Package 'EABN'

April 3, 2020

Version 0.5-2

Date 2020/02/15

Title Evidence Accumulation Bayes Net Engine

Author Russell Almond

Maintainer Russell Almond <ralmond@fsu.edu>

Depends R (>= 3.0), methods, mongolite, jsonlite, Peanut (>= 0.8), Proc4 (>= 0.3), futile.logger

Description Extracts observables from a sequence of events.

License Artistic-2.0

URL http://pluto.coe.fsu.edu/Proc4

Collate Evidence.R StudentRec.R EAEngine.R EAEngineMongo.R EAEngineNDB.R EngineGears.R

R topics documented:

`accumulateEvidence` *Merge evidence from an evidence set with the student record.*

Description

The function `accumulateEvidence` combines the evidence in the `EvidenceSet` with the exiting beliefs in the `StudentRecord`, updating the student record. The function `handleEvidence` is a wrapper around this which takes care of finding and updating the evidence sets.

Usage

```
accumulateEvidence(eng, rec, evidMess, debug = 0)
handleEvidence(eng, evidMess, srser = NULL, debug = 0)
```

Arguments

`eng`	The `BNEngine` which controls the process.
`rec`	The `StudentRecord` which will be updated.
`srser`	A serialized version of the student record for the no-database version of the model.
`evidMess`	An `EvidenceSet` which has the evidence to be incorporated.
`debug`	An integer flag. If greater than 1, then `recover()` will be called at strategic places during the processing to allow inspection of the process.

Details

The function accumulateEvidence performs the following steps:

1. Update the student record to associate it with the new evidence (updateRecord).
2. Update the student model with the new evidence (updateSM).
3. Update the statistics for the new student model (updateStats).
4. Update the history for the new evidence (updateHist).
5. Announce the availablity of new statistics (announceStats).
6. Save the updated student record (saveSR).

The function handleEvidence is a wrapper around accumulateEvidence which finds the student record. Note for BNEngineNDB, it is expected that the student record will be passed in as a serialized object (see getRecordForUser). It performs the following steps:

1. Fetch the student record for the uid associated with the evidence set (getRecordForUser).
2. Mark the evidence as belonging to this student record (logEvidence).
3. Update the record by calling accumulateEvidence.
4. Mark the evidence as processed (markProcessed).

If an error is encountered, then the error message is added to the evidence set.

Value

The modified StudentRecord which was just processed. If an error occurs during the call to accumulateEvidence both function will return an object of class try-error instead of the student record.

Logging, Error Handling and Debugging

The functions handleEvidence, accumulateEvidence and many of the functions they call use the flog.logger protocol. The default logging level of INFO will give messages in response to the announcements and warnings when an error occur. The DEBUG and TRACE levels will provide more information about the details of the update algorithm.

The body of accumulateEvidence is wrapped in withFlogging which captures and logs errors. This function returns an object of class try-error when an error occurs. Although handleEvidence does not use the flogging error handler, it will still pass on the try-error if one is generated.

The debug argument can be used to pause execution. Basically, recover() will be called between every step. This only happens in interactive mode as it just does not make sense in batch model.

Author(s)

Russell Almond

References

Almond, Mislevy, Steinberg, Yan and Williamson (2015). *Bayesian Networks in Educational Assessment*. Springer. Especially Chapters 5 and 13.

See Also

Classes: BNEngine BNEngineMongo, BNEngineNDB StudentRecord, EvidenceSet

Main Loop Functions: mainLoop, getRecordForUser, logEvidence, updateRecord, updateSM, updateStats, updateHist, announceStats, saveSR

Examples

```
## Requires database setup, also PNetica
library(RNetica)  ## Must load to setup Netica DLL
app <- "ecd://epls.coe.fsu.edu/EITest"
sess <- RNetica::NeticaSession()
RNetica::startSession(sess)

config.dir <- file.path(library(help="Peanut")$path, "auxdata")
net.dir <- file.path(library(help="PNetica")$path,"testnets")

netman <- read.csv(file.path(config.dir, "Mini-PP-Nets.csv"),
                   row.names=1, stringsAsFactors=FALSE)
stattab <- read.csv(file.path(config.dir, "Mini-PP-Statistics.csv"),
                   as.is=TRUE)

Nethouse <- PNetica::BNWarehouse(netman,session=sess,
             address=net.dir)

cl <- new("CaptureListener")
listeners <- list("cl"=cl)

ls <- ListenerSet(sender= paste("EAEngine[",app,"]"),
                  dburi="", listeners=listeners)

eng <- BNEngineNDB(app=app,warehouse=Nethouse,
                     listenerSet=ls,manifest=netman,
                     profModel="miniPP_CM",
                     histNodes="Physics",
                     statmat=stattab,
                     activeTest="EAActive.txt")

## Standard initialization methods.
loadManifest(eng,netman)
eng$setHistNodes("Physics")
configStats(eng,stattab)
setupDefaultSR(eng)

sr0 <- getRecordForUser(eng,"S1")

eap0 <- stat(sr0,"Physics_EAP")

e1 <- EvidenceSet(uid="S1",app="Test",context="PPcompEM",
                  obs=list("CompensatoryObs"="Right"))
```

```
e1 <- logEvidence(eng,sr0,e1)
sr1 <- accumulateEvidence(eng,sr0,e1)
stopifnot(m_id(sr1)!=m_id(sr0),sr1@prev_id==m_id(sr0))
stopifnot(seqno(sr1)==1L, seqno(e1)==1L)

eap1 <- stat(sr1,"Physics_EAP")
stopifnot(abs(eap1-eap0) > .001)
stopifnot(nrow(history(sr1,"Physcis"))==2L)

## handle Evidence.
sr1.ser <- as.json(sr1)
e2 <- EvidenceSet(uid="S2",app="Test",context="PPconjEM",
                  obs=list("ConjunctiveObs"="Wrong"))

sr2 <- handleEvidence(eng,e2,fromJSON(sr1.ser))
eap2 <- stat(sr2,"Physics_EAP")
stopifnot(uid(sr2)==uid(sr1),
          m_id(sr1)==sr2@prev_id,
          nrow(history(sr2,"Physics"))==3L,
          abs(eap1-eap2) > .001)

## <<HERE>> Need test with Mongo engine.
```

BNEngine-class *Class* "BNEngine"

Description

A generic engine for handling evidence messages (EvidenceSet objects).

Details

This is the basic class for running the evidence accumulation process. This is actually an abstract class, there are two subclasses: BNEngineMongo, which uses the Mongo database to store student records and as a message queue, and BNEngineNDB, which operates without a database. Note that the BNEngine constructor generates an error.

The following functions form the core of the Engine Protocol:

loadManifest This loads the network manifest for the PnetWarehouse.

setupDefaultSR Sets up the default Student Record (used for creating new student records)

configStats Configures the statistics that are reported in the main loop.

baselineHist Sets up the baselines for histories.

mainLoop This runs through a queue of messages, handling the evidence.

handleEvidence Handles evidence from one scoring context and one user.

accumulateEvidence Does the actual work of processing the evidence.

getRecordForUser Fetches the student record for a user, essentially a call to getSR.

logEvidence Logs the evidence as part of the student record.

updateSM Updates the student model for the new evidence.

updateStats Calculates new statistics for the revised student model.

updateHist Updates the history for the revised student model.

announceStats Updates other processes about the existance of updated statistics.

Extends

All reference classes extend and inherit methods from "envRefClass".

Methods

app `signature(x = "BNEngine")`: Returns the guid identifying the application that this engine is handling.

notifyListeners `signature(sender = "BNEngine")`: Notifies other processes that student records have been updated.

fetchNextEvidence `signature(eng = "BNEngine")`: Returns the next unprocessed EvidenceSet in the queue.

markProcessed `signature(eng = "BNEngine", eve = "EvidenceSet")`: marks the eve argument as processed.

Fields

app: Object of class character giving an globally unique identifier for the application

srs: Object of class StudentRecordSet of NULL giving the student record set for the application.

profModel: Object of class character giving the name of the proficiency model (for the default student record) in the warehouse manifest.

listenerSet: Object of class ListenerSet giving a set of listeners who will listen for new statistics.

statistics: Object of class list containing Statistic objects to be run on every update cycle.

histNodes: Object of class character giving the names of the nodes in the proficiency model whose history will be recorded.

warehouseObj: Object of class PnetWarehouse which stores the Bayes nets, both evidence models and student models are stored here.

waittime: Object of class numeric giving the time in seconds the main event loop should wait before checking again for messages.

processN: Object of class numeric giving the number of times that the main loop should run before stopping. If Inf, then the main loop will run without stopping.

Class-Based Methods

`isActivated()`: This function checks the database to see whether or not the flag is set to cause the process to stop.

`setHistNodes(nodenames)`: Sets the names of the history nodes. Note this should be called before the call to `baselineHist` or the history nodes will not be set properly in the default student record.

`fetchNextEvidence()`: Fetches the next evidence set to be handled.

`setError(mess, e)`: Adds an error flag to an evidence set that generated an error.

`getHistNodes()`: Retrieves the history nodes.

`saveStats(statmat)`: Updates the set of statitics associated with this engine.

`studentRecords()`: Fetches the `StudentRecordSet` associated with the engine. Note: This method should be called instead of the raw field as it will initialize the field if it is not set up yet.

`fetchStats()`: Fetches statistic objects from the database.

`stats()`: Returns the set of `Statistic` objects associate with the engine.

`activate()`: Sets the flag to indicate that the process is running.

`fetchManifest()`: Fetches the network manifest from the database.

`setManifest(manifest)`: Sets the manifest for the `PnetWarehouse`.

`saveManifest(manifest)`: Saves the network manifest to the database.

`show()`: Providse a printed representation of the database.

`setProcessed(mess)`: Sets an evidence set message as processed.

`warehouse()`: Returns the `PnetWarehouse` associated with this engine. Again, this function should be called in preference to directly accessing the field as it forces initialization when necessary.

`evidenceSets()`: A reference to the collection of evidence sets.

Author(s)

Russell Almond

References

Almond, Mislevy, Steinberg, Yan and Williamson (2015). *Bayesian Networks in Educational Assessment*. Springer. Especially Chapter 13.

See Also

Subclasses: `BNEngineMongo`, `BNEngineNDB`

Constituent parts: `StudentRecordSet`, `PnetWarehouse`

Setup Functions: `loadManifest`, `setupDefaultSR`, `configStats`, `baselineHist`,

Main Loop Functions: `mainLoop`, `handleEvidence`, `getRecordForUser`, `logEvidence`, `accumulateEvidence`, `updateRecord`, `updateSM`, `updateStats`, `updateHist`, `announceStats`,

Examples

```
showClass("BNEngine")
```

BNEngineMongo *Creates a Bayes Net Engine attached to a Mongo database.*

Description

The BNEngineMongo is a BNEngine which is attached to a MongoDB-class database, which hold both the queue and the StudentRecordSet.

Usage

```
BNEngineMongo(app = "default", warehouse, listenerSet = NULL,
dburi = "mongodb://localhost", dbname = "EARecords", processN = Inf,
admindbname = "Proc4", waittime = 0.25, profModel = character(),
...)
```

Arguments

app	A character scalar giving the globally unique identifier for the application.
warehouse	A PnetWarehouse which stores the default student model and evidence models. (It will also store the student models.
listenerSet	A ListenerSet which contains the listeners for clients of the engine's messages.
dburi	A character scalar giving the login information for the mongo database. See makeDBuri.
dbname	The name for the EA database.
processN	The number of records to process before stopping. The default value Inf runs the process until the active flag is cleared.
admindbname	The name of the admin database used to check for shutdown requests.
waittime	The amout of time (in seconds) to wait before checking again for new evidence sets when the evidence set queue is empty.
profModel	The name of the proficiency model (its ID in the warehouse manifest).
...	Extra room in case we later think of more things we should add.

Details

This creates an uninitialized BNEngine, specifically a BNEngineMongo.

The makeDBuri function provides a useful shorthand for calculating the dburi field.

Value

An object of calls BNEngineMongo which is capable of scoring student models.

Note

The database connections are not created right away, so it is important to use the class-based functions, `manifestdb()`, `statdb()`, `evidenceSets()`, `histNodesdb()`, `studentRecords()`, and `admindb()` rather than accessing the fields directly.

Author(s)

Russell Almond

References

Almond, Mislevy, Steinberg, Yan and Williamson (2015). *Bayesian Networks in Educational Assessment*. Springer. Especially Chapter 13.

