Bayesian Networks Second delivery_

Samar Moussa/NIU:1661270 _ Jamia Begum/NIU: 1676891

2022-12-09

This is Exercise 5 of the list of exercise set for Topic2 in Bayesian Networks

Q1- (a) Using gRain, develop the corresponding Bayesian Network and use it to compute P (M = T /H = F) and P (M = F/H = T).

```
# First: we will be considering the binary nodes
tf <- c("false", "true")
#then , we will Specify the CPTs (conditional probability table):
node.M <- cptable(~ M, values = c(80, 20), levels = tf)
node.S <- cptable(~ S + M, values = c(80, 20, 20, 80), levels = tf)
node.B <- cptable(~ B + M, values = c(95, 5, 80, 20), levels = tf)
node.C <-
  cptable(~ C + S + B,
          values = c(95, 5, 20, 80, 20, 80, 20, 80),
          levels = tf)
node.H <- cptable(~ H + B, values = c(40, 60, 20, 80), levels = tf)

# Second step, we will need to create an intermediate representation of the CPTs:
plist <- compileCPT(list(node.M, node.S, node.B, node.C, node.H))
plist
```

```
##  P( M )
##  P( S | M )
##  P( B | M )
##  P( C | S B )
##  P( H | B )
```

```
plist$M
```

```
## M
## false  true
##   0.8   0.2
```

```
plist$S
```

```
##        M
## S       false true
##   false   0.8  0.2
##   true    0.2  0.8
```

```
plist$B
```

```
##        M
## B       false true
##   false  0.95  0.8
##   true   0.05  0.2
```

```
plist$C
```

```
## , , B = false
##
##        S
## C       false true
##   false  0.95  0.2
##   true   0.05  0.8
##
## , , B = true
##
##        S
## C       false true
##   false   0.2  0.2
##   true    0.8  0.8
```

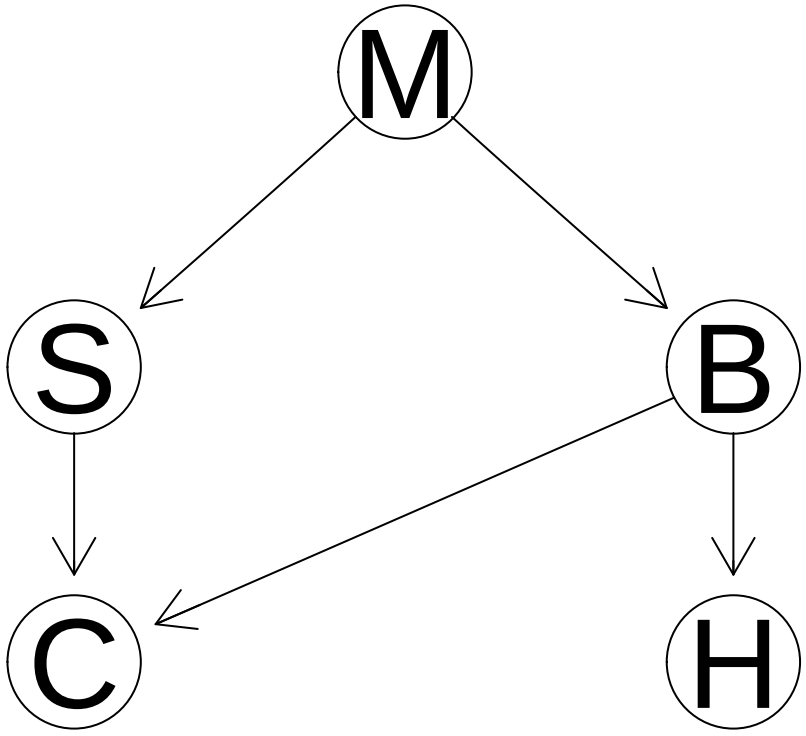
```
plist$H
```

```
##        B
## H       false true
##   false   0.4  0.2
##   true    0.6  0.8
```

```
# Third step is to use the gRain library to develop the Bayesian Network
#and plot it which answers the first question (a) from the 5th exercise
BN.net <- grain(plist)
#Providing a summary of the established Bayesian Network
summary(BN.net)
```

```
## Independence network: Compiled: TRUE Propagated: FALSE
##  Nodes : chr [1:5] "M" "S" "B" "C" "H"
##  Number of cliques:                3
##  Maximal clique size:              3
##  Maximal state space in cliques:   8
```

```
#Plotting the Bayesian Network, we used dag to remove the unwanted edge between S and B
BN_graph = plot(BN.net$dag)
```




```
# Calculating The probability distribution of M given H P(M/H)
M_given_H <- querygrain(BN.net, nodes = c("M", "H"), type = "conditional")
"P(M/H):"
```

```
## [1] "P(M/H):"
```

```
#This variable will show us the probability of M given H taking into consideration all possible values
M_given_H
```

```
##        H
## M        false      true
##   false 0.8125 0.7922078
##   true  0.1875 0.2077922
```

