Package ‘PNetica’

July 5, 2015

Version 0.1-3

Date 2015/07/05

Title Parameterized Bayesian Networks Netica Interface

Author Russell Almond

Maintainer Russell Almond <ralmond@fsu.edu>

Depends R (>= 3.0), RNetica (>= 0.4), CPTtools (>= 0.3), Peanut (>= 0.1)

Description This package provides RNetica implementation of Peanut interface.

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

R topics documented:

PNetica-package 1
BuildTable.NeticaNode 5
calcExpTables.NeticaBN 6
calcPnetLLike.NeticaBN 8
maxCPTParam.NeticaNode 10
Pnet.NeticaBN 12
Pnode.NeticaNode 14
PnodeParentTvals.NeticaNode 17

Index **20**

`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 implemenations 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 Implemenation 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 ont 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 implemenation independent Bayes net utilities.

Examples

```
## Building CPTs
tNet <- CreateNetwork("TestNet")


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(paste(library(help="PNetica")$path,
                           "testnets","IRT10.2PL.base.dne",
                           sep=.Platform$file.sep))
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 <- paste(library(help="PNetica")$path,
                           "testdat","IRT10.2PL.200.items.cas",
                           sep=.Platform$file.sep)
## 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)
```

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 [.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 proability 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

Examples

```
## Not run:
## The method is currently defined as
function (node)
{
    node[] <- calcDPCFrame(ParentStates(node), NodeStates(node),
        PnodeLnAlphas(node), PnodeBetas(node), PnodeRules(node),
        PnodeLink(node), PnodeLinkScale(node), PnodeQ(node),
        PnodeParentTvals(node))
    NodeExperience(node) <- GetPriorWeight(node)
    invisible(node)
  }

## End(Not run)
```

---

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 containg 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 spead up convergence, the parameter is left open for experimentation.) The tolerance is largely irrelvant 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

```
irt10.base <- ReadNetworks(paste(library(help="PNetica")$path,
                           "testnets","IRT10.2PL.base.dne",
                           sep=.Platform$file.sep))
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 <- paste(library(help="PNetica")$path,
                           "testdat","IRT10.2PL.200.items.cas",
                           sep=.Platform$file.sep)
## 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)
```

---

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

```
irt10.base <- ReadNetworks(paste(library(help="PNetica")$path,
                           "testnets","IRT10.2PL.base.dne",
                           sep=.Platform$file.sep))
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 <- paste(library(help="PNetica")$path,
                           "testdat","IRT10.2PL.200.items.cas",
                           sep=.Platform$file.sep)
## 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)
```

`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 applicatoin. The tolerance is largely irrelvant (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

Examples

```
## The function is currently defined as
function (node, Mstepit = 5, tol = sqrt(.Machine$double.eps))
{
    counts <- sweep(node[[]], 1, NodeExperience(node), "*")
    withCallingHandlers(est <- mapDPC(counts, ParentStates(node),
        NodeStates(node), PnodeLnAlphas(node), PnodeBetas(node),
        PnodeRules(node), PnodeLink(node), PnodeLinkScale(node),
        PnodeQ(node), control = list(reltol = tol, maxit = Mstepit)),
        warning = muffler)
    PnodeLnAlphas(node) <- est$lnAlphas
```

```
    PnodeBetas(node) <- est$betas
    PnodeLinkScale(node) <- est$linkScale
    invisible(node)
  }
```

Pnet.NeticaBN *RNetica implemenation of the Pnet class*

Description

This documentation file describes the use of a NeticaBN object as a Pnet. See details for descriptions of the methods.

Usage

```
## S3 method for class 'NeticaBN'
as.Pnet(x)
## S3 method for class 'NeticaBN'
PnetPriorWeight(net)
## S3 replacement method for class 'NeticaBN'
PnetPriorWeight(net) <- value
## S3 method for class 'NeticaBN'
PnetPnodes(net)
## S3 replacement method for class 'NeticaBN'
PnetPnodes(net) <- value
```

Arguments

x	A NeticaBN object to be converted to a Pnet object.
net	A NeticaBN object to be manipulated (should also be a Pnet, but this is not checked.
value	In the case of PnetPriorWeight(net) a numeric scalar giving the default weight for the prior. In the case of PnetPnodes(net) a list of NeticaNode objects belonging to net.