See Also

Classes: BNEngine, BNEngineNDB

Constituent parts: StudentRecordSet, PnetWarehouse ListenerSet

Setup Functions: loadManifest, setupDefaultSR, configStats, baselineHist,

Main Loop Functions: mainLoop, accumulateEvidence, handleEvidence, getRecordForUser, logEvidence, updateSM, updateStats, updateHist, announceStats,

Examples

```
## Requires database setup, also PNetica
library(RNetica)  ## Must load to setup Netica DLL
app <- "ecd://epls.coe.fsu.edu/EITest"
sess <- RNetica::NeticaSession()
RNetica::startSession(sess)

config.dir <- file.path(library(help="Peanut")$path, "auxdata")
net.dir <- file.path(library(help="PNetica")$path,"testnets")

netman <- read.csv(file.path(config.dir, "Mini-PP-Nets.csv"),
                   row.names=1, stringsAsFactors=FALSE)
stattab <- read.csv(file.path(config.dir, "Mini-PP-Statistics.csv"),
                   as.is=TRUE)

Nethouse <- PNetica::BNWarehouse(netman,session=sess,
             address=net.dir)

cl <- new("CaptureListener")
listeners <- list("cl"=cl)
ls <- ListenerSet(sender= paste("EAEngine[",basename(app),"]"),
                  dbname="EARecords", dburi=makeDBuri(host="localhost"),
                  listeners=listeners,
                  colname="Messages")
```

```
eng <- BNEngineMongo(app=app,warehouse=Nethouse,
                     listenerSet=ls,
                     dburi=makeDBuri(host="localhost"),
                     dbname="EARecords",profModel="miniPP_CM",
                     histNodes="Physics")

## Standard initialization methods.
loadManifest(eng,netman)
eng$setHistNodes("Physics")
configStats(eng,stattab)
setupDefaultSR(eng)
```

BNEngineMongo-class *Class* "BNEngineMongo"

Description

A Bayes net engine hooked to a Mongo database.

Extends

Class "BNEngine", directly.

All reference classes extend and inherit methods from "envRefClass".

Activation

When the mainLoop runs out of evidence sets to process, it checks the `is.active()` method of the engine. If this returns, false, it stops.

For the BNEngineMongo this checks the "AuthorizedApps" collection in the database to see if the current app is active. It returns the value of the `active` field for the app's record.

Fields

app: Object of class `character` giving an globally unique identifier for the application

srs: Object of class StudentRecordSet of `NULL` giving the student record set for the application.

profModel: Object of class `character` giving the name of the proficiency model (for the default student record) in the warehouse manifest.

listenerSet: Object of class `ListenerSet` giving a set of listeners who will listen for new statistics.

statistics: Object of class `list` containing Statistic objects to be run on every update cycle.

histNodes: Object of class `character` giving the names of the nodes in the proficiency model whose history will be recorded.

warehouseObj: Object of class PnetWarehouse which stores the Bayes nets, both evidence models and student models are stored here.

waittime: Object of class numeric giving the time in seconds the main event loop should wait before checking again for messages.

processN: Object of class numeric giving the number of times that the main loop should run before stopping. If Inf, then the main loop will run without stopping.

dburi: Object of class character giving the URI for the mongo database.

dbname: Object of class character giving the name of the database to be used.

manifestDB: Object of class MongoDB-class giving the collection used to store the manifest. This object may not be initialized so it should be accessed through the class-based function manifestdb().

evidenceDB: Object of class MongoDB-class accessing the evidence set collection. This object may not be initialized so it should be accessed through the class-based function evidenceSets().

statDB: Object of class MongoDB giving the statistics to use. This object may not be initialized so it should be accessed through the class-based function statdb().

histNodesDB: Object of class MongoDB giving the history nodes. This object may not be initialized so it should be accessed through the class-based function histNodesdb().

admindbname: Object of class character giving name admin (Proc4) database, used for various listeners and the is.active() method.

adminDB: Object of class MongoDB giving the link to the admin database. This object may not be initialized so it should be accessed through the class-based function admindb().

Methods

statdb(): Returns the database contianing the statistic objects.

studentRecords(): Returns the StudentRecordSet associated with this engine.

fetchStats(): Fetches the statistics marked in the database configuration.

activate(): Sets the field in the Proc4 database used to indicate that this application is active.

initialize(app, warehouse, listeners, username, password, host, port, dbname, P4dbname, profModel, initializes the class. Note that some initialization is done in the various XXXdb() functions, so these should be called instead of directly accessing the fields.

manifestdb(): Returns the MongoDB-class handle to the manifest information collection.

admindb(): Returns the MongoDB-class handle to the "AuthorizedApps" collection.

histNodesdb(): Returns the MongoDB-class handle to the hist nodes collection.

saveManifest(manifest): Saves the current PnetWarehouse manifest to the manifestdb() collection

fetchManifest(): Retrieves the saved manifest from the manifestdb() collection.

fetchNextEvidence(): Retrieves the next EvidenceSet from the evidenceSets() collection. Returns NULL if there are not unprocessed evidence sets.

saveStats(statmat): Saves the update statistic definitions to the statdb() collection.

setHistNodes(nodenames): Saves the history nodes to the histNodesdb() collection.

isActivated(): Checks to see if the active flag is set.

`setError(mess, e)`: Added an error message to an evidence set.

`evidenceSets()`: Returns a `MongoDB-class` handle to the collection/queue of evidence sets.

`getHistNodes()`: Fetches the history nodes from the `histNodesdb()` collection.

`show()`: Provides a printed representation of the engine.

The following methods are inherited (from the corresponding class): evidenceSets ("BNEngine"), getHistNodes ("BNEngine"), stats ("BNEngine"), setProcessed ("BNEngine"), setManifest ("BNEngine"), activate ("BNEngine"), isActivated ("BNEngine"), saveManifest ("BNEngine"), studentRecords ("BNEngine"), saveStats ("BNEngine"), fetchNextEvidence ("BNEngine"), warehouse ("BNEngine"), show ("BNEngine"), setHistNodes ("BNEngine"), setError ("BNEngine"), fetchManifest ("BNEngine"), fetchStats ("BNEngine")

Note

The database connections are not created right away, so it is important to use the class-based functions, `manifestdb()`, `statdb()`, `evidenceSets()`, `histNodesdb()`, `studentRecords()`, and `admindb()` rather than accessing the fields directly.

Author(s)

Russell Almond

References

Almond, Mislevy, Steinberg, Yan and Williamson (2015). *Bayesian Networks in Educational Assessment*. Springer. Especially Chapter 13.

See Also

Classes: `BNEngine`, `BNEngineNDB`

Constituent parts: `StudentRecordSet`, `PnetWarehouse`

Setup Functions: `loadManifest`, `setupDefaultSR`, `configStats`, `baselineHist`,

Main Loop Functions: `mainLoop`, `accumulateEvidence`, `handleEvidence`, `getRecordForUser`, `logEvidence`, `updateSM`, `updateStats`, `updateHist`, `announceStats`,

Examples

```
showClass("BNEngineMongo")
```

BNEngineNDB *Creates a Bayes net engine not attached to a database.*

Description

The BNEngineNDB is a BNEngine which is not attached to the database. In particular, it cannot store student records, so it cannot maintain state between scoring sessions without extenal help.

Usage

```
BNEngineNDB(app = "default", warehouse, listenerSet = NULL,
  manifest = data.frame(), processN = Inf, waittime = 0.25,
  profModel = character(), statmat = data.frame(),
  evidenceQueue = list(), activeTest =
  "EAActive.txt", ...)
```

Arguments

app	A character scalar giving the globally unique identifier for the application.
warehouse	A PnetWarehouse which stores the default student model and evidence models. (It will also store the student models.
listenerSet	A ListenerSet which contains the listeners for clients of the engine's messages.
manifest	A data frame providing a manifest for the PnetWarehouse.
processN	The number of records to process before stopping. The default value Inf runs the process until the active flag is cleared.
waittime	The amout of time (in seconds) to wait before checking again for new evidence sets when the evidence set queue is empty.
profModel	The name of the proficiency model (its ID in the warehouse manifest).
statmat	A data.frame describing the statistics. See configStats.
evidenceQueue	A list of EvidenceSet objects to be processed.
activeTest	The pathname for the file whose existance will be used to determine when the engine should shut down.
...	Extra room in case we later think of more things we should add.

Details

This creates an uninitialized BNEngine, specifically a BNEngineNDB.

Value

An object of calls BNEngineNDB which is capable of scoring student models.

Author(s)

Russell Almond

References

Almond, Mislevy, Steinberg, Yan and Williamson (2015). *Bayesian Networks in Educational Assessment*. Springer. Especially Chapter 13.

See Also

Classes: BNEngine, BNEngineMongo

Constituent parts: StudentRecordSet, PnetWarehouse ListenerSet

Setup Functions: loadManifest, setupDefaultSR, configStats, baselineHist,

Main Loop Functions: mainLoop, accumulateEvidence, handleEvidence, getRecordForUser, logEvidence, updateSM, updateStats, updateHist, announceStats,

Examples

```
## Requires database setup, also PNetica
library(RNetica)  ## Must load to setup Netica DLL
app <- "ecd://epls.coe.fsu.edu/EITest"
sess <- RNetica::NeticaSession()
RNetica::startSession(sess)

config.dir <- file.path(library(help="Peanut")$path, "auxdata")
net.dir <- file.path(library(help="PNetica")$path,"testnets")

netman <- read.csv(file.path(config.dir, "Mini-PP-Nets.csv"),
                   row.names=1, stringsAsFactors=FALSE)
stattab <- read.csv(file.path(config.dir, "Mini-PP-Statistics.csv"),
                   as.is=TRUE)

Nethouse <- PNetica::BNWarehouse(netman,session=sess,
             address=net.dir)

cl <- new("CaptureListener")
listeners <- list("cl"=cl)

ls <- ListenerSet(sender= paste("EAEngine[",app,"]"),
                  dburi="", listeners=listeners)

eng <- BNEngineNDB(app=app,warehouse=Nethouse,
                    listenerSet=ls,manifest=netman,
                    profModel="miniPP_CM",
                    histNodes="Physics",
                    statmat=stattab,
                    activeTest="EAActive.txt")
```

```
## Standard initialization methods.
loadManifest(eng,netman)
eng$setHistNodes("Physics")
configStats(eng,stattab)
setupDefaultSR(eng)
```

BNEngineNDB-class *Class* "BNEngineNDB"

Description

A BNEngine instance which is *not* connected to a database.

Extends

Class "BNEngine", directly.

All reference classes extend and inherit methods from "envRefClass".

Methods

evidence `signature(x = "BNEngineNDB")`: Returns list of EvidenceSets in the queue.

evidence `signature(x = "BNEngineNDB", value="list")`: Sets the list of EvidenceSets in the queue.

Activation

When the mainLoop runs out of evidence sets to process, it checks the `is.active()` method of the engine. If this returns, false, it stops.

For the BNEngineNDB this checks the existance of the file `activeTest`. If this file exists, the engine is considered active. Deleting this file will cause the engine to become inactive and stop after it finishes processing existing events.

Fields

`app`: Object of class `character` giving an globally unique identifier for the application

`srs`: Object of class StudentRecordSet of `NULL` giving the student record set for the application.

`profModel`: Object of class `character` giving the name of the proficiency model (for the default student record) in the warehouse manifest.

`listenerSet`: Object of class `ListenerSet` giving a set of listeners who will listen for new statistics.

`statistics`: Object of class `list` containing Statistic objects to be run on every update cycle.

`histNodes`: Object of class `character` giving the names of the nodes in the proficiency model whose history will be recorded.

`warehouseObj`: Object of class `PnetWarehouse` which stores the Bayes nets, both evidence models and student models are stored here.

`waittime`: Object of class `numeric` giving the time in seconds the main event loop should wait before checking again for messages.

`processN`: Object of class `numeric` giving the number of times that the main loop should run before stopping. If `Inf`, then the main loop will run without stopping.

`manifest`: Object of class `data.frame` which provides the manifest for the `PnetWarehouse`

`histnodes`: Object of class `character` which gives the names of the nodes for whom history will be recorded.

`evidenceQueue`: A `list` of `EvidenceSet` events to be processed.

`statmat`: Object of class `data.frame` which gives the descriptions of the `Statistic` objects to be used with the net.

`activeTest`: A pathname to the file whose existance will be checked to determine whether or not the engine should be considered active.

Class-Based Methods

`studentRecords()`: Returns the `StudentRecordSet` associated with this engine.

`fetchStats()`: Fetches the statistics marked in the database configuration.

`activate()`: Creates the `activeTest` file to activate the engine.

`fetchStats()`: Fetches the statistics or information in the `statmat` field.

`initialize(app, warehouse, listeners, profModel, waittime, statistics, histNodes, evidenceQueue, pr`
Initializes this class

`saveManifest(manifest)`: This sets the internal manifest field.

`fetchManifest()`: This returns the internal manifest field.

`fetchNextEvidence()`: This returns the first evidence set from the `evidenceQueue` field, and removes that element from the queue.

`saveStats(statmat)`: This saves the statistic table to the internal field.

`isActivated()`: This checks for the existance of the field in the `activeTets` field.

`evidenceSets()`: This returns NULL

`show()`: This produces a printable summary.

The following methods are inherited (from the corresponding class): evidenceSets ("BNEngine"), stats ("BNEngine"), setProcessed ("BNEngine"), setManifest ("BNEngine"), activate ("BNEngine"), isActivated ("BNEngine"), saveManifest ("BNEngine"), setHistNodes ("BNEngine"), studentRecords ("BNEngine"), saveStats ("BNEngine"), fetchNextEvidence ("BNEngine"), setError ("BNEngine"), getHistNodes ("BNEngine"), warehouse ("BNEngine"), show ("BNEngine"), fetchManifest ("BNEngine"), fetchStats ("BNEngine")

Note

The assumption of this engine is that the serialized student model will be passed in along with the evidence and will be returned along with the updated statistics.

