RNetica

Quick Start Guide

Downloading

- http://pluto.coe.fsu.edu/RNetica/
- Two Packages:
  - RNetica – R to Netica link
  - CPTtools – Design patterns for CPTs
- Source & binary version (Win 64, Mac OS X)
  - Binary versions include Netica.dll/libNetica.so
  - Source version need to download from http://www.norsys.com/ first

License

- R – GPL-3 (Free and open source)
- RNetica – Artistic (Free and open source)
- Netica.dll/libNetica.so – Commercial (open API, but not open source)
  - Free Student/Demo version
    - Limited number of nodes
    - Limited usage (education, evaluation of Netica)
  - Paid version (see http://www.norsys.com/ for price information)
    - Need to purchase API not GUI version of Netica
    - May want both (use GUI to visualize networks build in RNetica)

Installing the License Key

- When you purchase a license, Norsys will send you a license key. Something that looks like: "+Course/FloridaSU/Ex15-05-30,120,310/XXXXX" (Where I've obscured the last 5 security digits)
- To install the license key, start R in your project directory and type:

```
> NeticaLicenseKey <- "+Course/FloridaSU/Ex15-05-30,120,310/XXXXX"
> q("yes")
```

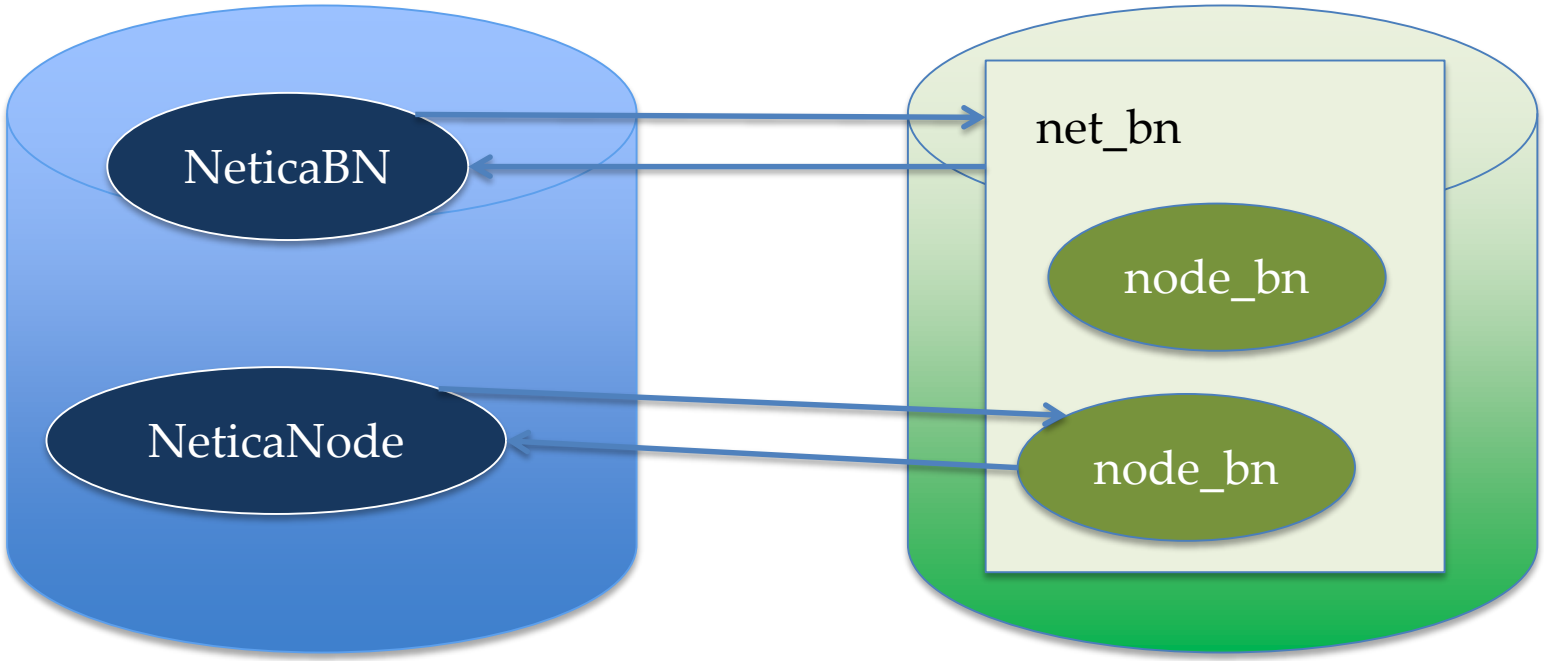
- Restart R and type

```
> library(RNetica)
```