Details

The Pnet object model is added to the NeticaNode class using two approaches. First, the PnetPriorWeight method uses the NetworkUserObj to serialize the prior weights and store them in one of the network's user fields ("priorWeight"). Second the PnetPnodes method uses node sets (NetworkNodesInSet) to mark the Pnodes in the graph (the node set is called "pnodes").

In addition to the "pnodes" node set, the PNetica implementation of Pnet requires an additional node set called "onodes". These correspond to the nodes present in the cases argument to GEMfit and related methods.

The as.Pnet.NeticaBN method merely adds "Pnet" to class(net). The default method of Pnet calls as.Pnet and also sets default values for the prior weight and pnodes fields. This is the recommended approach for creating new Pnet objects.

The user fields and node sets are saved and restored when a Netica network is saved to a file. (This is true for the user fields in the Pnode objects as well.) Calling as.Pnet on the newly restored network should correct the class field without overwriting the restored fields. (Generally, as.Pnode should be called on all of the Pnodes as well.)

Value

The method as.Pnet.NeticaBN returns an object of class c("Pnet", "NeticaBN"). The descriptions of the returns for the other methods can be found in the description of their generic functions.

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, NeticaBN, Pnode, Pnode.NeticaNode, PnetPriorWeight, PnetPnodes, NetworkUserObj, NetworkNodesInSet

Examples

```
#########################################
## Create network structure using RNetica calls
IRT10.2PL <- CreateNetwork("IRT10_2PL")

theta <- NewDiscreteNode(IRT10.2PL,"theta",
                         c("VH","High","Mid","Low","VL"))
NodeLevels(theta) <- effectiveThetas(NodeNumStates(theta))
NodeProbs(theta) <- rep(1/NodeNumStates(theta),NodeNumStates(theta))

J <- 10 ## Number of items
items <- NewDiscreteNode(IRT10.2PL,paste("item",1:J,sep=""),
                         c("Correct","Incorrect"))
for (j in 1:J) {
  NodeParents(items[[j]]) <- list(theta)
  NodeLevels(items[[j]]) <- c(1,0)
  NodeSets(items[[j]]) <- c("observables")
}

## Convert into a Pnet
IRT10.2PL <- Pnet(IRT10.2PL,priorWeight=10,pnodes=items)

## Convert nodes to Pnodes
for (j in 1:J) {
  items[[j]] <- Pnode(items[[j]])
}
```

```
DeleteNetwork(IRT10.2PL)

####################################
## Restore a network from a file.
irt10.base <- ReadNetworks(paste(library(help="PNetica")$path,
                           "testnets","IRT10.2PL.base.dne",
                           sep=.Platform$file.sep))
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]])
}

## Need to set onodes field by hand, using RNetica functions
casepath <- paste(library(help="PNetica")$path,
                           "testdat","IRT10.2PL.200.items.cas",
                           sep=.Platform$file.sep)
## Record which nodes in the casefile we should pay attention to
NetworkNodesInSet(irt10.base,"onodes") <-
   NetworkNodesInSet(irt10.base,"observables")



DeleteNetwork(irt10.base)
```

Pnode.NeticaNode *RNetica implemenation of the Pnode class*

Description

This documentation file describes the use of a NeticaNode object as a Pnode. See details for descriptions of the methods.

Usage

```
## S3 method for class 'NeticaNode'
 Pnode(node, lnAlphas, betas, rules="Compensatory",
                              link="partialCredit", Q=TRUE, linkScale=NULL,
                              priorWeight=NULL)
## S3 method for class 'NeticaNode'
as.Pnode(x)
## S3 method for class 'NeticaNode'
PnodeNet(node)
## S3 method for class 'NeticaNode'
```

```
PnodePriorWeight(node)
## S3 method for class 'NeticaNode'
PnodeQ(node)
## S3 method for class 'NeticaNode'
PnodeRules(node)
## S3 method for class 'NeticaNode'
PnodeLink(node)
## S3 method for class 'NeticaNode'
PnodeLnAlphas(node)
## S3 method for class 'NeticaNode'
PnodeBetas(node)
## S3 method for class 'NeticaNode'
PnodeLinkScale(node)
```