Author(s)

Russell Almond

References

Almond, Mislevy, Steinberg, Yan and Williamson (2015). *Bayesian Networks in Educational Assessment*. Springer. Especially Chapter 13.

See Also

Classes: BNEngine, BNEngineMongo

Constituent parts: StudentRecordSet, PnetWarehouse

Setup Functions: loadManifest, setupDefaultSR, configStats, baselineHist,

Main Loop Functions: mainLoop, accumulateEvidence, handleEvidence, getRecordForUser, logEvidence, updateSM, updateStats, updateHist, announceStats,

Examples

```
showClass("BNEngineNDB")
```

---

configStats *Configures the Statistic Objects for the BNEninge*

Description

As part of the scoring cycle, the BNEngine calculates the values of certain statistics of the student model. This function sets up those statistics.

Usage

```
configStats(eng, statmat = data.frame())
```

Arguments

eng	The BNEngine to be configured.
statmat	A data frame containing the statistic descriptions, see details.

Details

A Statistic is a functional that is applied to the student model (sm) of a StudentRecord. At the end of the evidence processing cycle, the function updateStats is called to calculate new values for the specified statistics.

The statmat argument should be a data.frame with three columns (all of mode character):

Name This column gives an identifier for the statistic used in the output message.

Fun This column gives the name of a function (see Statistic for a list of possible values) which calculates the statistic value.

Node This gives the name of a node in the competency model which is the focus of the statistic.

If the statmat argument is not supplied, then a default value based on the engine type is used. For the BNEngineMongo this data frame is taken from a table in the database. For the BNEngineNDB the default statmat is stored in a field in the engine.

Value

The modified engine argument is returned.

Author(s)

Russell Almond

References

Almond, Mislevy, Steinberg, Yan and Williamson (2015). *Bayesian Networks in Educational Assessment*. Springer. Especially Chapter 13.

See Also

Classes: BNEngine, Statistic updateStats, announceStats

Examples

```
## Requires database setup, also PNetica
library(RNetica)  ## Must load to setup Netica DLL
app <- "ecd://epls.coe.fsu.edu/EITest"
sess <- RNetica::NeticaSession()
RNetica::startSession(sess)

config.dir <- file.path(library(help="Peanut")$path, "auxdata")
net.dir <- file.path(library(help="PNetica")$path,"testnets")

netman <- read.csv(file.path(config.dir, "Mini-PP-Nets.csv"),
                   row.names=1, stringsAsFactors=FALSE)
stattab <- read.csv(file.path(config.dir, "Mini-PP-Statistics.csv"),
                   as.is=TRUE)

Nethouse <- PNetica::BNWarehouse(netman,session=sess,
             address=net.dir)

cl <- new("CaptureListener")
listeners <- list("cl"=cl)

ls <- ListenerSet(sender= paste("EAEngine[",app,"]"),
                  dburi="", listeners=listeners)
```

```
eng <- BNEngineNDB(app=app,warehouse=Nethouse,
                   listenerSet=ls,manifest=netman,
                   profModel="miniPP_CM",
                   histNodes="Physics",
                   activeTest="EAActive.txt")

## Standard initialization methods.
configStats(eng,stattab)
stats <- eng$stats()
stopifnot(all(sapply(stats,StatName)==stattab$Name),
          all(sapply(stats,function(s) s@fun)==stattab$Fun),
          all(sapply(stats,function(s) s@node)==stattab$Node))
```

`EvidenceSet` *Creates an Evidence Set Message*

Description

An `EvidenceSet` is a `P4Message` which contains observable variables for the Bayes net engine. It provides the observavbles associated with a single scoring context.

Usage

```
EvidenceSet(uid, context, timestamp = Sys.time(), obs = list(), app =
"default", mess = "Accumulate", sender = "EI", processed = FALSE)
```

Arguments

`uid`	A character scalar giving unique identifier for the student/player.
`context`	A character scalar giving a unique identifier for the scoring context (often game level or task).
`timestamp`	The time at which the evidence was recorded (POSIXt format).
`obs`	A named list giving the observable variables. The names and legal values correspond to the `context` and `app` values.
`app`	A character scalar giving the globally unique identifier of the application.
`mess`	A character scalar giving the message associated with the observables. (Part of the Proc 4 procotol).
`sender`	A character scalar giving the identity of the process which created the message. This will usually be an evidence identification process.
`processed`	A flag that is set when the evidence set has been processed.

Details

Aside from the `seqno` field, this is pretty much a generic `P4Message`. The `data` of the `P4Message` is the `observables` value fo the `EvidenceSet`.

Value

An object of class EvidenceSet.

Author(s)

Russell Almond

See Also

Class: EvidenceSet Methods: observables, seqno, parseEvidence

Using classes: StudentRecord

Examples

```
e1 <- EvidenceSet(uid="S1",app="Test",context="PPcompEM",
                  obs=list("CompensatoryObs"="Right"))

e2 <- EvidenceSet(uid="S1",app="Test",context="PPdurAttEM",
                  obs=list("Attempts"=2,"Duration"=38.3))
```

EvidenceSet-class *Class* "EvidenceSet"

Description

An EvidenceSet is a collection of observables that comes from a particular context (scoring window, task). It also has information about where it appears in the sequence of evidence that is recorded about a student. It is an extension of the P4Message class.

Objects from the Class

Objects can be created calls to the function EvidenceSet(uid, context, timestamp, obs, app, mess, sender).

Slots

seqno: Object of class "integer" which contains the order in which this object was processed.

_id: Object of class "character" which contains the database ID.

app: Object of class "character" which gives a guid for the application.

uid: Object of class "character" which gives an id for the student.

context: Object of class "character" which gives an id for the scoring context.

sender: Object of class "character" which gives an ID for the source of the evidence.

`mess`: Object of class `"character"` which gives a message about what is contained in the message.

`timestamp`: Object of class `"POSIXt"` which tells when the evidence was collected.

`processed`: Object of class `"logical"` which is a flag to tell of the evidence has been incorporated into the `StudentRecord`.

`pError`: Object of class `"ANY"` which contains processing error.

`data`: Named list which contains the evidence.

Extends

Class `"P4Message"`, directly.

Methods

as.jlist `signature(obj = "EvidenceSet", ml = "list")`: This is a helper function used in serialization. See `as.json`.

observables `signature(x = "EvidenceSet")`: returns a named list of observables (the `data`) field.

seqno `signature(x = "EvidenceSet")`: returns the sequence number.

seqno<- `signature(x = "EvidenceSet")`: sets the sequence number.

show `signature(object = "EvidenceSet")`: prints a summary of the evidence set.

toString `signature(x = "EvidenceSet")`: provides a summary string for the evidence set.

Author(s)

Russell Almond

See Also

`StudentRecord`, `accumulateEvidence`, `handleEvidence`, `logEvidence`, `parseEvidence`, `seqno`, `observables`

Examples

```
showClass("EvidenceSet")
```

---

fetchNextEvidence *Fetches evidence from the evidence stream and marks it as processed.*

Description

The `BNEngine` processes a queue of evidence objects (either in a database or in a list). The function `fetchNextEvidence` fetches the oldest unprocessed `EvidenceSet` from the queue. The function `markProcessed` marks the evidence set as processed.

Usage

```
fetchNextEvidence(eng)
markProcessed(eng, eve)
```

Arguments

eng	A BNEngine which is the event handling system.
eve	A EvidenceSet which has been processed.

Details

For the BNEngineMongo the EvidenceSets reside in a collection. The fetchNextEvidence fetches the oldest unprocessed record from the collection. The markProcessed updates the record in the database to indicate that it has been processed.

For the BNEngineNDB the queue is an in-memory list. The fetchNextEvidence fetches the pops the next element from the list, and markProcessed marks it as processed, but otherwise does nothing.

Value

Both functions return the EvidenceSet.

Author(s)

Russell Almond

See Also

BNEngine, BNEngineMongo, BNEngineNDB, EvidenceSet

mainLoop, handleEvidence

Examples

```
## Requires database setup, also PNetica
library(RNetica)  ## Must load to setup Netica DLL
app <- "ecd://epls.coe.fsu.edu/EITest"
sess <- RNetica::NeticaSession()
RNetica::startSession(sess)

config.dir <- file.path(library(help="Peanut")$path, "auxdata")
net.dir <- file.path(library(help="PNetica")$path,"testnets")

netman <- read.csv(file.path(config.dir, "Mini-PP-Nets.csv"),
                   row.names=1, stringsAsFactors=FALSE)
stattab <- read.csv(file.path(config.dir, "Mini-PP-Statistics.csv"),
                   as.is=TRUE)

Nethouse <- PNetica::BNWarehouse(netman,session=sess,
             address=net.dir)
```

```
cl <- new("CaptureListener")
listeners <- list("cl"=cl)

ls <- ListenerSet(sender= paste("EAEngine[",app,"]"),
                  dburi="", listeners=listeners)

eng <- BNEngineNDB(app=app,warehouse=Nethouse,
                     listenerSet=ls,manifest=netman,
                     profModel="miniPP_CM",
                     histNodes="Physics",
                     statmat=stattab,
                     activeTest="EAActive.txt")

## Standard initialization methods.
loadManifest(eng,netman)
eng$setHistNodes("Physics")
configStats(eng,stattab)
setupDefaultSR(eng)


e1 <- EvidenceSet(uid="S1",app="Test",context="PPcompEM",
                  obs=list("CompensatoryObs"="Right"))
e1@"_id" <- "_E1"


e2 <- EvidenceSet(uid="S1",app="Test",context="PPdurAttEM",
                  obs=list("Attempts"=2,"Duration"=38.3))
e2@"_id" <- "_E2"


evidence(eng) <- list(e1,e2)

eve1 <- fetchNextEvidence(eng)
stopifnot(m_id(eve1)==m_id(e1))

eve1 <- markProcessed(eng,eve1)
stopifnot(eve1@processed)

stopifnot(length(evidence(eng))==1L)

eve2 <- fetchNextEvidence(eng)
stopifnot(m_id(eve2)==m_id(e2))
```

`fetchSM` *Fetches student model from database or JSON*

Description

The function fetchSM retrieves the student model from a PnetWarehouse or if not there, attempts to recreate it from a serialized version. The function unpackSM does this unpacking.

Usage

```
fetchSM(sr, warehouse)
unpackSM(sr, warehouse)
```

Arguments

sr	An object of class StudentRecord whose student model we wish to retrieve.
warehouse	A PnetWarehouse which stores the student models.

Details

The StudentRecord object has two fields related to student models: sm and smser. The former contains the actual student model or NULL if it has not yet been initialized or restored from the database. The latter contains a character string which contains a serialized version of the student model. In particular, it is this serialized student model which is stored in the database, not the actual student model.

The function fetchSM is used to set the sm field. It checks the following places in order:

1. It looks in the warehouse for a student net for the given uid for the record.
2. It calls unpackSM to unpack the serialized record.

The function unpackSM is wrapper for the function WarehouseUnpack.

Value

The function fetchSM returns the modified StudentRecord.

The function unpackSM returns the student model (a Pnet).

Author(s)

Russell Almond

See Also

StudentRecord

PnetWarehouse, WarehouseUnpack

Examples

```
library(PNetica)

##Start with manifest
sess <- NeticaSession()
startSession(sess)

## BNWarehouse is the PNetica Net Warehouse.
## This provides an example network manifest.
config.dir <- file.path(library(help="Peanut")$path, "auxdata")
netman1 <- read.csv(file.path(config.dir,"Mini-PP-Nets.csv"),
                    row.names=1, stringsAsFactors=FALSE)
net.dir <- file.path(library(help="PNetica")$path, "testnets")
Nethouse <- BNWarehouse(manifest=netman1,session=sess,key="Name",
                        address=net.dir)

dsr <- StudentRecord("*DEFAULT*",app="ecd://epls.coe.fsu.edu/P4Test",
                     context="*Baseline*")
sm(dsr) <- WarehouseSupply(Nethouse,"miniPP_CM")
PnetCompile(sm(dsr))

## dsr <- updateStats(eng,dsr)
statmat <- read.csv(file.path(config.dir,"Mini-PP-Statistics.csv"),
                    stringsAsFactors=FALSE)
rownames(statmat) <- statmat$Name
statlist <- sapply(statmat$Name,function (st)
    Statistic(statmat[st,"Fun"],statmat[st,"Node"],st))
names(statlist) <- statmat$Name
dsr@stats <- lapply(statlist,
                    function (stat) calcStat(stat,sm(dsr)))
names(dsr@stats) <- names(statlist)

## dsr <- baselineHist(eng,dsr)
dsr@hist <- lapply(c("Physics"),
                   function (nd)
                   EABN:::uphist(sm(dsr),nd,NULL,"*Baseline*"))
names(dsr@hist) <- "Physics"

pnodenames <- names(PnetPnodes(sm(dsr)))


## Serialization and unserialization
dsr.ser <- as.json(dsr)

dsr1 <- parseStudentRecord(fromJSON(dsr.ser))
stopifnot(is.null(sm(dsr1)))
## at this point, SM has not yet been restored.


## It is there in the serial field
net1 <- unpackSM(dsr1,Nethouse)
```

```
stopifnot(all.equal(pnodenames,names(PnetPnodes(net1))))
dsr1 <- fetchSM(dsr1,Nethouse)
stopifnot(all.equal(pnodenames,names(PnetPnodes(sm(dsr1)))))

## Try this again, but first delete net from warehouse,
## So we are sure we are building it from serialized version.
WarehouseFree(Nethouse,PnetName(sm(dsr)))

dsr1 <- parseStudentRecord(fromJSON(dsr.ser))
stopifnot(is.null(sm(dsr1)))
## at this point, SM has not yet been restored.


## It is there in the serial field
net1 <- unpackSM(dsr1,Nethouse)
stopifnot(all.equal(pnodenames,names(PnetPnodes(net1))))
dsr1 <- fetchSM(dsr1,Nethouse)
stopifnot(all.equal(pnodenames,names(PnetPnodes(sm(dsr1)))))
```

`getRecordForUser` *Gets or makes the student record for a given student.*

Description

The `BNEngine` contains a `StudentRecordSet`, which is a collection of `StudentRecord` objects. The function `getRecordForUser` fetches one from the collection (if it exists) or creates a new one.