Q2-(b) Develop the LS Algorithm to find these two probabilities with R and compare the results with previous item.

```
# we will use the logic sampling algorithm as a function with 5 parameters,
#where the first parameter is the query node, the second parameter is the evidence node, # the third pa
# and the last fifth parameter is the maximum number of iterations

Nodes <- names(plist)
```

```r
Values <- c()
Logic_Sampling <- function(query_node,
               evidence_node,
               state_of_query,
               state_of_evidence,
               iter_max) {

  # Initialization step for the LS algorithm:
  # • Creating a count variable Count(xi,e) and a count variable Count(e).
  # • Initializing both count variables to 0, we will call them i and j here

  j = 0 # This is for variable Count(xi,e)
  i = 0 # This is for the variable count (e)


  for (k in 1:iter_max) {


    # 1. For all root nodes:
    # Randomly choose a value for it, weighting the choice by the
    # priors.

    # 2. Loop:
    # Choose values randomly for children, using the conditional
    # probabilities given the known values of the parents.
    # Nodes <- names(plist)
    # Values <- c()

    for (node in Nodes) {
      if (node == "M") {
        Values[1] <- sample(tf, 1, prob = c(80, 20))
      }

      else if (node == "S") {
        if (Values[1] == "false") {
          Values[2] <- sample(tf, 1, prob = c(80, 20))
        }
        else
          Values[2] <- sample(tf, 1, prob = c(20, 80))
      }

      else if (node == "B") {
        if (Values[1] == "false") {
          Values[3] <- sample(tf, 1, prob = c(95, 5))
        }
        else
          Values[3] <- sample(tf, 1, prob = c(80, 20))
      }

      else if (node == "C") {
        if (Values[2] == "false") {
          if (Values[3] == "false") {
            Values[4] <- sample(tf, 1, prob = c(95, 5))
```

```
          }
          else
            Values[4] <- sample(tf, 1, prob = c(20, 80))
        }

        else {
          if (Values[3] == "false") {
            Values[4] <- sample(tf, 1, prob = c(20, 80))
          }
          else
            Values[4] <- sample(tf, 1, prob = c(20, 80))
        }
      }

      else{
        if (Values[3] == "false") {
          Values[5] <- sample(tf, 1, prob = c(40, 60))
        }
        else{
          Values[5] <- sample(tf, 1, prob = c(20, 80))
        }
      }

    }

    # 3. Updating the run counts:
    # If the case includes E=e, Count (e)
    # Count(e)+1
    # If the case includes X=xi and E=e, Count(xi,e)
    # Count(xi,e)+1

    query = which(Nodes == query_node)
    evidence = which(Nodes == evidence_node)

    if (Values[evidence] == state_of_evidence){
      i = i + 1
    }
    if ((Values[query] == state_of_query) &
        (Values[evidence] == state_of_evidence)){
      j = j + 1
    }


  }
  return(j/i)
}
```

calculating Logic sampling approximate probability for different values of M and H

```
# We are calculating the approximate probability
# P(M =T | H=F)
Logic_Sampling("M","H","true","false",3000)
```

```
## [1] 0.176521
```

```r
# We are calculating the approximate probability
# P(M = F| H = T)

Logic_Sampling("M","H","false","true",3000)
```

```
## [1] 0.7827253
```