Arguments

x	A NeticaNode object to be converted to a Pnode object.
node	A NeticaNode object to be manipulated (should also be a Pnode, but this is not checked.
lnAlphas	A numeric vector of list of numeric vectors giving the log slope parameters. See PnodeLnAlphas for a description of this parameter. If missing, the constructor will try to create a pattern of zero values appropriate to the rules argument and the number of parent variables.
betas	A numeric vector of list of numeric vectors giving the intercept parameters. See PnodeBetas for a description of this parameter. If missing, the constructor will try to create a pattern of zero values appropriate to the rules argument and the number of parent variables.
rules	The combination rule or a list of combination rules. These should either be names of functions or function objects. See PnodeRules for a description of this argument.
link	The name of the link function or the link function itself. See PnodeLink for a description of the link function.
Q	A logical matrix or the constant TRUE (indicaticating that the Q-matrix should be a matrix of TRUEs). See PnodeQ for a description of this parameter.
linkScale	A numeric vector of link scale parameters or NULL if scale parameters are not needed for the chosen link function. See PnodeLinkScale for a description of this parameter.
priorWeight	A numeric vector of weights given to the prior parameter values for each row of the conditional probability table when learning from data (or a scalar if all rows have equal prior weight). See PnodePriorWeight for a description of this parameter.

Details

The Pnode object model is added to the NeticaNode class using the NodeUserObj method to serialize the value and store them in one of the node's user fields. Note that most of the functions described above have setter as well as getter methods defined (see under the corresponding arguments for descriptions).

The as.Pnode.NeticaNode method merely adds “Pnode” to class(net). The NeticaNode method of Pnode calls as.Pnode and also sets default values for various pnode fields. This is the recommended approach for creating new Pnode objects. 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).

The user fields are saved and restored when a Netica network is saved to a file. (This is true for the user fields in the Pnet objects as well.) Calling as.Pnode on the appropriate nodes of the newly restored network should correct the class field without overwriting the restored fields. (Generally, as.Pnet should be called on the Pnet as well.)

Note that the PnodeParentTvals.NeticaNode method assumes that the parent variables have had numeric values assigned to their states using the NodeLevels function.

Value

The method as.Pnode.NeticaNode returns an object of class c("Pnode", "NeticaNode"). The descriptions of the returns for the other methods can be found in the description of their generic functions.

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, Pnet, Pnet.NeticaBN, PnodePriorWeight, PnodeNet, PnodeQ, PnodeRules, PnodeLink, PnodeLnAlphas, PnodeBetas, PnodeLinkScale, PnodeParentTvals.NeticaNode, NetworkUserObj,

Examples

```
##########################################
## Create network structure using RNetica calls
IRT10.2PL <- CreateNetwork("IRT10_2PL")

theta <- NewDiscreteNode(IRT10.2PL,"theta",
                         c("VH","High","Mid","Low","VL"))
NodeLevels(theta) <- effectiveThetas(NodeNumStates(theta))
NodeProbs(theta) <- rep(1/NodeNumStates(theta),NodeNumStates(theta))

J <- 10 ## Number of items
items <- NewDiscreteNode(IRT10.2PL,paste("item",1:J,sep=""),
                         c("Correct","Incorrect"))
```

```
for (j in 1:J) {
  NodeParents(items[[j]]) <- list(theta)
  NodeLevels(items[[j]]) <- c(1,0)
  NodeSets(items[[j]]) <- c("observables")
}

## Convert into a Pnode
IRT10.2PL <- Pnet(IRT10.2PL,priorWeight=10,pnodes=items)

## Convert nodes to Pnodes
for (j in 1:J) {
  items[[j]] <- Pnode(items[[j]])
}

DeleteNetwork(IRT10.2PL)

####################################
## Restore a network from a file.
irt10.base <- ReadNetworks(paste(library(help="PNetica")$path,
                           "testnets","IRT10.2PL.base.dne",
                           sep=.Platform$file.sep))
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]])
}

DeleteNetwork(irt10.base)
```

PnodeParentTvals.NeticaNode
Fetches a list of numeric variables corresponding to parent states

---

Description

In constructing a conditional probability table using the discrete parital 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

```
tNet <- CreateNetwork("TestNet")

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)
```