- If license key is not installed, then you will get the limited/student mode. Most of these examples will run

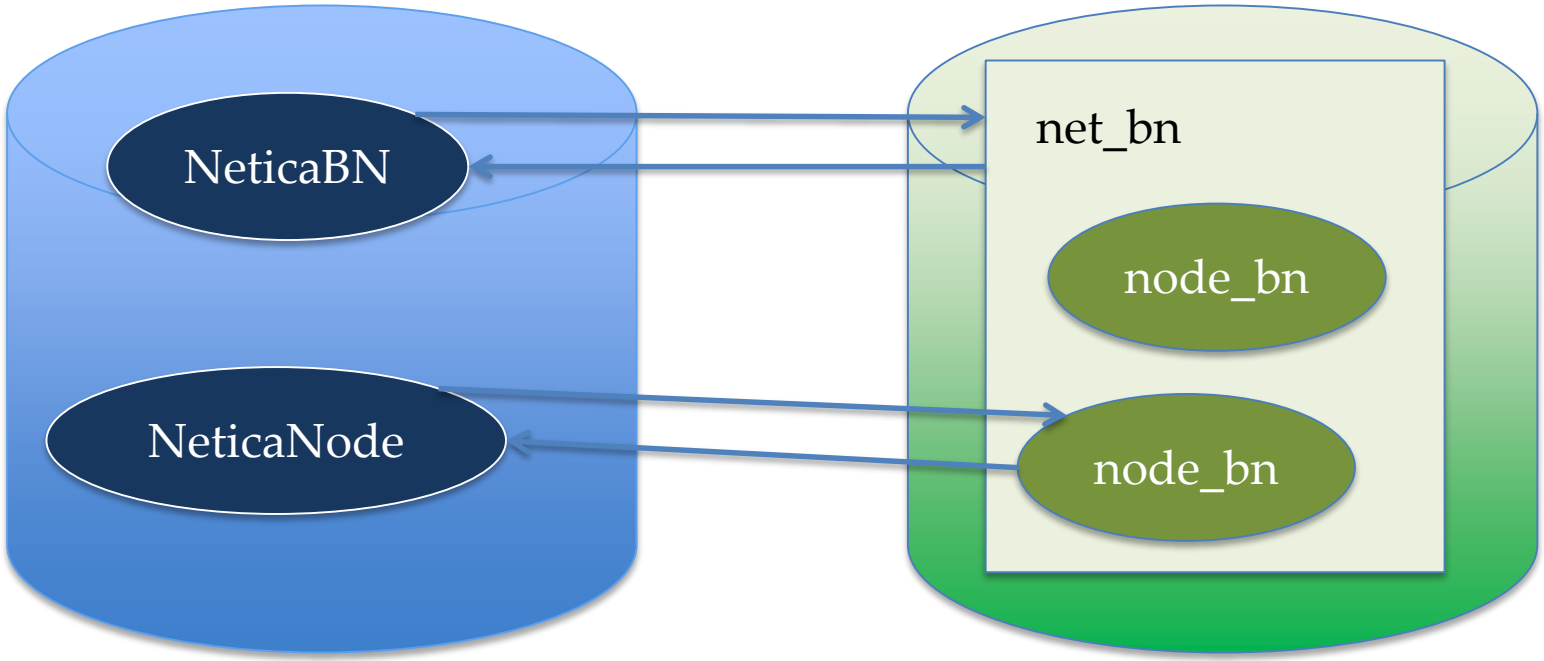
The R heap and the Netica heap

- R and Netica have two different workspaces (memory heaps)
- R workspace is saved and restored automatically when you quick and restart R.
- Netica heap must be reconnected manually.