Q3-(c) Develop the LW Algorithm to find these two probabilities with R and compare the results with previous items.

```r
Likelihood_Weighting <- function(query_node,
                evidence_node,
                state_of_query,
                state_of_evidence,
                iter_max) {

  # Initialization of the LW algorithm
  # • Creating  a count variable Count(xi,e) and a count variable Count(e).
  # • Initializing both count variables to 0

  j = 0
  i = 0


  for (k in 1:iter_max) {


    # 1. For all root nodes:
    # If a root is the evidence node E,
    # Choose E=e, and assign likelihood(E=e)
    # P(E=e)
    # Else
    # Randomly choose a value for it, weighting the choice by the
    # priors.
    # 2. Loop:
    # If a child is the evidence node E,
    # Choose E=e, and likelihood(E=e)
    # P(E=e / chosen parents
    # values)
    # Else
    # Choose values randomly for children, using the conditional
    # probabilities given the known values of the parents.
    # Until all the leaves are reached.

    for (node in Nodes) {
      if (node == "M") { #for root node
        if (node == evidence_node) {
          Values[1] <- state_of_evidence
          if (Values[1] == "false") {
            likelihood <- 0.8
          }
```

```
      else
        likelihood <- 0.2
    }
    else{
      Values[1] <- sample(tf, 1, prob = c(80, 20)) # if the node is not an evidence
      #we are randomly choosing a value
    }
  }

  # for the child node
  else if (node == "S") {
    if (Values[1] == "false") { # when the parent M is false
      if (node == evidence_node) {
        Values[2] <- state_of_evidence
        if (Values[2] == "false") { # P(S|M=F)
          likelihood <- 0.8
        }
        else
          likelihood <- 0.2
      }
      else
        Values[2] <- sample(tf, 1, prob = c(80, 20))
    }
    else{
      if (node == evidence_node) {
        Values[2] <- state_of_evidence
        if (Values[2] == "false") { # P(S|M=T)
          likelihood <- 0.2
        }
        else
          likelihood <- 0.8
      }
      else
        Values[2] <- sample(tf, 1, prob = c(20, 80))
    }
  }

  else if (node == "B") {
    if (Values[1] == "false") {
      if (node == evidence_node) {
        Values[3] <- state_of_evidence
        if (Values[3] == "false") { # assigning likelihood as P(B|M=F)
          likelihood <- 0.95
        }
        else
          likelihood <- 0.05
      }
      else
        Values[3] <- sample(tf, 1, prob = c(95, 5)) # Otherwise, randomly choosing
      # the event
    }
    else{
      if (node == evidence_node) {   # assigning likelihood as P(B|M=T)
```

```
      Values[3] <- state_of_evidence
      if (Values[3] == "false") {
        likelihood <- 0.8
      }
      else
        likelihood <- 0.2
    }
    else
      Values[3] <- sample(tf, 1, prob = c(80, 20))
  }
}

else if (node == "C") {
  if (Values[2] == "false") {
    if (Values[3] == "false") {
      if (node == evidence_node) {
        Values[4] <- state_of_evidence
        if (Values[4] == "false") {
          likelihood <- 0.95
        }
        else
          likelihood <- 0.05
      }
      else
        Values[4] <- sample(tf, 1, prob = c(95, 5))
    }
    else{
      if (node == evidence_node) {
        Values[4] <- state_of_evidence
        if (Values[4] == "false") {
          likelihood <- 0.2
        }
        else
          likelihood <- 0.8
      }
      else
        Values[4] <- sample(tf, 1, prob = c(20, 80))
    }
  }

  else {
    if (Values[3] == "false") {
      if (node == evidence_node) {
        Values[4] <- state_of_evidence
        if (Values[4] == "false") {
          likelihood <- 0.2
        }
        else
          likelihood <- 0.8
      }
      else
        Values[4] <- sample(tf, 1, prob = c(20, 80))
    }
```

```
        else{
          if (node == evidence_node) {
            Values[4] <- state_of_evidence
            if (Values[4] == "false") {
              likelihood <- 0.2
            }
            else
              likelihood <- 0.8
          }
          else
            Values[4] <- sample(tf, 1, prob = c(20, 80))
        }
      }
    }

    else{
      if (Values[3] == "false") {
        if (node == evidence_node) {
          Values[5] <- state_of_evidence
          if (Values[5] == "false") {
            likelihood <- 0.4
          }
          else
            likelihood <- 0.6
        }
        else
          Values[5] <- sample(tf, 1, prob = c(40, 60))
      }
      else{

        if (node == evidence_node) {
          Values[5] <- state_of_evidence
          if (Values[5] == "false") {
            likelihood <- 0.2
          }
          else
            likelihood <- 0.8
        }
        else
          Values[5] <- sample(tf, 1, prob = c(20, 80))
      }
    }
  }



  # Update the counter for evidence (E=e)
  i = i + likelihood

  # Control for the occurrence of X=xi
  # and add the likelihood if X=xi
  query = which(Nodes == query_node)
```

```
    if (Values[query] == state_of_query) #if we find the query and evidence together
      #only then we need to count j
      j = j + likelihood


  }
  return(j/i)
}
```