Usage

```
getRecordForUser(eng, uid, srser = NULL)
```

Arguments

`eng`	The `BNEngine` in question.
`uid`	A character scalar giving the unique identifier for the student.
`srser`	A serialized version of the student record. Used to extract the student record in database-free mode. This should either be a list which is the output of `fromJSON` or `NULL`.

Details

The student record set can either be attached to a database (the `dburi` field passed to `StudentRecordSet` is non-empty, or not. In the database mode, recrods are saved in the database, so that they can be retrieved across sessions. In the database-free mode, the serialized student record (if it exists) should be passed into the `getRecordForUser` function.

If no student record is available for the `uid`, then a new one is created by cloning the default student record (see `setupDefaultSR`).

This function mostly just calls getSR on the StudentRecordSet; however, if a new record is generated, then announceStats is called to advertise the baseline statistics for the new user.

Value

The StudentRecord object is returned.

Warning

Calling this multiple times will not return the same student record. In particular, the student model associated with the old version of the record could be replaced with a new version, rendering the student model in the old records inactive. Be careful when dealing with old records.

Author(s)

Russell Almond

References

Almond, Mislevy, Steinberg, Yan and Williamson (2015). *Bayesian Networks in Educational Assessment*. Springer. Especially Chapter 13.

See Also

BNEngine, StudentRecordSet, StudentRecord

handleEvidence, setupDefaultSR, fetchSM, getSR

Examples

```
library(PNetica)

##Start with manifest
sess <- NeticaSession()
startSession(sess)

## BNWarehouse is the PNetica Net Warehouse.
## This provides an example network manifest.
config.dir <- file.path(library(help="Peanut")$path, "auxdata")
netman1 <- read.csv(file.path(config.dir,"Mini-PP-Nets.csv"),
                    row.names=1, stringsAsFactors=FALSE)
net.dir <- file.path(library(help="PNetica")$path, "testnets")
stattab <- read.csv(file.path(config.dir, "Mini-PP-Statistics.csv"),
                    as.is=TRUE)
Nethouse <- BNWarehouse(manifest=netman1,session=sess,key="Name",
                        address=net.dir)


cl <- new("CaptureListener")
listeners <- list("cl"=cl)
```

```
ls <- ListenerSet(sender= "EAEngine[Test]",
                  dburi="", listeners=listeners)

eng <- BNEngineNDB(app="Test",warehouse=Nethouse,
                     listenerSet=ls,manifest=netman,
                     profModel="miniPP_CM",
                     histNodes="Physics",
                     statmat=stattab,
                     activeTest="EAActive.txt")

## Standard initialization methods.
loadManifest(eng,netman1)
eng$setHistNodes("Physics")
configStats(eng,stattab)
setupDefaultSR(eng)

sr0a <- getRecordForUser(eng,"Student1")
sr0 <- getRecordForUser(eng,"Student1")
## This is announcing twice, so not quite working with NDB engine.

stopifnot(is.active(sm(sr0)),!is.active(sm(sr0a)))
stopifnot(all.equal(stats(sr0),stats(sr0a)))
eap0<- stat(sr0,"Physics_EAP")


e1 <- EvidenceSet(uid="Student1",app="Test",context="PPcompEM",
                  obs=list("CompensatoryObs"="Right"))

e1 <- logEvidence(eng,sr0,e1)
sr1 <- accumulateEvidence(eng,sr0,e1)
stopifnot(m_id(sr1)!=m_id(sr0),sr1@prev_id==m_id(sr0))
stopifnot(seqno(sr1)==1L, seqno(e1)==1L)

eap1 <- stat(sr1,"Physics_EAP")
stopifnot(abs(eap1-eap0) > .001)
stopifnot(nrow(history(sr1,"Physcis"))==2L)

sr1.ser <- as.json(sr1)
WarehouseFree(Nethouse,PnetName(sm(sr1))) # Delete student model to
                                          # force restore.
sr1a <- getRecordForUser(eng,"Student1",fromJSON(sr1.ser))
#PnetCompile(sm(sr1a))
eap1a <- stat(sr1a,"Physics_EAP")
stopifnot(abs(eap1-eap1a) < .001)
stopifnot(nrow(history(sr1a,"Physcis"))==2L)


##  <<Here>>  Need test with Mongo engine
```

getSR *Save and retrieve student records from a record set.*

Description

A StudentRecordSet is a collection of StudentRecord objects. The function getSR fetches one from the collection if it exists. The function newSR creates a new one. The function saveSR saves the student record, and clearSRs clears out the saved student records.

Usage

```
getSR(srs, uid, ser = "")
newSR(srs, uid)
saveSR(srs, rec)
clearSRs(srs)
```

Arguments

srs	The StudentRecordSet in question.
uid	A character scalar giving the unique identifier for the student.
ser	A serialized version of the student record. Used to extract the student record in database-free mode. This should either be a list which is the output of fromJSON or NULL.
rec	A StudentRecord to be saved.

Details

The student record set can either be attached to a database (the dburi field passed to StudentRecordSet is non-empty, or not. In the database mode, recrods are saved in the database, so that they can be retrieved across sessions. In the database-free mode, the serialized student record (if it exists) should be passed into the getSR function.

The functions operate as follows:

getSR If the ser argument is not NULL, then the serialized student record is used to fetch the student record. Otherwise, the database (if it exists) is searched for a student record with the proper application and user ids. Then fetchSM is called to fetch the student model. If both of those methods fail, it returns NULL.

newSR This creates a new StudentRecord from the defaultSR field of the student record set (see setupDefaultSR). The function saveSR is called to save the new record.

saveSR If the database exists, the student record is saved to the database. Otherwise, if no m_id exists for the record one is created from the uid and seqno.

clearSRs In database mode, it clears the database. Otherwise, nothing is done.

Value

The functions `getSR`, `newSR` and `saveSR` return the student record or `NULL` if the record was not found or created.

The function `clearSRs` returns the student record set (its argument).

Author(s)

Russell Almond

See Also

Classes: `BNEngine`, `StudentRecordSet`, `StudentRecord`

Functions: `handleEvidence`, `setupDefaultSR`, `fetchSM`, `StudentRecordSet`

Examples

```
## Requires database setup, also PNetica
library(RNetica)  ## Must load to setup Netica DLL
app <- "ecd://epls.coe.fsu.edu/EITest"
sess <- RNetica::NeticaSession()
RNetica::startSession(sess)

config.dir <- file.path(library(help="Peanut")$path, "auxdata")
net.dir <- file.path(library(help="PNetica")$path,"testnets")

netman <- read.csv(file.path(config.dir, "Mini-PP-Nets.csv"),
                   row.names=1, stringsAsFactors=FALSE)
stattab <- read.csv(file.path(config.dir, "Mini-PP-Statistics.csv"),
                   as.is=TRUE)

Nethouse <- PNetica::BNWarehouse(netman,session=sess,
             address=net.dir)

cl <- new("CaptureListener")
listeners <- list("cl"=cl)

ls <- ListenerSet(sender= paste("EAEngine[",app,"]"),
                  dburi="", listeners=listeners)

eng <- BNEngineNDB(app=app,warehouse=Nethouse,
                    listenerSet=ls,manifest=netman,
                    profModel="miniPP_CM",
                    histNodes="Physics",
                    statmat=stattab,
                    activeTest="EAActive.txt")

## Standard initialization methods.
loadManifest(eng,netman)
eng$setHistNodes("Physics")
```

```
configStats(eng,stattab)
setupDefaultSR(eng)

tr1 <- newSR(eng$studentRecords(),"Test1")
PnetCompile(sm(tr1))
stopifnot(uid(tr1)=="Test1",abs(stat(tr1,"Physics_EAP")) < .0001)
stopifnot(is.na(m_id(tr1)))  # id is NA as it has not been saved yet.

tr1 <- saveSR(eng$studentRecords(),tr1)
m_id(tr1)
stopifnot(!is.na(m_id(tr1)))  # Now set

sr0 <- getRecordForUser(eng,"S1")

eap0 <- stat(sr0,"Physics_EAP")

e1 <- EvidenceSet(uid="S1",app="Test",context="PPcompEM",
                  obs=list("CompensatoryObs"="Right"))

e1 <- logEvidence(eng,sr0,e1)
sr1 <- accumulateEvidence(eng,sr0,e1)
stopifnot(m_id(sr1)!=m_id(sr0),sr1@prev_id==m_id(sr0))
stopifnot(seqno(sr1)==1L, seqno(e1)==1L)

eap1 <- stat(sr1,"Physics_EAP")
stopifnot(abs(eap1-eap0) > .001)
stopifnot(nrow(history(sr1,"Physcis"))==2L)

sr1.ser <- as.json(sr1)
WarehouseFree(Nethouse,PnetName(sm(sr1))) # Delete student model to
                                          # force restore.

sr1a <- getSR(eng$studentRecords(),"S1",fromJSON(sr1.ser))
PnetCompile(sm(sr1a))
eap1a <- stat(sr1a,"Physics_EAP")
stopifnot(abs(eap1-eap1a) < .001)
stopifnot(nrow(history(sr1a,"Physcis"))==2L)



##  <<Here>> Need test with Mongo implementation
```

`history` *Retrieves node histories from a Student Record*

Description

A history is a `data.frame` whose rows correspond to `EvidenceSet` objects and whose columns correspond to the states of a `Pnode`. Each row is a probability distribution, and they show the

changes to the probabilities over time.

The function history returns the history for a single node in a given StudentRecord. The function histNames returns the names of the nodes for which the record has history information.

Usage

```
history(sr, name)
histNames(sr)
```

Arguments

sr	A StudentRecord whose history is to be accessed.
name	The name of the node whose history is requested.

Details

When the student record is first initialized, the function baselineHist is called to setup "*BASELINE*" values for each of the history nodes identified by the BNEngine. These are data.frame objects giving the prior marginal distributions for each of the identified variables.

After the student model is updated in response to evidence (see handleEvidence, the updateHist function is called to add a new row to each of the data frames.

The histNames function returns the names of the history nodes being tracked by a student model. The history function returns the history for a node.

Value

The function histNames returns a list of node names. These are suitable for the name argument of the history function.

The function history returns a data frame with rows corresponding to evidence sets and columns corresponding to states of the variables. Each row is a marginal probability distribution.

Note

These are designed to work with the functions woeHist and woeBal in the CPTtools-package.

Author(s)

Russell Almond

See Also

StudentRecord for student records.

baselineHist and updateHist for history construction.

BNEngine for specifying the history nodes.

woeHist and woeBal for applications.

Examples

```
library(PNetica)

##Start with manifest
sess <- NeticaSession()
startSession(sess)

## BNWarehouse is the PNetica Net Warehouse.
## This provides an example network manifest.
config.dir <- file.path(library(help="Peanut")$path, "auxdata")
netman1 <- read.csv(file.path(config.dir,"Mini-PP-Nets.csv"),
                    row.names=1, stringsAsFactors=FALSE)
net.dir <- file.path(library(help="PNetica")$path, "testnets")
Nethouse <- PNetica::BNWarehouse(manifest=netman1,session=sess,key="Name",
                       address=net.dir)
dsr <- StudentRecord("*DEFAULT*",app="ecd://epls.coe.fsu.edu/P4Test",
                     context="*Baseline*")
sm(dsr) <- WarehouseSupply(Nethouse,"miniPP_CM")
PnetCompile(sm(dsr))
## dsr <- updateStats(eng,dsr)

statmat <- read.csv(file.path(config.dir,"Mini-PP-Statistics.csv"),
                    stringsAsFactors=FALSE)
rownames(statmat) <- statmat$Name
statlist <- sapply(statmat$Name,function (st)
    Statistic(statmat[st,"Fun"],statmat[st,"Node"],st))
names(statlist) <- statmat$Name
dsr@stats <- lapply(statlist,
                    function (stat) calcStat(stat,sm(dsr)))
names(dsr@stats) <- names(statlist)
stat(dsr,"Physics_EAP")
stat(dsr,"Physics_Margin")

## dsr <- baselineHist(eng,dsr)

dsr@hist <- lapply(c("Physics"),
                     function (nd)
                     EABN:::uphist(sm(dsr),nd,NULL,"*Baseline*"))
names(dsr@hist) <- "Physics"
stopifnot(histNames(dsr)=="Physics")
history(dsr,"Physics")
```

`loadManifest` *Loads the mainifest for the compentency and evidence models in the BNEngine*

Description

This sets the manifest of networks used in the scoring engine. In particular, it sets the WarehouseManifest of the PnetWarehouse associated with a BNEngine.

