is to recreate it using PnetFindNode() after the network is reloaded.

Author(s)

Russell Almond

References

http://norsys.com/onLineAPIManual/index.html, GetNodeNamed_bn(), GetNetNodes_bn()

See Also

NodeNet() retrieves the network from the node.

Examples

```
sess <- NeticaSession()
startSession(sess)

tnet <- CreateNetwork("TestNet",sess)
nodes <- NewDiscreteNode(tnet,c("A","B","C"))

nodeA <- PnetFindNode(tnet,"A")
stopifnot (nodeA==nodes[[1]])

nodeBC <- PnetFindNode(tnet,c("B","C"))
stopifnot(nodeBC[[1]]==nodes[[2]])
stopifnot(nodeBC[[2]]==nodes[[3]])

allnodes <- PnetPnodes(tnet)
stopifnot(length(allnodes)==0)

## Need to mark nodes a Pnodes before they will be seen.
nodes <- lapply(nodes,as.Pnode)
allnodes <- PnetPnodes(tnet)
stopifnot(length(allnodes)==3)
stopifnot(any(sapply(allnodes,"==",nodeA))) ## NodeA in there somewhere.

## Not run:
## Safe way to preserve node and network objects across R sessions.
tnet <- WriteNetworks(tnet,"Tnet.neta")
q(save="yes")
# R
library(RNetica)
sess <- NeticaSession()
startSession(sess)
tnet <- ReadNetworks(tnet,sess)
nodes <- NetworkFindNodes(tnet,as.character(nodes))

## End(Not run)
DeleteNetwork(tnet)
```

PnetHub *Returns the name of the hub net if this is a spoke net.*

Usage

```
PnetHub(net)
```

Arguments

net

PnetName *Gets or Sets the name of a Netica network.*

Description

Gets or sets the name of the network. Names must conform to the IDname rules

Usage

```
PnetName(net)
PnetName(net) <- value
```

Arguments

net	A NeticaBN object which links to the network.
value	A character scalar containing the new name.

Details

Network names must conform to the IDname rules for Netica identifiers. Trying to set the network to a name that does not conform to the rules will produce an error, as will trying to set the network name to a name that corresponds to another different network.

The PnetTitle() function provides another way to name a network which is not subject to the IDname restrictions.

Value

The name of the network as a character vector of length 1.

The setter method returns the modified object.

Note

NeticaBN objects are internally implemented as character vectors giving the name of the network. If a network is renamed, then it is possible that R will hold onto an old reference that still using the old name. In this case, PnetName(net) will give the correct name, and GetNamedNets(PnetName(net)) will return a reference to a corrected object.

Author(s)

Russell Almond

References

http://norsys.com/onLineAPIManual/index.html: GetNetName_bn(), SetNetName_bn()

See Also

CreateNetwork(), NeticaBN, GetNamedNetworks(), PnetTitle()

Examples

```
sess <- NeticaSession()
startSession(sess)

net <- CreateNetwork("funNet",sess)
netcached <- net

stopifnot(PnetName(net)=="funNet")

PnetName(net)<-"SomethingElse"
stopifnot(PnetName(net)=="SomethingElse")

stopifnot(PnetName(net)==PnetName(netcached))

DeleteNetwork(net)
```

PnetPathname *Returns the path associated with a network.*

Usage

```
PnetPathname(net)
```

Arguments

net

PnetSerialize *Methods for (un)serializing a Netica Network*

Description

Methods for functions PnetSerialize and unserializePnet in package **Peanut**, which serialize NeticaBN objects. Note that in this case, the factory is the NeticaSession object. These methods assume that there is a global variable with the name of the session object which points to the Netica session.

Methods

PnetSerialize, signature(net = "NeticaBN") Returns a vector with three components. The name field is the name of the network. The data component is a raw vector produced by calling serialize(...,NULL) on the output of a WriteNetworks operation. The factory component is the name of the NeticaSession object. Note that the PnetUnserialize function assumes that there is a global variable with name given by the factory argument which contains an appropriate NeticaSession object for the restoration.

unserializePnet, signature(factory = "NeticaSession") This method reverses the previous one. In particular, it applies ReadNetworks to the serialized object.

Examples

```
## Need to create session whose name is is the same a the symbol it is
## stored in.
MySession <- NeticaSession(SessionName="MySession")
startSession(MySession)

irt5 <- ReadNetworks(file.path(library(help="RNetica")$path,
                           "sampleNets","IRT5.dne"), session=MySession)
NetworkAllNodes(irt5)
CompileNetwork(irt5) ## Ready to enter findings
NodeFinding(irt5$nodes$Item_1) <- "Right"
NodeFinding(irt5$nodes$Item_2) <- "Wrong"

## Serialize the network
irt5.ser <- PnetSerialize(irt5)
stopifnot (irt5.ser$name=="IRT5",irt5.ser$factory=="MySession")

NodeFinding(irt5$nodes$Item_3) <- "Right"


## now revert by unserializing.
irt5 <- PnetUnserialize(irt5.ser)
NetworkAllNodes(irt5)
stopifnot(NodeFinding(irt5$nodes$Item_1)=="Right",
          NodeFinding(irt5$nodes$Item_2)=="Wrong",
          NodeFinding(irt5$nodes$Item_3)=="@NO FINDING")

DeleteNetwork(irt5)
stopSession(MySession)
```

PnetTitle *Gets the title or comments associated with a Netica network.*

Description

The title is a longer name for a network which is not subject to the Netica IDname restrictions. The comment is a free form text associated with a network.

Usage

```
PnetTitle(net)
PnetTitle(net) <- value
PnetDescription(net)
PnetDescription(net) <- value
```

Arguments

net	A NeticaBN object.
value	A character object giving the new title or comment.