Calculating the Likelihood weighting approximation probability:

```
# we are getting the approximate probability P(M=T|H=F)
Likelihood_Weighting("M","H","true","false",3000)
```

```
## [1] 0.1848637
```

```
# we are getting the approximate probability P(M=F|H=T)

Likelihood_Weighting("M","H","false","true",3000)
```

```
## [1] 0.7914769
```

Q4- (d) Compute exactly "by hand" P (M = T /H = F), and compare with (a), (b) and (c)

$$P(M/H) = \frac{P(H/M) \times P(M)}{P(H)}$$

Applying the law of total probability to conditional probability we get:

$$P(H/M) = P(H/M, B) \times P(B/M) + P(H/M, B^c) \times P(B^c/M)$$

Markov condition implies that:

$$P(H/M, B) = P(H/B)$$

And:

$$P(H/M, B^c) = P(H/B^c)$$

Law of total probability:

$$P(H) = P(H/B) \times P(B) + P(H/B^c) \times P(B^c)$$
$$P(B) = P(B/M) \times P(M) + P(B/M^c) \times P(M^c)$$

Summarizing and applying for M = T and H = F we get:

$$P(M = T/H = F) = \frac{(0.2 \times 0.2 + 0.4 \times 0.8) \times 0.2}{0.2 \times 0.08 + 0.4 \times 0.92} = 0.1875$$

looking at this result, we can add that the gRain in R gives more precise calculation, the Logic sampling and Likelihood weighting can give almost the same results(approximation) as we can see below

```
Logic_Sampling("M","H","true","false",100)
```

```
## [1] 0.1842105
```

```
Likelihood_Weighting("M","H","true","false",100)
```

```
## [1] 0.1785714
```

```
Logic_Sampling("M","H","true","false",1000)
```

```
## [1] 0.2126582
```

```
Likelihood_Weighting("M","H","true","false",1000)
```

```
## [1] 0.2041885
```

```
Logic_Sampling("M","H","true","false",3000)
```

```
## [1] 0.1897302
```

```
Likelihood_Weighting("M","H","true","false",3000)
```

```
## [1] 0.183936
```

```
Logic_Sampling("M","H","true","false",10000)
```

```
## [1] 0.1789692
```

```
Likelihood_Weighting("M","H","true","false",10000)
```

```
## [1] 0.1826313
```

Q5-(e) For the evidence "H = F" and the query variable M , compute the Kullback-Leibler divergence for the LS Algorithm, and also compute it for the LW Algorithm, and compare them. Which algorithm seems to be better?

In this question, we will calculate the Kullback-Leibler divergence for the two algorithms for evidence H = F and query M=T

```
KL_LS <- c()
KL_LW <- c()
for (iter in seq(100, 10000, by=100)){
  KL_LS <- c( KL_LS,
    0.1875 * log(0.1875 / Logic_Sampling("M", "H", "true", "false", iter)) + (1 - 0.1875) *
    log((1 - 0.1875) / Logic_Sampling("M", "H", "true", "true", iter)))
  KL_LW <- c( KL_LW,
    0.1875 * log(0.1875 / Likelihood_Weighting("M", "H", "true", "false", iter)) + (1 - 0.1875) *
    log((1 - 0.1875) / Likelihood_Weighting("M", "H", "true", "true", iter)))
}

df <- data.frame("Index"=seq(1, 10000, by=100), "KL_LS"=KL_LS, "KL_LW"=KL_LW)
```