Usage

```
loadManifest(eng, manifest = data.frame())
```

Arguments

eng	A BNEngine whose manifest is to be set.
manifest	A dataframe containing a network manifest (see BuildNetManifest). If missing, then the manifest will be retrieved from the database or other cached source.

Details

The BNEngine requires a proficiency or competency model (which is used to build student models) and a collection of evidence models (one for each scoring context) which are all expressed as Pnets. The manifest is basically a table of which evidence model networks go with which scoring contexts. The proficienty model usually serves as the hub in the hub-and-spoke framework. (In fact, if the profModel argument is not supplied when the BNEngine is built, the engine will look for a network which has no hub in the manifest.

In fact, the manifest is part of the PnetWarehouse which is a field of the engine. It should have the format associate with manifests described in WarehouseManifest. Note that the Bayes nets should have already been built, so the the warehouse should point to where they can be loaded from the filesystem on demand.

For the BNEngineMongo, the default manifest is located in a table in the database. If no manifest is supplied, then the manifest is read from the database. For the BNEngineNDB, the manifest must be specified manually when the engine is contructed (or when loadManifest is called).

Value

This function returns the engine argument.

Note

The loadManifest call is part of the initialization sequence for the BNEngine. However, if the manifest is loaded into the PnetWarehouse as it is built, it is really redundant.

Author(s)

Russell Almond

References

Almond, Mislevy, Steinberg, Yan and Williamson (2015). *Bayesian Networks in Educational Assessment*. Springer. Especially Chapter 13.

See Also

Classes: BNEngine, BNEngineMongo, BNEngineNDB, PnetWarehouse

Functions: WarehouseManifest, BuildNetManifest

Examples

```
## Requires database setup, also PNetica
library(RNetica)  ## Must load to setup Netica DLL
app <- "ecd://epls.coe.fsu.edu/EITest"
sess <- RNetica::NeticaSession()
RNetica::startSession(sess)

config.dir <- file.path(library(help="Peanut")$path, "auxdata")
net.dir <- file.path(library(help="PNetica")$path,"testnets")

netman <- read.csv(file.path(config.dir, "Mini-PP-Nets.csv"),
                   row.names=1, stringsAsFactors=FALSE)
stattab <- read.csv(file.path(config.dir, "Mini-PP-Statistics.csv"),
                   as.is=TRUE)

## Deliberately build warehouse without empty manifest.
Nethouse <- PNetica::BNWarehouse(session=sess,
             address=net.dir)

cl <- new("CaptureListener")
listeners <- list("cl"=cl)

ls <- ListenerSet(sender= paste("EAEngine[",app,"]"),
                  dburi="", listeners=listeners)

eng <- BNEngineNDB(app=app,warehouse=Nethouse,
                    listenerSet=ls,manifest=netman,
                    profModel="miniPP_CM",
                    histNodes="Physics",
                    statmat=stattab,
                    activeTest="EAActive.txt")

stopifnot(nrow(WarehouseManifest(eng$warehouse())) == 0L)

## Standard initialization methods.
loadManifest(eng,netman)
stopifnot(nrow(WarehouseManifest(eng$warehouse())) == 5L)
```

---

logEvidence *Handle the relationship between evidence sets and student records.*

Description

A StudentRecord differs from the baseline student record according to how many EvidenceSet objects have been incorporated into the estimate. These functions tie and student record and evidence set together.

Usage

```
update
logEvidence(eng, rec, evidMess)
seqno(x)
seqno(x) <- value
evidence(x)
evidence(x) <- value
```

Arguments

eng	A BNEngine which is currently not used (could later be used to save the evidence to a database).
rec	A StudentRecord into which the evidence will be incorporated.
evidMess	A EvidenceSet which will be associated with the student record.
x	An EvidenceSet object.
value	For seqno(x) <- value, an integer giving a new sequence number. For evidence(x) <- value, a character vector giving the sequence of evidence ID.

Details

There are several fields in the StudentRecord class which need to be updeated in the face of new evidence.

context and timestamp These needs to be set to the values in the new evidence message.

seqno This needs to be incremented.

evidence The new evidence needs to be prepended to this list.

prev_id and "_id" The prev_id needs to point to the old field and the "_id" is set to NA (it will be updated on save).

In the case of the BNEngineMongo, the IDs in question are the database ids for these objects so that they can be easily found. The function m_id For the BNEngineNDB case presumably some external system is issuing IDs to evidence sets and student records.

The evidence field of a StudentRecord is a list of IDs (m_id) for the accumulated evidence.

The seqno field is an optional ordering used to track the order in which evidence sets were incorporated into the student model. The value of seqno gives the number of evidence sets incorporated into the recrod.

The logEvidence function sets the sequence number of the evidence message to one more than the last sequence number for the student record. If no m_id exists for the record (no database mode), then one is generated by concatenating the uid and the seqno.

Value

The updateRecord returns a new StudentRecord object, which points back to the old one.

The logEvidence function returns the modified EvidenceSet.

The function seqno returns an integer (or NA if has not been set).

The function evidence returns a character vector giving the IDs (m_id) of the encorpated evidence sets.

Note

This is largely untested code for future fast retraction of evidence.

The prev_id field of the StudentRecord should leave a trace of previous student records in the database, including old serialized models. This should allow the scoring engine to quickly jump back in time.

The evidence field provides a list of the m_ids of all the incorporated evidence sets. This should enable one or more evidence sets to be replaced and the student model to be recalculated.

Author(s)

Russell Almond

See Also

BNEngine, EvidenceSet, EvidenceSet StudentRecord, handleEvidence P4Message

Examples

```
recset <- StudentRecordSet(dburi="")

sr0 <-
    StudentRecord("S1","*baseline*",as.POSIXct("2020-03-30 09:00:00"))
seqno(sr0) <- 0
sr0 <- saveSR(recset,sr0) # Sets the m_id

e1 <- EvidenceSet(uid="S1",app="Test",context="PPcompEM",
                  obs=list("CompensatoryObs"="Right"))

e2 <- EvidenceSet(uid="S1",app="Test",context="PPdurAttEM",
                  obs=list("Attempts"=2,"Duration"=38.3))

stopifnot(is.na(seqno(e1)), seqno(sr0)==0L)
stopifnot(length(evidence(sr0))==0L)

e1 <- logEvidence(NULL,sr0,e1)
stopifnot(seqno(e1)==1L,!is.na(m_id(e1)))

sr1 <- updateRecord(sr0,e1)
stopifnot(is.na(m_id(sr1)),sr1@prev_id==m_id(sr0))
sr1 <- saveSR(recset,sr1)    # Sets the m_id
```

```
stopifnot(length(evidence(sr1))==1L,any(m_id(e1)==evidence(sr1)))
stopifnot(context(sr1)==context(e1),timestamp(sr1)==timestamp(e1))

e2 <- logEvidence(NULL,sr1,e2)
stopifnot(seqno(e2)==2L,!is.na(m_id(e2)))

sr2 <- updateRecord(sr1,e2)
stopifnot(is.na(m_id(sr2)),sr2@prev_id==m_id(sr1))
sr2 <- saveSR(recset,sr2) # Sets the m_id

stopifnot(length(evidence(sr2))==2L,any(m_id(e2)==evidence(sr2)))
stopifnot(context(sr2)==context(e2),timestamp(sr2)==timestamp(e2))
```

`mainLoop` *This function loops through the processing of evidence sets.*

Description

The `mainLoop` is used when the `BNEngine` is used as a server. It checks the queue (database or internal list), for unprocessed `EvidenceSet` objects, and calls `handleEvidence` on them in the order of their `timestamp`s. As a server, this is potentially an infinite loop, see details for ways of gracefully terminating the loop.

Usage

```
mainLoop(eng)
```

Arguments

`eng`	An `BNEngine` which will handle the evidence sets.

Details

The `BNEngineMongo` class uses the EvidenceSets collection in the database as a queue. All events have a `processed` field which is set to true when the evidence set is processed. The function `fetchNextEvidence` fetches the oldest unprocessed evidence set.

The `BNEngineNDB` has an internal `eventQueue` field which is a list of evidence setsn. The `fetchNextEvidence` simply fetches the first evidence set in the queue.

The `mainLoop` function iterates over the following steps.

1. Fetch the oldest unprocessed Event: `eve <- fetchNextEvidence()`.
2. Process the evidence set: `out <- handleEvidence(eng,eve)`. (Note: this expression will always return. If it generates an error, the error will be logged and an object of class `try-error` will be returned.)
3. Mark the event as processed: `markProcessed(eve)`.

At its simplest level, the funciton produces an infinite loop over these three statements, with some additional steps related to logging and control.

First, if the event queue is empty, the process sleeps for a time given by `eng$waittime` and then checks the queue again. At the same time, it checks status of the active flag for the process using the `eng$isActivated()` call.

For the Mongo implementation, `eng$isActivated()` checks the `active` field of the record corresponding to `app(eng)` in the collection `AuthorizedApps` in the database `Proc4`. Setting that field to false manually will result in the `mainLoop` terminating when the queue is empty. As R is running in server mode when this happens, this often needs to be done using an external process. The following command issues from the Mongo shell will shut down the server for an application containing the string "appName" as part of its name.

```
db.AuthorizedApps.update({app:{$regex:"appName"}},    {$set:{active:false}});
```

For the no database implementation, `eng$isActivated()` checks for the existance of the file named in the `activeTest` field. Deleting that file will have the same effect as setting the active field to false in the database version.

To facilitate testing, the field `eng$processN` can be set to a finite value. This number is decremented at every cycle, and when it reaches 0, the `mainLoop` is terminated, whether or not their are any remaining events to be processed. Setting `eng$processN` to an infinite value, will result in an infinite loop that can only be stopped by using the active flag (or interrupting the process).

Value

There is no return value. The function is used entirely for its side effects.

Note

Currently, when running in server model (i.e., with `eng$processN` set to infinity), there are two ways of stopping the engine: a clean stop after all events are processed using the `active` flag, and an immediate stop, possibly mid cycle, by killing the server process. It became apparent during testing that there was a need for a graceful but immediate stop, i.e., a stop after processing the current event. This should appear in later versions.

Author(s)

Russell Almond

See Also

`BNEngine`, `BNEngineMongo`, `BNEngineNDB`

`fetchNextEvidence`, `handleEvidence`

Examples

```
## Not run:
## From EABN.R script
  app <- "ecd://epls.coe.fsu.edu/P4test"
  loglevel <- "DEBUG"
```

```
source("/usr/local/share/Proc4/EAini.R")
flog.appender(appender.file(logfile))
flog.threshold(loglevel)

sess <- NeticaSession(LicenseKey=NeticaLicenseKey)
startSession(sess)
listeners <- lapply(names(EA.listenerSpecs),
                    function (ll) do.call(ll,EA.listenerSpecs[[ll]]))
names(listeners) <- names(EA.listenerSpecs)

eng <- do.call(BNEngineMongo,
               c(EAeng.params,list(session=sess,listeners=listeners),
                 EAeng.common))
loadManifest(eng)
configStats(eng)
setupDefaultSR(eng)

## Activate engine (if not already activated.)
eng$activate()
mainLoop(eng)
## Wait for cows to come home.


## End(Not run)
```

---

observables *Access parts of an evidence set message.*

Description

The function observables access the list of observables contained in this EvidenceSet. The function seqno access the order in which the evidence sets were incorporated into the student record.

Usage

```
observables(x)
```

Arguments

x	An EvidenceSet object.

Details

The observables function access the data field of the underlying P4Message. This should be a named list of values that the BNEngine knows how to process.

Value

The function observables returns a named list of observable values.

Author(s)

Russell Almond

See Also

EvidenceSet, EvidenceSet StudentRecord, handleEvidence P4Message

Examples

```
e1 <- EvidenceSet(uid="S1",app="Test",context="PPcompEM",
                  obs=list("CompensatoryObs"="Right"))

e2 <- EvidenceSet(uid="S1",app="Test",context="PPdurAttEM",
                  obs=list("Attempts"=2,"Duration"=38.3))

stopifnot(all.equal(observables(e1),
                    list("CompensatoryObs"="Right")))

stopifnot(all.equal(observables(e2)$Attempts,2))

stopifnot(is.na(seqno(e1)))
seqno(e1) <- 1
stopifnot(seqno(e1)==1L)
```

---

parseEvidence *Convert EvidenceSet objects to and from JSON*

Description

The as.json function takes an EvidenceSet (among other objects) and turns it into JSON. The function parseEvidence takes the list produced as the output to fromJSON and turns it back into an EvidenceSet object.

