AML Lab 1 (Graphical models)

Roshni Sundaramurthy

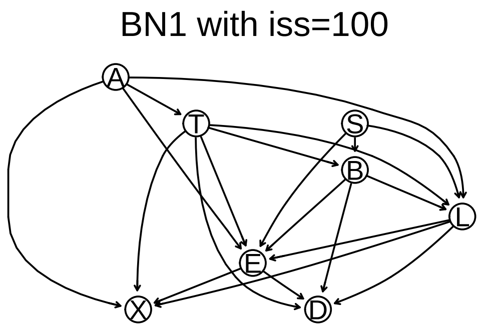
20/09/2019

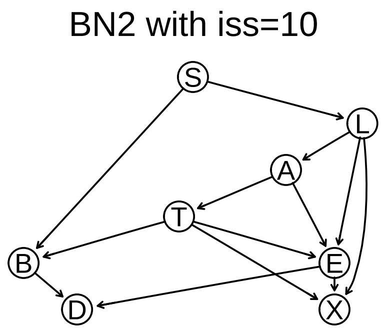
Question 1

Show that multiple runs of the hill-climbing algorithm can return non-equivalent Bayesian network (BN) structures. Explain why this happens. Use the Asia dataset which is included in the bnlearn package. To load the data, run data("asia").

Hint: Check the function hc in the bnlearn package. Note that you can specify the initial structure, the number of random restarts, the score, and the equivalent sample size (a.k.a imaginary sample size) in the BDeu score. You may want to use these options to answer the question. You may also want to use the functions plot, arcs, vstructs, cpdag and all.equal.

```
library(bnlearn)

# load data
data("asia")

# ... choose an algorithm and learn the structure of the network from the data...

hc1 = hc(asia, score = "bde", restart = 5, iss=100)
hc2 = hc(asia, score = "bde", restart = 5, iss=10)

# iss set to a very small value to reduce the relative weight of the prior distribution

# ... plot it...

#source("http://bioconductor.org/biocLite.R")
#biocLite(c("graph", "RBGL", "Rgraphviz"))
#install.packages("gRain", dependencies=TRUE)

library(gRain)
library(Rgraphviz)

par(mfrow=c(1,2))
graphviz.plot(hc1, main="BN1 with iss=100")
graphviz.plot(hc2, main="BN2 with iss=10")
```





```r
#library(bnviewer)
#viewer(hc1,bayesianNetwork.title="Bayesian Network 1 - Asia")
#viewer(hc2,bayesianNetwork.title="Bayesian Network 2 - Asia")


################### COMPARISON ###########################
# ARCS
all.equal(arcs(hc1),arcs(hc2))
```

```
## [1] "Attributes: < Component \"dim\": Mean relative difference: 0.35 >"
## [2] "Lengths (40, 26) differ (string compare on first 26)"
## [3] "24 string mismatches"
```

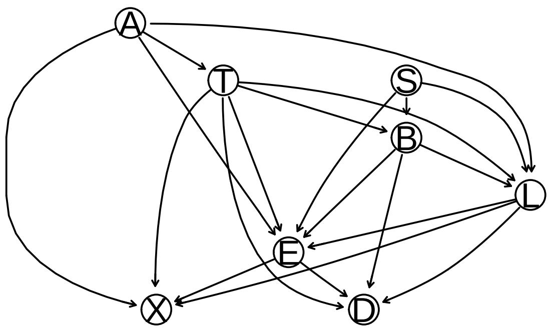
```r
# Score - to verify the goodness of fit of the learned network with respect to
# a particular score function
#score(hc1,asia, type = "bde")

# VSTRUCTS
vstructs(hc1)
```