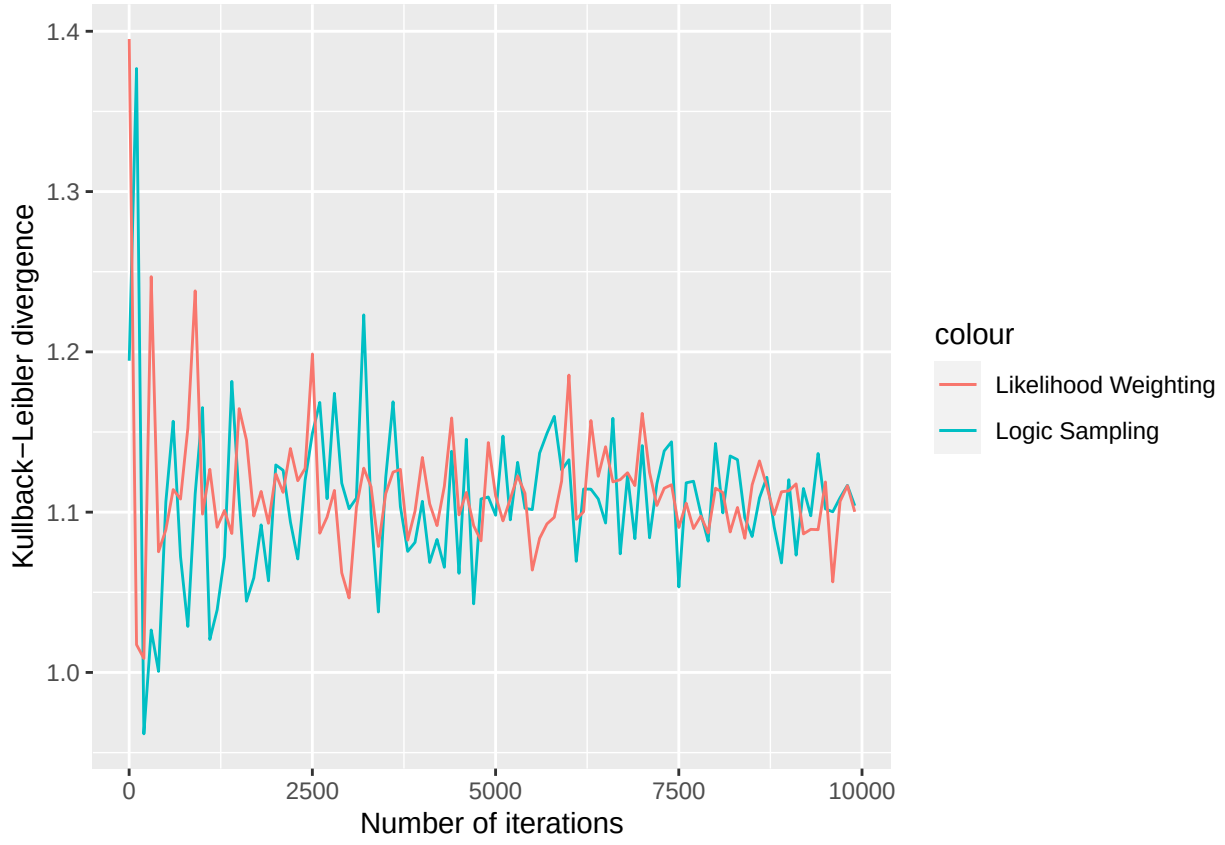
```
ggplot(df, aes(x=df$Index)) +
  geom_line(aes(y = df$KL_LS, color = "Logic Sampling")) +
  geom_line(aes(y = df$KL_LW, color="Likelihood Weighting")) +
  xlab("Number of iterations") +
  ylab("Kullback-Leibler divergence")
```

```
## Warning: Use of ‘df$KL_LS‘ is discouraged.
## i Use ‘KL_LS‘ instead.
```

```
## Warning: Use of ‘df$Index‘ is discouraged.
## i Use ‘Index‘ instead.
```

```
## Warning: Use of ‘df$KL_LW‘ is discouraged.
## i Use ‘KL_LW‘ instead.
```

```
## Warning: Use of ‘df$Index‘ is discouraged.
## i Use ‘Index‘ instead.
```



Looking at the plot we have here, in comparison between likelihood weighting approximation results and the Logic sampling approximation results we can see that the two algorithm's performances are similar in this case.