Active and Inactive pointers

- When RNetica creates/finds a Netica object it creates a corresponding R object
- If the R object is active then it points to the Netica object, and the Netica object points back at it
- If the pointer gets broken (saving & restarting R, deleting the network/node) then the R object becomes inactive.
- The function is.active(nodeOrNet) test to see if the node/net is active



Tips for Dealing with Memory

- Call **`DeleteNetwork(net)`** when finished finished with **`net`** to deactivate nodes
- Save network with **`WriteNetworks(net,path)`** before quitting R
- **`net <- ReadNetworks(net)`** will read network from wherever it was last saved
- Refresh nodes with **`nodes <- NetworkAllNodes(net)`**
  - Then access individual nodes by **`nodes$name`**
- Use nodesets to mark groups of nodes for retrieval (e.g., observables and reporting variables)

Example IRT5 net

```
## Script for reading the IRT 5 network
## Read from file
irt5 <- ReadNetworks("T:/Documents/R/win-
   library/2.13/RNetica/sampleNets/IRT5.dne"
   )
## Find the proficiency node
irt5.theta <- NetworkFindNode(irt5,"Theta")
## Find the observable nodes
irt5.x <- NetworkFindNode(irt5,
              paste("Item",1:5,sep="_"))
```

Netica Names

- Certain Netica objects (particularly, networks, nodes and node states) have names which must follow general variable naming conventions:
  - Start with a letter
  - Only contain letters, numbers and "_"
  - No more than 40 chars
- Function **`is.IDname(str)`** checks rule, function **`as.IDname(str)`** coerces **`str`** to follow rules
- Be wary of long names, Netica sometimes internally adds symbols to names (particularly names of stub variables) so that names which are too long can fail at a later date.
- In addition to the name, networks, nodes and states have a title which does not have these restrictions.

Conditional Probability Tables (CPTs)

Conditional Probability Frame (CPF)

```
> arf
   A  B C.c1 C.c2 C.c3 C.c4
1 a1 b1    1    7   13   19
2 a1 b2    2    8   14   20
3 a1 b3    3    9   15   21
4 a2 b1    4   10   16   22
5 a2 b2    5   11   17   23
6 a2 b3    6   12   18   24
```

A & B are parents, C is child

Subclass of data.frame

Conditional Probability Array (CPA)

```
> arfa
, , C = c1

    B
A    b1 b2 b3
  a1  1  2  3
  a2  4  5  6

, , C = c2

    B
A    b1 b2 b3
  a1  7  8  9
  a2 10 11 12

, , C = c3

    B
A    b1 b2 b3
  a1 13 14 15
  a2 16 17 18

, , C = c4

    B
A    b1 b2 b3
  a1 19 20 21
  a2 22 23 24

attr(,"class")
[1] "CPA"
"array"
```

CPAs and CPFs

- Use **`as.CPA()`** and **`as.CPF()`** to translate between the representations
- The expression **`NodeProbs(node)`** gets (and with `<-` sets) the CPT of node as an CPA
- The expression **`node[]`** gets (and with `<-` sets) the CPT of node as a CPF
- Internally **`node[selection] <- value`** is more efficient than **`NodeProbs(node)[selection] <- value`** if only a subset of the CPT is to be change.
- CPTtools package exists to collect design patterns for CPFs

Extract.NeticaNode (node[])

- **`node[]`** extracts/sets the whole table.
- **`node[[]]`** extracts (no setter) only the numeric part
- **`node[selection]`** has several effects depending on **`selection`** (see **`help("Extract.NeticaNode")`**)
  - List of integers or state names selects single roll
  - Vector or blank or "*" or EVERY_STATE in list selects multiple rows
  - Named list selects parents by name
  - Single integer selects a single row
  - Data frame argument selects multiple rows
  - Follows R convention of starting indexes at 1 (not Netica convention of starting indexes at 0).