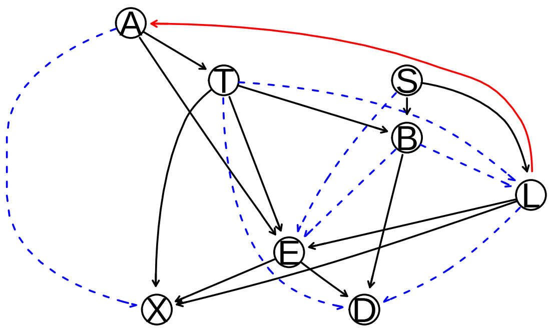
```
##        X   Z   Y
##  [1,] "A" "L" "S"
##  [2,] "A" "L" "T"
##  [3,] "A" "L" "B"
##  [4,] "S" "L" "T"
##  [5,] "S" "L" "B"
##  [6,] "T" "L" "B"
##  [7,] "S" "B" "T"
##  [8,] "A" "E" "S"
##  [9,] "A" "E" "T"
## [10,] "A" "E" "B"
## [11,] "S" "E" "T"
## [12,] "S" "E" "B"
## [13,] "T" "E" "B"
```

```r
vstructs(hc2)
```

```
##      X   Z   Y
## [1,] "S" "B" "T"
## [2,] "T" "E" "L"
## [3,] "T" "X" "L"
## [4,] "B" "D" "E"
```

```r
# GRAPHICAL COMPARISON
graphviz.compare(hc1,hc2)
```





```r
# SKELETONS
all.equal(hc1,hc2)
```

```
## [1] "Different number of directed/undirected arcs"
```

```r
all.equal(cpdag(hc1),cpdag(hc2))
```

```
## [1] "Different number of directed/undirected arcs"
```

```r
all.equal(skeleton(hc1),skeleton(hc2))
```

```
## [1] "Different number of directed/undirected arcs"
```

```r
# counts the number of arcs that are different between two network
paste("Hamming distance between hc1 and hc2:",hamming(hc1,hc2))
```

```
## [1] "Hamming distance between hc1 and hc2: 7"
```

The two graphs are equivalent if the respective BNs define the same probability distribution and the structures of those networks are equivalent. The above results are drawn for different parameter settings (say restart, iss) while learning the structure of the network from the asia data. The "iss" is set to vary the relative weight of the prior distribution. When it is large (say 100), the network will become denser. The "restart" denotes number of random restarts in the graph.

When compared two networks,

- We consider 1st BN true and in other network, true positive arcs are in black.
- False positive arcs (which are missing or have different directions in the true network) are in red.
- False negative arcs are in blue, and drawn using a dashed line.

There exists string mismatches and it seems 7 arcs (hamming) differ between 2 BNs and they have different skeletons. So, we can say that they are not equivalent.

Question 2

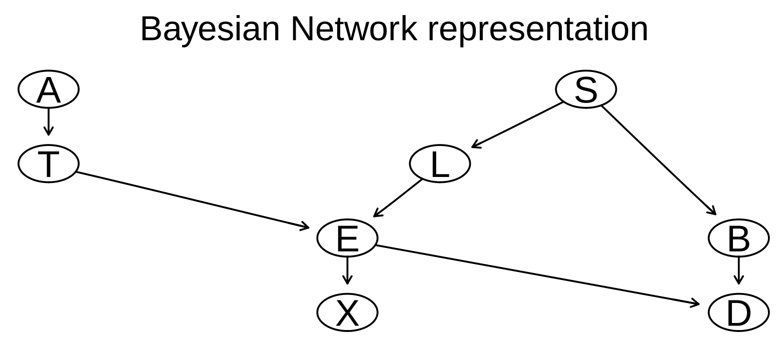
Learn a BN from 80 % of the Asia dataset. The dataset is included in the bnlearn package. To load the data, run data("asia"). Learn both the structure and the parameters. Use any learning algorithm and settings that you consider appropriate. Use the BN learned to classify the remaining 20 % of the Asia dataset in two classes: S = yes and S = no. In other words, compute the posterior probability distribution of S for each case and classify it in the most likely class. To do so, you have to use exact or approximate inference with the help of the bnlearn and gRain packages, i.e. you are not allowed to use functions such as predict. Report the confusion matrix, i.e. true/false positives/negatives. Compare your results with those of the true Asia BN, which can be obtained by running *dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")*.