Usage

```
parseEvidence(rec)
## S4 method for signature 'EvidenceSet,list'
as.jlist(obj, ml, serialize=TRUE)
```

Arguments

rec	A list which comes from running fromJSON on a JSON string, or database extraction method.
obj	The object being serialized; usually attributes(obj).
ml	A list of fields of the object.
serialize	A logical flag. If true, serializeJSON is used to protect the data field (and other objects which might contain complex R code.

Details

See the description for as.json for more description of the JSON conversion prototocl.

The parseEvidence method is designed to be used with the getOneRec and getManyRecs functions for fetching information from the database.

Value

The function parseEvidence returns an object of class EvidenceSet.

The as.jlist method returns a list which can be passed to toJSON to produce legible JSON from the R object.

Author(s)

Russell Almond

See Also

EvidenceSet, as.json, getOneRec, getManyRecs

Examples

```
e1 <- EvidenceSet(uid="S1",app="Test",context="PPcompEM",
                  obs=list("CompensatoryObs"="Right"))

e2 <- EvidenceSet(uid="S1",app="Test",context="PPdurAttEM",
                  obs=list("Attempts"=2,"Duration"=38.3))

e1.ser <- as.json(e1)
e1a <- parseEvidence(fromJSON(e1.ser))
e2.ser <- as.json(e2)
e2a <- parseEvidence(fromJSON(e2.ser))

stopifnot(all.equal(e1,e1a), all.equal(e2,e2a))
```

parseStats *Functions for (un)serializing stats from student records.*

Description

The functions unparseStats and stats2json serialize the statistics as a JSON record. The function parseStats reverses the process.

Usage

```
parseStats(slist)
unparseStats(slist, flatten=FALSE)
stats2json(slist, flatten=FALSE)
```

Arguments

slist	A list of statistics. For parseStats this should be the output of fromJSON. For the others, this is just a list of statistic values.
flatten	If true, then vector-valued statistics (i.e., PnodeMargin, will have their values flattened into scalars. If not they will be left as vectors.

Details

The function unparseStats massages the list of statistics so it will be output in clean JSON (in particular, using unboxer to make sure scalars appear as scalars and not vectors). The function stats2json is just toJSON(unparseStats(slist)).

If flatten is true, then vector value statstics will be flattened. For example, if the statistic "Physics_Margin" has three values with labels "High", "Medium", and "Low", then it will be replaced with three statistics with the names "Physics_Margin.High", "Physics_Margin.Medium", and "Physics_Margin.Low".

The function parseStatistics is designed to reverse the process.

Value

The function unparseStats returns a list which is ready to be passed to toJSON. In particular, scalars are marked using unboxer.

The function stats2json returns a string containing the JSON.

The function parseStats returns a list of statistics values. this is suitable for being set to the stats field of the StudentRecord object.

Note

When using flatten=TRUE, avoid periods, '.', in the names of statistics, as this marker is used to recreate the nested structure in parseStats.

Author(s)

Russell Almond

See Also

ParseMessage gives general information about how the parsing/unparsing protocol works.

Statistic gives a list of available statistics.

StudentRecord talks about the statitic fields of the student records.

Examples

```
stats <- list(Physics_EAP=0,EnergyTransfer_EAP=.15,
              Physics_Margin=c(High=1/3,Medium=1/3,
                               Low=1/3))
stats2json(stats)

stats1 <- parseStats(ununboxer(unparseStats(stats)))
stopifnot(all.equal(stats,stats1,tolerance=.0002))

stats2json(stats,flatten=TRUE)

stats2 <- parseStats(ununboxer(unparseStats(stats,flatten=TRUE)))
stopifnot(all.equal(stats,stats2,tolerance=.0002))
```

---

parseStudentRecord *Covert Student Records to/from JSON*

Description

The as.json function takes an StudentRecord (among other objects) and turns it into JSON. The function parseStudentRecord takes the list produced as the output to fromJSON and turns it back into an StudentRecord object.

Usage

```
parseStudentRecord(rec)
## S4 method for signature 'StudentRecord,list'
as.jlist(obj, ml, serialize=TRUE)
```

Arguments

rec	A list which comes from running fromJSON on a JSON string, or database extraction method.
obj	The object being serialized; usually attributes(obj).
ml	A list of fields of the object.
serialize	A logical flag. If true, serializeJSON is used to protect the data field (and other objects which might contain complex R code.

Details

See the description for as.json for more description of the general JSON conversion prototocl.

The StudentRecord contains a Pnet field in the student model. This takes some post-processing to to properly restore.

The as.jlist method for the StudentRecord serializes the sm field using the PnetSerialize method. This produces a slob (string large object) which is stored in the smser field of the StudentRecord.

The parseStudentRecord function restores the smser field, but not the sm field. This must be done in the context of the StudentRecordSet, or equivalently the PnetWarehouse, which is currently managing the networks. To finish the process, call fetchSM to restore the student model network.

Value

The function parseStudentRecord returns a student record object with the student model not yet initialized.

The as.jlist method returns a list which can be passed to toJSON to produce legible JSON from the R object.

Author(s)

Russell Almond

See Also

StudentRecord, as.json, getOneRec, getManyRecs

fetchSM, PnetSerialize

Examples

```
## Requires database setup, also PNetica
library(RNetica)  ## Must load to setup Netica DLL
app <- "ecd://epls.coe.fsu.edu/EITest"
sess <- RNetica::NeticaSession()
RNetica::startSession(sess)

config.dir <- file.path(library(help="Peanut")$path, "auxdata")
net.dir <- file.path(library(help="PNetica")$path,"testnets")

netman <- read.csv(file.path(config.dir, "Mini-PP-Nets.csv"),
                   row.names=1, stringsAsFactors=FALSE)
stattab <- read.csv(file.path(config.dir, "Mini-PP-Statistics.csv"),
                   as.is=TRUE)

Nethouse <- PNetica::BNWarehouse(netman,session=sess,
             address=net.dir)

cl <- new("CaptureListener")
listeners <- list("cl"=cl)
```

```
ls <- ListenerSet(sender= paste("EAEngine[",app,"]"),
                  dburi="", listeners=listeners)

eng <- BNEngineNDB(app=app,warehouse=Nethouse,
                     listenerSet=ls,manifest=netman,
                     profModel="miniPP_CM",
                     histNodes="Physics",
                     statmat=stattab,
                     activeTest="EAActive.txt")

## Standard initialization methods.
loadManifest(eng,netman)
eng$setHistNodes("Physics")
configStats(eng,stattab)
setupDefaultSR(eng)

recset <- eng$studentRecords()

sr0 <- getRecordForUser(eng,"S1")
eap0 <- stat(sr0,"Physics_EAP")

sr0.ser <- as.json(sr0)
sr0a <- parseStudentRecord(fromJSON(sr0.ser))
sr0a <- fetchSM(sr0a,recset$warehouse)
## This should relink to the same student model
stopifnot(sm(sr0a)==sm(sr0),abs(eap0-stat(sr0a,"Physics_EAP")) <.0001)

## Next add some evidence and test again.

e1 <- EvidenceSet(uid="S1",app="Test",context="PPcompEM",
                  obs=list("CompensatoryObs"="Right"))

e1 <- logEvidence(eng,sr0,e1)
sr1 <- accumulateEvidence(eng,sr0,e1)
eap1 <- stat(sr1,"Physics_EAP")
sr1.ser <- as.json(sr1)

## Force delete student model to make sure that it is properly
## recovered.
WarehouseFree(Nethouse,PnetName(sm(sr1)))
stopifnot(!is.active(sm(sr1)))  # No longer active.

sr1a <- parseStudentRecord(fromJSON(sr1.ser))
sr1a <- fetchSM(sr1a,recset$warehouse)
eap1a <- stat(sr1a,"Physics_EAP")
stopifnot(all(evidence(sr1)==evidence(sr1a)),
          abs(eap1-eap1a) <.001)
```

setupDefaultSR *Set up the Default Student Record for an StudentRecordSet*

Description

The default student record is a field associated with a StudentRecordSet which provides a template student record for a student just staring the assessment. The setupDefaultSR function needs to be called at the start of every scoring session to initialize the defaultSR field of the student record set.

Usage

```
setupDefaultSR(eng)
```

Arguments

eng	A BNEngine which contains the student record details.

Details

This function creates a new StudentRecord object with the special uid "*DEFAULT*" and the special context ID "*Baseline*". The student model is actually the competency or proficiency model: the baseline student model giving the population distribution of the the measured proficiencies. This is fetched by name from the PnetWarehouse attached to the engine; the name is given in the profModel field of the eng.

Setting up a default student record actually takes a number of steps:

1. The student record set (eng$studentRecrods()) is cleared by calling clearSRs.
2. A new blank student record (uid="*DEFAULT*") is created.
3. The sm field of the new student record is initialized to the proficiency model.
4. The student model is compiled (PnetCompile).
5. The baseline statisics are calculated (updateStats).
6. The baseline history is set (baselineHist).
7. The default student record is saved in the defaultSR field of the StudentRecordSet and in the database (saveSR).
8. The baseline statistics are announced (announceStats).

Value

This function is called for its side effects.

Author(s)

Russell Almond

References

Almond, Mislevy, Steinberg, Yan and Williamson (2015). *Bayesian Networks in Educational Assessment*. Springer. Especially Chapter 13.

See Also

Classes: BNEngine, StudentRecord, StudentRecordSet, PnetWarehouse

Functions: clearSRs, PnetCompile, updateStats, baselineHist, saveSR, announceStats

Examples

```
## Requires database setup, also PNetica
library(RNetica)  ## Must load to setup Netica DLL
app <- "ecd://epls.coe.fsu.edu/EITest"
sess <- RNetica::NeticaSession()
RNetica::startSession(sess)

config.dir <- file.path(library(help="Peanut")$path, "auxdata")
net.dir <- file.path(library(help="PNetica")$path,"testnets")

netman <- read.csv(file.path(config.dir, "Mini-PP-Nets.csv"),
                   row.names=1, stringsAsFactors=FALSE)
stattab <- read.csv(file.path(config.dir, "Mini-PP-Statistics.csv"),
                   as.is=TRUE)

Nethouse <- PNetica::BNWarehouse(netman,session=sess,
             address=net.dir)

cl <- new("CaptureListener")
listeners <- list("cl"=cl)

ls <- ListenerSet(sender= paste("EAEngine[",app,"]"),
                  dburi="", listeners=listeners)

eng <- BNEngineNDB(app=app,warehouse=Nethouse,
                    listenerSet=ls,manifest=netman,
                    profModel="miniPP_CM",
                    histNodes="Physics",
                    statmat=stattab,
                    activeTest="EAActive.txt")

## Standard initialization methods.
loadManifest(eng,netman)
eng$setHistNodes("Physics")
configStats(eng,stattab)
setupDefaultSR(eng)

defrec <- eng$studentRecords()$defaultSR
```

```
stopifnot(uid(defrec)=="*DEFAULT*", app(defrec)==app(eng),
          context(defrec)=="*Baseline*",
          PnetName(sm(defrec))==eng$profModel)
```

---

sm *Access the student model (Pnet) associated with a studnet record*

Description

A characteristic of the EABN model is that each code StudentRecord is associated with a *student model*–a Pnet which tracks our knowledge about the student's knowledge skills and abilities. The function sm accesses the net.

Usage

```
sm(x)
sm(x) <- value
```

Arguments

x	An object of class StudentRecord whose student model will be accessed.
value	A Pnet object which will be the new student model.

Value

The function sm returns an object which implements the Pnet protocol, or none is the student model has not been generated.

The setter version returns the student record.

Author(s)

Russell Almond

See Also

fetchSM, unpackSM, setupDefaultSR

Examples

```
library(PNetica)

##Start with manifest
sess <- NeticaSession()
startSession(sess)
```

```
## BNWarehouse is the PNetica Net Warehouse.
## This provides an example network manifest.
config.dir <- file.path(library(help="Peanut")$path, "auxdata")
netman1 <- read.csv(file.path(config.dir,"Mini-PP-Nets.csv"),
                    row.names=1, stringsAsFactors=FALSE)
net.dir <- file.path(library(help="PNetica")$path, "testnets")
Nethouse <- BNWarehouse(manifest=netman1,session=sess,key="Name",
                        address=net.dir)

dsr <- StudentRecord("*DEFAULT*",app="ecd://epls.coe.fsu.edu/P4Test",
                     context="*Baseline*")
sm(dsr) <- WarehouseSupply(Nethouse,"miniPP_CM")
PnetCompile(sm(dsr))
```

`stat` *Access statistics from a Student Record*

Description

These functions access the `stats` field of a `StudentRecord` object. The function `stat` accesses a single statistics and `stats` returns all of the statistics. The function `statNames` returns the names of the available statistics.

Usage

```
stat(sr, name)
stats(x)
statNames(sr)
```

Arguments

`sr, x`	A `StudentRecord` object whose statsitics are to be accessed.
`name`	A character object giving the name of the specific statististic to access.