node[selection] <- value

- If value is a single number it is replicated as needed
- If value has one fewer columns than there are number of states, the last column is calculated by normalization (very handy for binary variables)
- If value is the name of a state it will generate a logical function for that row.
- In the expression **`node[] <- value`**, if value is a data.frame, then only the selected rows will be effected
- See **`help("Extract.NeticaNode")`** for details

Nodesets

- Netica maintains a list of node sets (which is basically just a vector of strings)
- **`NodeSets(node)`** returns the list of sets a node belongs to
- Use that function with <- to change the node sets
  - **`NodeSets(node) <- c("setname",NodeSets(node)`**
- The expression **`NetworkNodesInSet(net,name)`** returns a list of nodes in the set.
- Typical node set names: Observables, Proficiencies, ReportingVars

Node sets are useful in restoring networks

```
miniACED <- ReadNetworks(miniACED)
reportingVars <-
     NetworkNodesInSet(miniACED,"ReportingVars")
for (node in reportingVars) {
  cat("\nBeliefs for node",NodeName(node),"\n")
  print(NodeBeliefs(node))
}
```

NodeKinds

`NodeKind(node)` gets (sets with <-)

- "Nature" – regular random variable
- "Decision" – deterministic, used with decision nets
- "Utility" – continuous, used with decision nets
- "Constant" – usually continuous, used in formulae
- "Stub" – created when edge is disconnected from a node

Continuous Nodes

- Netica is mostly designed around discrete nodes but allows continuous nodes
- **`NodeLevels(node)`** when node is continuous gets/sets cut points for discretizing the node (including upper and lower endpoints)
- Most uses of continuous nodes require them to be discretized, exceptions:
  - Used as constants in formulas
  - Used as utilities in decision nets

Numeric Nodes

- Using **`NodeLevels(node)`** with a discrete node associates numeric values with each state of the node.
- *Numeric nodes* are nodes that are either discrete nodes made numeric, or continuous nodes made discrete
- Certain functions require these numeric nodes

RNETICA FUNCTIONS GROUPED BY TOPIC

R-Netica interface

- StartNetica() ## called by library(RNetica)
- StopNetica()
- NeticaVersion
- ReportErrors() ## called by most RNetica functions
- ClearAllErrors()
- is.active(obj)
- is.IDname(str), as.IDname(str)

Network Management

- CreateNetwork(name)
- DeleteNetwork(net)
- GetNthNetwork(n)
- GetNameNetworks(names)
- CopyNetworks(nets)
- WriteNetworks(nets,paths)
- ReadNetwors(paths)
- GetNetworkFilename(net)
- is.NeticaBN(obj)

Network Metadata

- NetworkName(net) ; NetworkName(net) <- value
- NetworkTitle(net); NetworkTitle(net) <- value
- NetworkComment(net); NetworkComment(net) <- value
- NetworkUserField(net,fieldname); NetworkUserField(net,fieldname) <- value
- Network AllUserFields(net)

Node Management

- NewDiscreteNode(net,names,states)
- NewContinuousNode(net,names)
- DeleteNodes(nodes)
- CopyNodes(nodes,newnames,newnet)
- is.NeticaNode(obj)
- is.discrete(node); is.continuous(node)
- NodeNet(node)
- NetworkAllNodes(net)
- NetworkFindNodes(net,names)

Node Metadata

- NodeName(node); NodeName(node) <- value
- NodeTitle(node); NodeTitle(node) <- value
- NodeDescription(node); NodeDescription(node) <-value
- NodeUserField(node,fieldname); NodeUserField(node,fieldname)<-value
- NodeAllUserFields(node)
- NodeKind(node); NodeKind(node)<-value
- NodeVisStyle(node); NodeVisStyle(node) <-value
- NodeVisPos(node); NodeVisPos(node)<-value

Node States

- NodeNumStates(node)
- NodeStates(node); NodeStates(node) <- values
- NodeStateTitles(node); NodeStateTitles(node) <- values
- NodeStateComments(node); NodeStateComments(node) <-values
- NodeLevels(node); NodeLevels(node) <- values

