Bayesian networks in R with RUnBBayes package

June 11, 2014

Developers: Fernando Santos
Yuri Lavinas
Samuel Pala
Diego Marques
Pedro Henrique
Instructor: Rommel Carvalho

1 Introduction

The RUnBBayes package provides access to some functionalities of the UnBBayes framework. UnBBayes (`http://unbbayes.sourceforge.net/`) is an open source software for modeling, learning and reasoning upon probabilistic networks developed in Java. Making use of rJava, this package provides an interface to implement probabilistic networks within R.

2 The chest clinic example

This section explains how to use RUnBBayes in the chest clinic example of Lauritzen and Spiegelhalter (1988) (Figure 1). As stated by Lauritzen and Spiegelhalter (1988):

> Shortness-of-breath (dyspnoea) may be due to tuberculosis, lung cancer or bronchitis, or none of them, or more than one of them. A recent visit to Asia increases the chances of tuberculosis, while smoking is known to be a risk factor for both lung cancer and bronchitis. The results of a single chest X-ray do not discriminate between lung cancer and tuberculosis, as neither does the presence or absence of dyspnoea.

2.1 Defining the network nodes

We can create a network by defining its nodes, together with their conditional probabilities and their state.

```
> library(runbbayes)
> node.a = createNodeInfo(~asia, prob=c(0.01, 0.99),
+                         states=c("yes","no"))
> node.t = createNodeInfo(~tub|asia, prob=c(0.05, 0.95, 0.01, 0.99),
+                         states=c("yes","no"))
> node.s = createNodeInfo(~smoke, prob=c(0.5,0.5),
+                         states=c("yes","no"))
> node.l = createNodeInfo(~lung|smoke, prob=c(0.1, 0.9, 0.01, 0.99),
+                         states=c("yes","no"))
> node.b = createNodeInfo(~bronc|smoke, prob=c(0.6, 0.4, 0.3, 0.7),
+                         states=c("yes","no"))
> node.e = createNodeInfo(~either|lung:tub,prob=c(1,0,1,0,1,0,0,1),
```

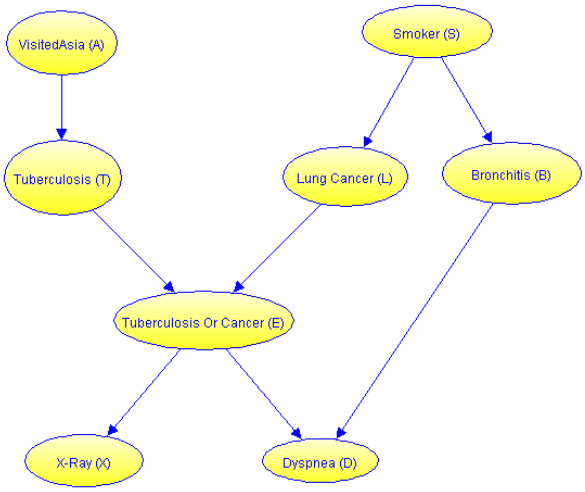

Figure 1: Chest clinic example

```
+                         states=c("yes","no"))
> node.x = createNodeInfo(~xray|either, prob=c(0.98, 0.02, 0.05, 0.95),
+                         states=c("yes","no"))
> node.d = createNodeInfo(~dysp|bronc:either, prob=c(0.9, 0.1, 0.7, 0.3, 0.8, 0.2, 0.1, 0.9),
+                         states=c("yes", "no"))
```

Each of these calls will return a "nodeinfo" structure.

```
> node.a = createNodeInfo(~asia, prob=c(0.01, 0.99),
+                         states=c("yes","no"))
> node.a
```

```
Node:           asia
Parents:        NA
Probabilities:  0.01, 0.99
States:         yes, no
```

2.2 Compiling the network

Create a probabilistic network, from a node list. Compile list of conditional probability tables and create the network.

```
> nodeList = list(node.a, node.t, node.s, node.l, node.b, node.e, node.x, node.d)
> network = createNetwork(nodeList, compile=TRUE)
> network
```

```
Compiled: TRUE
P ( asia )
P ( tub | asia )
P ( smoke )
P ( lung | smoke )
P ( bronc | smoke )
P ( either | lung tub )
P ( xray | either )
P ( dysp | bronc either )
```

Setting compile as true, gives you a compiled network. Otherwise, the network won't be compiled by default.So, it's necessary to use the compileNetwork function as an option to build the junction tree.

```
> netCompiled = compileNetwork(network)
> netCompiled

Compiled: TRUE
P ( asia )
P ( tub | asia )
P ( smoke )
P ( lung | smoke )
P ( bronc | smoke )
P ( either | lung tub )
P ( xray | either )
P ( dysp | bronc either )
```

2.3 Querying the network

1. The network can be queried to return the priori probabilities of all nodes:

```
> prioriProb = queryNetwork(network)
> prioriProb

$asia
                yes                no
                0.01                0.99

$tub
                yes                no
                0.0104                 0.9896

$smoke
                yes                no
                0.5                0.5

$lung
                yes                no
                0.055                0.945

$bronc
                yes                no
                0.45                0.55

$either
                yes                no
                0.0648                 0.9352

$xray
                yes                no
                0.1103                 0.8897

$dysp
                yes                no
                0.436                0.564
```

```
> prioriProb$xray

$yes

                0.1103

$no

                0.8897

> prioriProb$xray$yes

[1] 0.1103
```

2. The network can be queried to return the priori probabilities of some specific nodes:

```
> prioriProb = queryNetwork(network, c("bronc", "dysp"))
> prioriProb

$bronc
                yes                no
                0.45                0.55

$dysp
                yes                no
                0.436                0.564
```

3. The network can return the posteriori probabilities of some event given some evidences without modifying the current network object:

```
> posterioriProb = queryNetwork(network, c("either"), list(c("asia",
+ "yes"), c("smoke","no")))
> posterioriProb

$either
                yes                no
                0.0595                0.9405
```

4. Evidences can be set and reset in the network:

```
> network = setEvidence(network, list(c("asia", "yes"), c("smoke", "no")))
> network = propagateEvidence(network)
> posterioriProb = queryNetwork(network, c("dysp", "yes"))
> posterioriProb

$dysp
                yes                no
                0.3368                0.6632

> network = resetEvidence(network)
> prioriProb = queryNetwork(network, c("dysp", "yes"))
> prioriProb

$dysp
                yes                no
                0.436                0.564
```

2.4 Updating the network

1. Nodes can be added or removed from a compiled network:

```
> network = addNode(network, ~asthma|smoke, prob = c(0.6, 0.4, 0.85, 0.15), states = c("yes",
+ "no"))
> network = removeNode(network, "asthma")
```