Value

The function `stat` returns the value of a single statistic, which could be numeric, character or something else.

The function `stats` returns a named list of statistics.

The function `statNames` returns a character vector.

Author(s)

Russell Almond

See Also

StudentRecord for the student record class.

Statistic for statistic objects which return the statistics.

Examples

```
stats <- list(Physics_EAP=0,EnergyTransfer_EAP=.15,
              Physics_Margin=c(High=1/3,Medium=1/3,
                               Low=1/3))

dsr <- StudentRecord("*DEFAULT*",app="ecd://epls.coe.fsu.edu/P4Test",
                     context="*Baseline*",stats=stats)

stats(dsr)
stopifnot(all.equal(stats,stats,tolerance=.0002))

statNames(dsr)
stopifnot(all(statNames(dsr)==names(stats)))

stat(dsr,"Physics_Margin")
stopifnot(all.equal(stat(dsr,"Physics_Margin"),stats[[3]],tolerance=.0002))
```

StudentRecord *Constructor for* StudentRecord *object*

Description

This is the constructor for a StudentRecord object. Basically, this is a wrapper around the studnet model for the appropriate user, with meta-data about the evidence that has been absorbed.

Usage

```
StudentRecord(uid, context = "", timestamp = Sys.time(), smser = list(), sm = NULL, stats = list(), hist
```

Arguments

uid	A user identifier for the student/player.
context	An identifer for the scoring context/window.
timestamp	Timestamp of the last evidence set absorbed for this user.
smser	A serialized Bayesian network (see WarehouseUnpack).
sm	A Pnet containing the student model (or NULL if it has not been initialized.
stats	A list of statistics calculated for the model.

hist	A list of node histories for the measured nodes.
evidence	A character vector of ids for the absorbed evidence sets.
app	A guid (string) identifying the application.
seqno	A sequence number, basically a count of absorbed evidence sets.
prev_id	The database ID of the previous student model.

Value

An object of class StudentRecord.

Author(s)

Russell Almond

See Also

StudentRecord

Examples

```
library(PNetica)

##Start with manifest
sess <- NeticaSession()
startSession(sess)

## BNWarehouse is the PNetica Net Warehouse.
## This provides an example network manifest.
config.dir <- file.path(library(help="Peanut")$path, "auxdata")
netman1 <- read.csv(file.path(config.dir,"Mini-PP-Nets.csv"),
                    row.names=1, stringsAsFactors=FALSE)
net.dir <- file.path(library(help="PNetica")$path, "testnets")
Nethouse <- BNWarehouse(manifest=netman1,session=sess,key="Name",
                        address=net.dir)

dsr <- StudentRecord("*DEFAULT*",app="ecd://epls.coe.fsu.edu/P4Test",
                     context="*Baseline*")
sm(dsr) <- WarehouseSupply(Nethouse,"miniPP_CM")
PnetCompile(sm(dsr))

## dsr <- updateStats(eng,dsr)
statmat <- read.csv(file.path(config.dir,"Mini-PP-Statistics.csv"),
                    stringsAsFactors=FALSE)
rownames(statmat) <- statmat$Name
statlist <- sapply(statmat$Name,function (st)
    Statistic(statmat[st,"Fun"],statmat[st,"Node"],st))
names(statlist) <- statmat$Name
dsr@stats <- lapply(statlist,
                    function (stat) calcStat(stat,sm(dsr)))
```

```
names(dsr@stats) <- names(statlist)
stat(dsr,"Physics_EAP")
stat(dsr,"Physics_Margin")

## dsr <- baselineHist(eng,dsr)

dsr@hist <- lapply(c("Physics"),
                   function (nd)
                   EABN:::uphist(sm(dsr),nd,NULL,"*Baseline*"))
names(dsr@hist) <- "Physics"
history(dsr,"Physics")

## Serialization and unserialization
dsr.ser <- as.json(dsr)

dsr1 <- parseStudentRecord(fromJSON(dsr.ser))
dsr1 <- fetchSM(dsr1,Nethouse)

### dsr and dsr1 should be the same.
stopifnot(
 app(dsr)==app(dsr1),
 uid(dsr)==uid(dsr1),
 context(dsr)==context(dsr1),
 all.equal(timestamp(dsr),timestamp(dsr1)),
 all.equal(seqno(dsr),seqno(dsr1)),
 all.equal(stats(dsr),stats(dsr1),tolerance=.0002),
 all.equal(history(dsr,"Physics"),history(dsr1,"Physics")),
 PnetName(sm(dsr)) == PnetName(sm(dsr))
)
```

`StudentRecord-class` *Class* `"StudentRecord"`

Description

This is a wrapper for the Bayesian network information for a particular student. It contains a local copy of the Bayesian network.

Objects from the Class

Objects can be created by calls to the function `StudentRecord`, `uid`, `context`, `timestamp`, `smser`, `sm`, `stats`, `hist`, `e`

Slots

`_id`: Object of class `"character"` The `mongo` ID of the object, empty character if it has not been saved in the database. If Mongo is not being used, this field can be used for other kinds of IDs.

`app`: Object of class `"character"` that gives the identifier for the application this record is used with.

`uid`: Object of class `"character"` which is the unique identifier for the user (student, player).

`context`: Object of class `"character"` which identifies the scoring context (scoring window).

`evidence`: Object of class `"character"` giving the IDs of the evidence sets applied to this student model.

`timestamp`: Object of class `"POSIXt"` giving the timestamp of the last evidence set applied to this model.

`sm`: Object of class `"Pnet"`, the actual student model (or `NULL` if it is not yet built).

`smser`: Object of class `"list"` the serialized student model.

`seqno`: Object of class `"integer"` a sequence number, that is the number of evidence sets applied.

`stats`: Object of class `"list"` the most recent statistics generated from this model.

`hist`: Object of class `"list"` list of history lists for the designed history variables. There is one element for each history variable.

`prev_id`: Object of class `"character"` the Mongo ID of the previous student model.

Methods

app `signature(x = "StudentRecord")`: returns the application id associated with this record.

as.jlist `signature(obj = "StudentRecord", ml = "list")`: serialized the record as JSON

context `signature(x = "StudentRecord")`: return the context (scoring window) identifier associated with the last processed evidence set.

evidence `signature(x = "StudentRecord")`: returns the ids of the aborbed evidence sets.

evidence<- `signature(x = "StudentRecord")`: sets the ids of the aborbed evidence sets.

histNames `signature(sr = "StudentRecord")`: returns the names of the history variables.

history `signature(sr = "StudentRecord", name = "character")`: returns the history list for the variable.

seqno `signature(x = "StudentRecord")`: returns the sequence number for this record.

seqno<- `signature(x = "StudentRecord")`: sets the sequence number for this record.

show `signature(object = "StudentRecord")`: prints the record.

sm `signature(x = "StudentRecord")`: returns the Bayes net (`Pnet`) associated with this record.

sm<- `signature(x = "StudentRecord", value="ANY")`: sets the Bayes net (`Pnet`) associated with this record.

stat `signature(sr = "StudentRecord", name = "character")`: returns the current value of the named statistics.

statNames `signature(sr = "StudentRecord")`: returns the names of the statistics.

stats `signature(x = "StudentRecord")`: returns all of the statistics.

timestamp `signature(x = "StudentRecord")`: returns the timestamp of the last absorbed evidence set.

toString `signature(x = "StudentRecord")`: creates a printed representation.

uid `signature(x = "StudentRecord")`: returns the ID for the student/player.

Author(s)

Russell Almond

References

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Williamson, D.M. and Yan, D. (2015) *Bayesian Networks in Educational Assessment.* Springer. Chapter 13.

See Also

StudentRecord, EvidenceSet, StudentRecordSet

Examples

```
showClass("StudentRecord")
```

StudentRecordSet *Constructor for "StudentRecordSet" class*

Description

A StudentRecordSet is a collection of collection of StudentRecord objects. It is always connected to a PnetWarehouse and could be connected to a database as well.

Usage

```
StudentRecordSet(app = "default", warehouse = NULL,
  dburi = "mongodb://localhost", dbname = "EARecords", ...)
```

Arguments

app	A character scalar providing a guid for the application.
warehouse	An object of type PnetWarehouse that contains already built student models.
dburi	A character scalar giving the URI for the database, or an empty character string if the record set is not connected to the database.
dbname	A character scalar giving the name of the database where records are stored.
...	Other arguments for future extensions.

Details

A StudentRecordSet is a collection of student recrods. It contains a PnetWarehouse which contains the student models and possibly a database containing the student records.

The StudentRecordSet operates in two modes, depending on the value of dburi. If dburi references a MongoDB-class database, then the StudentRecordSet set will save student records (including serialized Bayes nets) to the database and restore them on demmand. This facilitates scoring across several sessions.

If the dburi argument is an empty string or NULL no database connection will be created. Instead, the calls to the getSR function should pass in a serialized version of the student record function. If no serialized record is available, a new record will be created.

Value

An object of class StudentRecordSet.

Author(s)

Russell Almond

See Also

StudentRecordSet, StudentRecord, getSR, saveSR, newSR, clearSRs

Examples

```
library(PNetica)

##Start with manifest
sess <- NeticaSession()
startSession(sess)

## BNWarehouse is the PNetica Net Warehouse.
## This provides an example network manifest.
config.dir <- file.path(library(help="Peanut")$path, "auxdata")
netman1 <- read.csv(file.path(config.dir,"Mini-PP-Nets.csv"),
                    row.names=1, stringsAsFactors=FALSE)
net.dir <- file.path(library(help="PNetica")$path, "testnets")
Nethouse <- BNWarehouse(manifest=netman1,session=sess,key="Name",
                        address=net.dir)

## Setup to test without Mongo
SRS <- StudentRecordSet(app="Test",warehouse=Nethouse,
                         dburi="")
stopifnot(is.null(SRS$recorddb()))

## Setup default SR
dsr <- StudentRecord("*DEFAULT*",app="Test",
                     context="*Baseline*")
sm(dsr) <- WarehouseSupply(Nethouse,"miniPP_CM")
```

```
PnetCompile(sm(dsr))

## dsr <- updateStats(eng,dsr)
statmat <- read.csv(file.path(config.dir,"Mini-PP-Statistics.csv"),
                    stringsAsFactors=FALSE)
rownames(statmat) <- statmat$Name
statlist <- sapply(statmat$Name,function (st)
    Statistic(statmat[st,"Fun"],statmat[st,"Node"],st))
names(statlist) <- statmat$Name
dsr@stats <- lapply(statlist,
                    function (stat) calcStat(stat,sm(dsr)))
names(dsr@stats) <- names(statlist)

dsr@hist <- lapply(c("Physics"),
                     function (nd)
                     EABN:::uphist(sm(dsr),nd,NULL,"*Baseline*"))
names(dsr@hist) <- "Physics"

SRS$defaultSR <- dsr
saveSR(SRS, dsr)

## Make a new Student Record for a student.
sr1 <- newSR(SRS,"S1")
stopifnot(uid(sr1)=="S1",app(sr1)==app(dsr),
          all.equal(stats(dsr),stats(sr1),.0002))


sr1a <- getSR(SRS,"S1")

clearSRs(SRS)
```

`StudentRecordSet-class` *Class* `"StudentRecordSet"`

Description

This class provides a collection of student records. Optionally, it can be hitched to a database so that student can be saved and restored across scoring sessions.

Details

The `StudentRecordSet` exists to hold a collection of `StudentRecord` objects. If, when constructed, the record set is passed information about a database, the record set is stored in the database. If not, it is merely stored in memory. The database version, in particular, allows restoring the object from memory. The primary key for the student record in the database is the `app` ID (which is a field in the record set) and the `uid` which is passed through the `getSR` method.

The method getSR takes different arguments based on which version is passed. In particular, the ser argument allows a serialized (JSON) version of the data to be passed in. In particular, getSR will do one of the following things (in order of priority):

1. If the ser argument is supplied, the student record will be restored from this.
2. If the StudentRecordSet is connected to a database, then the student record is restored from information in the database, based on the uid argument and the app field.
3. A new student record is created for the uid.

The record set also contains a link to a PnetWarehouse which it uses to try and find the Pnet associated with the StudentRecord. If the Pnet already exists in the warehouse, it is just connected to the fetched record. If not, then it is restored from a serialized version either from the passed in serialized record, or from the serialized Pnet in the database.

Extends

All reference classes extend and inherit methods from "envRefClass".

Methods

app signature(x = "StudentRecordSet"): Returns the application this record set is associated with.

getSR signature(srs = "StudentRecordSet", uid="ANY",ser="character"): Returns the student record for the specified ID. If ser is supplied it should be a json list object containing the student record.

newSR signature(srs = "StudentRecordSet", uid="character"): Creates a new Student Record for the specified ID by cloning the default student record.

saveSR signature(srs = "StudentRecordSet"): If connected to a database, the SR is saved to the database.

clearSR signature(srs = "StudentRecordSet"): If connected to a database, the SR in the database are cleared.