Details

The title is meant to be a human readable alternative to the name, which is not limited to the IDname restrictions. The title also affects how the network is displayed in the Netica GUI.

The comment is any text the user chooses to attach to the network. If value has length greater than 1, the vector is collapsed into a long string with newlines separating the components.

Value

A character vector of length 1 providing the title or comment.

Author(s)

Russell Almond

References

http://norsys.com/onLineAPIManual/index.html: GetNetTitle_bn(), SetNetTitle_bn(), GetNetComments_bn(), SetNetComments_bn()

See Also

NeticaBN, NetworkName()

Examples

```
sess <- NeticaSession()
startSession(sess)

firstNet <- CreateNetwork("firstNet",sess)

PnetTitle(firstNet) <- "My First Bayesian Network"
stopifnot(PnetTitle(firstNet)=="My First Bayesian Network")

now <- date()
NetworkComment(firstNet)<-c("Network created on",now)
## Print here escapes the newline, so is harder to read
cat(NetworkComment(firstNet),"\n")
stopifnot(NetworkComment(firstNet) ==
```

```
  paste(c("Network created on",now),collapse="\n"))


DeleteNetwork(firstNet)
```

PnodeLabels *Accesses labels associated with a Pnode*

Usage

```
PnodeLabels(node)
```

Arguments

node

PnodeParentTvals.NeticaNode

Fetches a list of numeric variables corresponding to parent states

---

Description

In constructing a conditional probability table using the discrete partial credit framework (see calcDPCTable), each state of each parent variable is mapped onto a real value called the effective theta. The PnodeParentTvals method for Netica nodes returns the result of applying NodeLevels to each of the nodes in NodeParents(node).

Usage

```
## S3 method for class 'NeticaNode'
PnodeParentTvals(node)
```

Arguments

node	A Pnode which is also a NeticaNode.

Details

While the best practices for assigning values to the states of the parent nodes is probably to assign equal spaced values (using the function effectiveThetas for this purpose), this method needs to retain some flexibility for other possibilities. However, in general, the best choice should depend on the meaning of the parent variable, and the same values should be used everywhere the parent variable occurs.

Netica already provides the NodeLevels function which allows the states of a NeticaNode to be associated with numeric values. This method merely gathers them together. The method assumes that all of the parent variables have had their NodeLevels set and will generate an error if that is not true.

Value

PnodeParentTvals(node) should return a list corresponding to the parents of node, and each element should be a numeric vector corresponding to the states of the appropriate parent variable. If there are no parent variables, this will be a list of no elements.

Note

The implementation is merely: lapply(NodeParents(node), NodeLevels).

Author(s)

Russell Almond

References

Almond, R. G. (2015) An IRT-based Parameterization for Conditional Probability Tables. Paper presented at the 2015 Bayesian Application Workshop at the Uncertainty in Artificial Intelligence Conference.

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Williamson, D.M. and Yan, D. (2015) *Bayesian Networks in Educational Assessment.* Springer. Chapter 8.

See Also

Pnode.NeticaNode, Pnode, effectiveThetas, BuildTable.NeticaNode, maxCPTParam.NeticaNode

Examples

```
sess <- NeticaSession()
startSession(sess)
tNet <- CreateNetwork("TestNet", session=sess)

theta1 <- NewDiscreteNode(tNet,"theta1",
                          c("VH","High","Mid","Low","VL"))
## This next function sets the effective thetas for theta1
NodeLevels(theta1) <- effectiveThetas(NodeNumStates(theta1))
NodeProbs(theta1) <- rep(1/NodeNumStates(theta1),NodeNumStates(theta1))
theta2 <- NewDiscreteNode(tNet,"theta2",
                          c("High","Mid","Low"))
## This next function sets the effective thetas for theta2
NodeLevels(theta2) <- effectiveThetas(NodeNumStates(theta2))
NodeProbs(theta2) <- rep(1/NodeNumStates(theta2),NodeNumStates(theta2))

partial3 <- NewDiscreteNode(tNet,"partial3",
                            c("FullCredit","PartialCredit","NoCredit"))
NodeParents(partial3) <- list(theta1,theta2)

## Usual way to set rules is in constructor
partial3 <- Pnode(partial3,rules="Compensatory", link="partialCredit")

PnodeParentTvals(partial3)
do.call("expand.grid",PnodeParentTvals(partial3))
```

```
DeleteNetwork(tNet)
stopSession(sess)
```

PnodeStateTitles *Accesses titles and descriptions of a Pnode's states.*

Usage

```
PnodeStateTitles(node)
PnodeStateDescriptions(node)
```

Arguments

node

PnodeStateValues *Accesses numeric values associated with a state.*

Usage

```
PnodeStateValues(node)
```

Arguments

node

PnodeTitle *Access the title or description of a node.*

Usage

```
PnodeTitle(node)
PnodeDescription(node)
```

Arguments

node

WarehouseData *Fetches the manifest data asscoiated with an item in the manifest.*

Usage

```
WarehouseData(warehouse, name)
```

Arguments

warehouse

name

WarehouseFetch *Fetches an item from the Warehouse if it exsits.*

Usage

```
WarehouseFetch(warehouse, name)
```

Arguments

warehouse

name

WarehouseFree *Removes an item from the warehouse inventory.*

Usage

```
WarehouseFree(warehouse, name)
```

Arguments

warehouse

name

WarehouseMake *Makes a new item of the specified type using manifest description.*

Usage

```
WarehouseMake(warehouse, name)
```

Arguments

```
warehouse
name
```

WarehouseManifest *Accesses the manifest for a warehouse.*

Usage

```
WarehouseManifest(warehouse)
WarehouseManifest(warehouse) <- value
```

Arguments

```
warehouse
value
```

Index