Node Sets

- NodeSets(node); NodeSets(node) <- values
- NetworkNodesInSet(net,setname)
- NetworkSetPriority(net,setlist)
- NetworkNodeSetColor(net,setname,newcolor)

Edges & Graph Structure

- AddLink(parent,child);
  DeleteLink(parent,child);
  ReverseLink(node1,node2)
- NodeParents(child)
- NodeParents(child) <- parents ## adds links and reorder parents
- NodeChildren(parent)
- GetRelatedNodes(node);
  is.NodeRelated(node1, node2)

Probability Tables

- NodeProbs(node); NodeProbs(node) <- value
- Extract.NeticaNode; node[selection]; node[selection] <- value; node[[selection]]
- ParentStates(node); ParentNames(node)
- normalize(obj) ## Generic, supports CPA and CPF
- IsNodeDeterministic(node)
- HasNodeTable(node)
- DeleteTable(node)
- as.CPA(cpf); as.CPF(cpa)

Graph Fragments

- AbsorbNodes(nodelist)
- NodeInputNames(node); NodeInputNames(node) <- values
- ParentNames(node) – returns input names
- AdjoinNetork(base,fragment)
- NetworkFootprint(fragment)
- MakeCliqueNode(nodelist); is.CliqueNode(node); GetClique(cliquenode)

Compiling the Network

- CompileNetwork(net); UncompileNetwork(net); is.NetworkCompiled(net)
- CompiledSize(net); JunctionTreeReport(net)
- EliminationOrder(net); EliminationOrder(net) <- nodelist
- GetNetworkAutoUpdate (net); SetNetworkAutoUpdate(net,value); WithoutAutoUpdate(net,expression)
- IsBeliefUpdated(node)

Entering Findings

- NodeFinding(node); NodeFinding(node) <- value
- EnterNegativeFinding(node,state)
- RetractNetFindings(net)
- RetractNodeFinding(node)
- EnterFindings(net, findings)
- NodeLikelihood(node); NodeLikelihood(node) <- value
- NodeValue(node); NodeValue(node) <- value
- EnterGaussianFinding(node,mean,sem); EnterIntervalFinding(node,low,high) [Currently not working]

Network Queries

- NodeBeliefs(node)
- FindingsProbability(net)
- JointProbability(nodelist)
- MostProbableConfig(net,nth=0)
- NodeExpectedValue(node)
- MutalInfo(node,nodelist); VarianceOfReal(node,nodelist)
- CalcNodeState(node); CalcNodeValue(node) [Currently behaving unexpectedly]

Node Equations

- Probably easier to write calculations in R using Extract.NeticaNode than use Netica equations, mostly for compatibility
- NodeEquation(node);
  NodeEquation(node) <- value
- EquationToTable(node)

Case Streams

- CaseFileDelimiter(newdelimiter); CaseFileMissingCode(newcode)
- CaseFileStream(pathname); is.CaseFileStream(obj)
- OpenCaseStream(path); CloseCaseStream(stream); WithOpenCaseStream(stream,expr)
- isCaseStreamOpen(stream); is.NeticaCaseStream(obj)
- getCaseStreamPath(stream); getCaseStreamLastId(stream); getCaseStreamPos(stream); getCaseStreamLastFreq(stream)
- read.CaseFile(file,…); write.CaseFile(x, file, …)
- ReadFindings(nodes,stream); WriteFindings(nodes, pathOrStream)

Memory Streams

- [Currently having difficulties with these, better to write to file and use file streams]
- MemoryCaseStream(data.frame, label)
- is.MemoryCaseStream(obj)
- getCaseStreamDataFrameName(stream)
- MemoryStreamContents(stream);
  MemoryStreamContents(stream) <- value

Network Learning

- NodeExperience(node);
  NodeExperience(node) <- value
- FadeCPT(node)
- LearnFindings(nodes,weight=1)
- LearnCases(caseStream,nodelist,weight=1)
- LearnCPTs(caseStream,nodelist,…)
  - [Warning: this function does not provide convergence information for EM method]