Fields

app: Object of class character which contains the application identifier

dbname: Object of class character which contains the name of the database.

dburi: Object of class character containing the URI for connecting to the database, for example "mongodb://localhost:271017". To create a record set not connecteded to the database, set this value to character().

db: Object of class MongoDB a connection to the database or NULL if the object is not connected to the database. Users should call the recorddb() function rather than access this field directly.

warehouse: Object of class PnetWarehouse which contains already loaded nets.

defaultSR: Object of class StudentRecord or NULL. This is the default student record which is cloned to create new studnet records.

Class-Based Methods

`initialize(app, dbname, dburi, db, warehouse, ...)`: Initializes the student record set.

`recorddb()`: Returns the database handle (if connected to a database) or `NULL` if not connected to a database. Note that this initializes the database the first time it is called, so it should be called rather than accessing the `db` field directly.

`clearAll(clearDefault=FALSE)`: Clears all records from the database and the warehouse. If `clearDefault==FALSE`, then the default record is not cleared.

Author(s)

Russell Almond

See Also

`StudentRecordSet` for the constructor. `StudentRecord` for the contained objects.

`PnetWarehouse` and `Pnet` for information about the contained Bayesian networks.

`BNEngine` for the engine that holds it.

Examples

```
showClass("StudentRecordSet")
```

---

updateHist *Update the node history in a student record*

Description

The `StudentRecord` object can track the history of zero or more `Pnode` in the student model (`sm`). The history is a data frame with columns corresponding to the states of the variables and the rows corresponding to the `EvidenceSet`s absorbed into the student record. The function `updateHist` add a new row to each history corresponding to the evidence set. The function `baselineHist` creates the initial row.

Usage

```
updateHist(eng, rec, evidMess, debug = 0)
baselineHist(eng, rec)
```

Arguments

`eng`	The `BNEngine` controlling the operation.
`rec`	The `StudentRecord` which will be updated.
`evidMess`	The `EvidenceSet` which has just been added to the student model using `updateSM`.
`debug`	An integer flag. If bigger than 1, then a call to `recover` will be made inside the function call.

Details

A history tracks a single node in the student model as it changes in response to the incomming evidence sets. The history for a node is data frame with columns representing variable states and rows representing evidence sets (evidence from different scoring windows or tasks).

The function `baselineHist` is called as part of `setupDefaultSR`. This initializes a history data frame for each node in the `histNodes` field of the `BNEngine`. It inserts a first row, which is always given the name "*Baseline*". The values in the first row are the marginal distribution of those nodes (`PnodeMargin`).

The function `updateHist` adds row to each history table. The name of the row corresponds to the `context` field of the `EvidenceSet`. The value is the curent marginal distribution for the history nodes.

The function `history` retrieves the history. The functions `woeHist` and `woeBal` in the `CPTtools-package` describe possible applications for the history function.

Value

Both functions return the modified `StudentRecord`

Note

With the Netica implementation, the student model needs to be compiled (`PnetCompile(sm(rec))`) before the `baselineHist` function is run.

This is probably true of `updateHist` as well, but `updateSM` recompiles the network.

Author(s)

Russell Almond

References

Madigan, Mosurski and Almond, (1997). Graphical explanation in belief networks. *Journal of Computational and Graphical Statistics,* **6**, 160–181.

Almond, Kim, Shute and Ventura (2013). Debugging the evidence chain. *Proceeings of the 2013 UAI Application Workshops (UAI2013AW).* 1–10. CEUR workshop proceedings, vol 1024. http://ceur-ws.org/Vol-1024/paper-01.pdf

See Also

Classes: `BNEngine`, `EvidenceSet` `StudentRecord`

Functions in EABN: `accumulateEvidence`, `updateStats`, `updateSM`, `history`

Peanut Functions: `PnodeMargin`

CPTtools Functions `woeHist`, `woeBal`

Examples

```
## Requires database setup, also PNetica
library(RNetica)  ## Must load to setup Netica DLL
app <- "ecd://epls.coe.fsu.edu/EITest"
sess <- RNetica::NeticaSession()
RNetica::startSession(sess)

config.dir <- file.path(library(help="Peanut")$path, "auxdata")
net.dir <- file.path(library(help="PNetica")$path,"testnets")

netman <- read.csv(file.path(config.dir, "Mini-PP-Nets.csv"),
                   row.names=1, stringsAsFactors=FALSE)
stattab <- read.csv(file.path(config.dir, "Mini-PP-Statistics.csv"),
                   as.is=TRUE)

Nethouse <- PNetica::BNWarehouse(netman,session=sess,
             address=net.dir)

cl <- new("CaptureListener")
listeners <- list("cl"=cl)

ls <- ListenerSet(sender= paste("EAEngine[",app,"]"),
                  dburi="", listeners=listeners)

eng <- BNEngineNDB(app=app,warehouse=Nethouse,
                    listenerSet=ls,manifest=netman,
                    profModel="miniPP_CM",
                    histNodes="Physics",
                    statmat=stattab,
                    activeTest="EAActive.txt")

## Standard initialization methods.
loadManifest(eng,netman)
eng$setHistNodes(character())
configStats(eng,stattab)
setupDefaultSR(eng)



sr1 <- getRecordForUser(eng,"S1")
history(sr1,"Physics")
stopifnot(is.null(history(sr1,"Physics")))

## Now set up history.
eng$setHistNodes("Physics")
PnetCompile(sm(sr1))
sr1 <- baselineHist(eng,sr1)
history(sr1,"Physics")
stopifnot(nrow(history(sr1,"Physics"))==1L)
```

```
e1 <- EvidenceSet(uid="S1",app="Test",context="PPcompEM",
                  obs=list("CompensatoryObs"="Right"))
sr1 <- updateSM(eng,sr1,e1)
sr1 <- updateHist(eng,sr1,e1)

e2 <- EvidenceSet(uid="S1",app="Test",context="PPconjEM",
                  obs=list("ConjunctiveObs"="Wrong"))
sr1 <- updateSM(eng,sr1,e2)
sr1 <- updateHist(eng,sr1,e2)

history(sr1,"Physics")
stopifnot(nrow(history(sr1,"Physcis"))==3L)
woeHist(history(sr1,"Physics"),pos="High",neg=c("Medium","Low"))
```

---

updateSM *Updates the Student model with additional evidence.*

Description

This function is the core of the EABN algorithm. It finds and attaches the evidence model to the student model, enters the findings from the evidence message, and then detaches the evidence model, leaving the student model updated.

Usage

```
updateSM(eng, rec, evidMess, debug = 0)
```

Arguments

eng	The BNEngine supervising the opeeration.
rec	The StudentRecord for the student in question.
evidMess	The EvidenceSet containing the new evidence.
debug	An integer describing how much debugging to do. If set to a number greater than 1, it will issue a call to recover at various stages to aid in debugging models.

Details

The update algorithm performs the following step:

1. Finds the evidence model by name according to the context field of the EvidenceSet. See WarehouseSupply.
2. Adjoins the sm of the student record with the evidence model and compiles the modified network. See PnetAdjoin and PnetCompile.
3. Loops over the observables in the evidence set, if they correspond to nodes in the evidence model, then instantiate their values using PnodeEvidence.
4. Detatch the evidence model and recompile the network. See PnetDetach.

Value

The updated student record is returned.

Author(s)

Russell Almond

References

Almond, Mislevy, Steinberg, Yan and Williamson (2015). *Bayesian Networks in Educational Assessment*. Springer. Especially Chapters 5 and 13.

See Also

Classes: BNEngine, PnetWarehouse, StudentRecord, EvidenceSet, Pnet

Functions in EABN: accumulateEvidence, updateHist, updateStats, getRecordForUser

Peanut Functions: WarehouseSupply, PnetAdjoin, PnetCompile, PnetDetach, PnodeEvidence

Examples

```
## Requires database setup, also PNetica
library(RNetica)  ## Must load to setup Netica DLL
app <- "ecd://epls.coe.fsu.edu/EITest"
sess <- RNetica::NeticaSession()
RNetica::startSession(sess)

config.dir <- file.path(library(help="Peanut")$path, "auxdata")
net.dir <- file.path(library(help="PNetica")$path,"testnets")

netman <- read.csv(file.path(config.dir, "Mini-PP-Nets.csv"),
                   row.names=1, stringsAsFactors=FALSE)
stattab <- read.csv(file.path(config.dir, "Mini-PP-Statistics.csv"),
                   as.is=TRUE)

Nethouse <- PNetica::BNWarehouse(netman,session=sess,
             address=net.dir)

cl <- new("CaptureListener")
listeners <- list("cl"=cl)

ls <- ListenerSet(sender= paste("EAEngine[",app,"]"),
                  dburi="", listeners=listeners)

eng <- BNEngineNDB(app=app,warehouse=Nethouse,
                    listenerSet=ls,manifest=netman,
                    profModel="miniPP_CM",
                    histNodes="Physics",
                    statmat=stattab,
                    activeTest="EAActive.txt")
```

```
## Standard initialization methods.
loadManifest(eng,netman)
eng$setHistNodes("Physics")
configStats(eng,stattab)
setupDefaultSR(eng)

sr1 <- getRecordForUser(eng,"S1")
PnetCompile(sm(sr1))
eap1 <- PnodeEAP(sm(sr1),PnetFindNode(sm(sr1),"Physics"))

e1 <- EvidenceSet(uid="S1",app="Test",context="PPcompEM",
                  obs=list("CompensatoryObs"="Right"))

sr1a <- updateSM(eng,sr1,e1)
eap1a <- PnodeEAP(sm(sr1),PnetFindNode(sm(sr1),"Physics"))
## This should have changed.
stopifnot(abs(eap1-eap1a) > .001)
```

---

updateStats *Recalculates statistics for changed student model.*

Description

When the student model of a StudentRecord changes, because the function updateSM has been run, the statistics need to be recalculated. The function updateStats recalculates the statistics. The function announceStats lets the listeners know that new statistics are available for this user.

Usage

```
updateStats(eng, rec, debug = 0)
announceStats(eng, rec)
```

Arguments

eng	A BNEngine controlling the operation.
rec	A StudentRecord, particularly, one that has just been updated via a call to updateSM.
debug	An integer flag. If the value is greater than 1, there will be a call recover inside of the call.

Details

The BNEngine contains a number of Statistic objects. Every time the student model (sm) of the StudentRecord changes, the stats of the record need to be updated as well.

The function updateStats simply loops through the statistic collection and calculates the new values. The corresponding field of the student record is then updated.

The function announceStats takes the new statistic values and generates a P4Message containing the new statistics. This is sent to all of the Listener objects in the ListenerSet attached to the engine.

The function stats returns the latest statistics from the student record.

Value

The function updateStats returns the updates StudentRecord object.

The function announceStats is called for its side effects. Its return value should not be used.

Author(s)

Russell Almond

References

Almond, Mislevy, Steinberg, Yan and Williamson (2015). *Bayesian Networks in Educational Assessment*. Springer. Especially Chapters 5 and 13.

See Also

Classes: BNEngine, ListenerSet StudentRecord, Statistic, P4Message

Functions in EABN: accumulateEvidence, updateHist, updateSM, stats

Peanut Functions: calcStat

Proc4Functions notifyListeners

Examples

```
## Requires database setup, also PNetica
library(RNetica)  ## Must load to setup Netica DLL
app <- "ecd://epls.coe.fsu.edu/EITest"
sess <- RNetica::NeticaSession()
RNetica::startSession(sess)

config.dir <- file.path(library(help="Peanut")$path, "auxdata")
net.dir <- file.path(library(help="PNetica")$path,"testnets")

netman <- read.csv(file.path(config.dir, "Mini-PP-Nets.csv"),
                   row.names=1, stringsAsFactors=FALSE)
stattab <- read.csv(file.path(config.dir, "Mini-PP-Statistics.csv"),
                   as.is=TRUE)
```

```
Nethouse <- PNetica::BNWarehouse(netman,session=sess,
             address=net.dir)

cl <- new("CaptureListener")
listeners <- list("cl"=cl)

ls <- ListenerSet(sender= paste("EAEngine[",app,"]"),
                  dburi="", listeners=listeners)

eng <- BNEngineNDB(app=app,warehouse=Nethouse,
                     listenerSet=ls,manifest=netman,
                     profModel="miniPP_CM",
                     histNodes="Physics",
                     statmat=stattab,
                     activeTest="EAActive.txt")

## Standard initialization methods.
loadManifest(eng,netman)
eng$setHistNodes("Physics")
configStats(eng,stattab)
setupDefaultSR(eng)

sr0 <- getRecordForUser(eng,"S1")

eap0 <- stat(sr0,"Physics_EAP")

e1 <- EvidenceSet(uid="S1",app="Test",context="PPcompEM",
                  obs=list("CompensatoryObs"="Right"))


sr1 <- updateRecord(sr0,e1)
sr1 <- updateSM(eng,sr1,e1)
sr1 <- updateStats(eng,sr1)
eap1 <- stat(sr1,"Physics_EAP")

## This should have changed.
stopifnot(abs(eap1-eap0) > .001)

announceStats(eng,sr1)
## Look at the resulting message.
cl$lastMessage()
details(cl$lastMessage())
stopifnot(uid(cl$lastMessage())=="S1",context(cl$lastMessage())=="PPcompEM")
```

Index