Index

∗Topic **attrib**
PnodeParentTvals.NeticaNode, 17
∗Topic **distribution**
BuildTable.NeticaNode, 5
∗Topic **graphs**
calcPnetLLike.NeticaBN, 8
Pnet.NeticaBN, 12
PNetica-package, 1
Pnode.NeticaNode, 14
∗Topic **manip**
calcExpTables.NeticaBN, 6
maxCPTParam.NeticaNode, 10
Pnet.NeticaBN, 12
Pnode.NeticaNode, 14
∗Topic **package**
PNetica-package, 1
[.NeticaNode, *5, 6*

as.Pnet, *2*, *16*
as.Pnet.NeticaBN *(*Pnet.NeticaBN*)*, 12
as.Pnode, *2*, *13*
as.Pnode.NeticaNode *(*Pnode.NeticaNode*)*, 14

BuildAllTables, *9*
BuildTable, *7*
BuildTable.NeticaNode, 5, *18*

calcDPCFrame, *5*
calcDPCTable, *5, 6*, *17*
calcExpTables, *8, 9*
calcExpTables.NeticaBN, *2*, 6
calcPnetLLike, *8*
calcPnetLLike.NeticaBN, *2*, 8
CompileNetwork, *2*
CPTtools, *3*

effectiveThetas, *18*

FindingsProbability, *9*

GEMfit, *2*, *7–9*, *11, 12*
GetPriorWeight, *5–7*

LearnCPTs, *6–8*, *10*

mapDPC, *10, 11*
maxAllTableParams, *8, 9*, *11*
maxCPTParam, *10*
maxCPTParam.NeticaNode, 10, *18*

NeticaBN, *2*, *12, 13*
NeticaNode, *2*, *10*, *12*, *14–16*, *18*
NetworkNodesInSet, *2*, *7–9*, *12, 13*
NetworkUserObj, *2*, *12, 13*, *16*
NodeExperience, *5–7*, *11*
NodeLevels, *16–18*
NodeParents, *2*, *5*, *16, 17*
NodeUserObj, *2*, *15*

Peanut, *2, 3*
Pnet, *2*, *8, 9*, *12, 13*, *16*
Pnet.default, *2*
Pnet.NeticaBN, *2*, *7–9*, 12, *16*
PNetica *(*PNetica-package*)*, 1
PNetica-package, 1
PnetPnodes, *11–13*
PnetPnodes.NeticaBN, *2*
PnetPnodes.NeticaBN *(*Pnet.NeticaBN*)*, 12
PnetPnodes<-.NeticaBN *(*Pnet.NeticaBN*)*, 12
PnetPriorWeight, *5*, *12, 13*
PnetPriorWeight.NeticaBN, *2*
PnetPriorWeight.NeticaBN *(*Pnet.NeticaBN*)*, 12
PnetPriorWeight<-.NeticaBN *(*Pnet.NeticaBN*)*, 12
Pnode, *2*, *5, 6*, *10–16*, *18*
Pnode.NeticaNode, *2*, *5, 6*, *11*, *13*, 14, *18*
PnodeAlphas, *6*
PnodeBetas, *2*, *6*, *11*, *15, 16*

PnodeBetas.NeticaNode
(Pnode.NeticaNode), 14
PnodeLink, *6*, *15, 16*
PnodeLink.NeticaNode
(Pnode.NeticaNode), 14
PnodeLinkScale, *6*, *15, 16*
PnodeLinkScale.NeticaNode
(Pnode.NeticaNode), 14
PnodeLnAlphas, *2*, *6*, *11*, *15, 16*
PnodeLnAlphas.NeticaNode
(Pnode.NeticaNode), 14
PnodeNet, *16*
PnodeNet.NeticaNode (Pnode.NeticaNode),
14
PnodeParentTvals.NeticaNode, *16*, 17
PnodePriorWeight, *5, 6*, *15, 16*
PnodePriorWeight.NeticaNode
(Pnode.NeticaNode), 14
PnodeQ, *6*, *15, 16*
PnodeQ.NeticaNode (Pnode.NeticaNode), 14
PnodeRules, *6*, *15, 16*
PnodeRules.NeticaNode
(Pnode.NeticaNode), 14

RNetica, *2, 3*

write.CaseFile, *2*, *7–9*
WriteNetworks, *2*