Hint: You already know the Lauritzen-Spiegelhalter algorithm for inference in BNs, which is an exact algorithm. There are also approximate algorithms for when the exact ones are too demanding computationally.

For exact inference, you may need the functions bn.fit and as.grain from the bnlearn package, and the functions compile, setFinding and querygrain from the package gRain. For approximate inference, you may need the functions prop.table, table and cpdist from the bnlearn package. When you try to load the package gRain, you will get an error as the package RBGL cannot be found. You have to install this package by running the following two commands (answer no to any offer to update packages): source("https://bioconductor.org/biocLite.R") biocLite("RBGL")

HC Algorithm

```
# split data

n=dim(asia)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.8))
train=asia[id,]
test=asia[-id,]

# learn structure of asia

hc1 = hc(train, score="bde")
graphviz.plot(hc1,shape = "ellipse", main = "Bayesian Network representation")
```



```
train_fit <- bn.fit(hc1, train)

# gRain steps when performing inference:
# • compile: moralize, triangulate, find RIP ordering, form initial clique potentials
# • optional: absorb evidence
# • propagate: transform clique potentials to clique marginals

# querygrain performs those steps internally, no need to do them oneself


# network
hc2 <- as.grain(train_fit)
summary(hc2)
```

```
## Independence network: Compiled: FALSE Propagated: FALSE
##  Nodes : chr [1:8] "A" "S" "T" "L" "B" "E" "X" "D"
```

```
# The compile() method performs the following steps:
# (i) Creates the moral graph.
# (ii) Detects that the moral graph is not triangulated and therefore creates a triangulated graph
# by making fill-ins.
# (iii) Establishes a potential representation by absorbing each CPT into an appropriate
```

```r
# clique potential; i.e., establish the representation in (8.
# (iv) Creates a junction tree of the cliques.

library(gRbase)
hc2 <- compile(hc2)

tabel<-matrix(c(0,0,0,0),nrow=2,ncol=2)

for(i in 1:1000){
  z<-NULL
  for(j in c("A","T","L","B","E","X","D")){
    if(test[i,j]=="no"){
      z<-c(z,"no")
    }
    else{
      z<-c(z,"yes")
    }
  }
  hc3<-setFinding(hc2,nodes=c("A","T","L","B","E","X","D"),states=z) # Enter the evidence
  x<-querygrain(hc3,c("S")) # network with the findings queried
  y <- ifelse(x$S[1]>x$S[2],1,2)
  if(test[i,2]=="yes"){tabel[y,2]<-tabel[y,2]+1}
  else{tabel[y,1]<-tabel[y,1]+1}
}
accuracy_2 = (sum(diag(tabel))/ 1000) * 100
```

```
## Confusion matrix for Hill climbing :

##      [,1] [,2]
## [1,]  322  120
## [2,]  146  412

## Accuracy for Hill climbing :

## [1] 73.4
```

Comparing with True Asia BN:

```r
# Comparing

dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
dag
```

```
##
##   Random/Generated Bayesian network
##
##   model:
##    [A][S][B|S][L|S][T|A][E|L:T][D|B:E][X|E]
##   nodes:                                 8
##   arcs:                                  8
##     undirected arcs:                     0
##     directed arcs:                       8
##   average markov blanket size:           2.50
##   average neighbourhood size:            2.00
##   average branching factor:              1.00
##
```

```
##   generation algorithm:                    Empty
```

```
train_fit <- bn.fit(dag, train)

# network
hc2 <- as.grain(train_fit)

hc2 <- compile(hc2)

tabel<-matrix(c(0,0,0,0),nrow=2,ncol=2)

for(i in 1:1000){
  z<-NULL
  for(j in c("A","T","L","B","E","X","D")){
    if(test[i,j]=="no"){
      z<-c(z,"no")
    }
    else{
      z<-c(z,"yes")
    }
  }
  hc3<-setFinding(hc2,nodes=c("A","T","L","B","E","X","D"),states=z) # Enter the evidence
  x<-querygrain(hc3,c("S")) # network with the findings queried
  y <- ifelse(x$S[1]>x$S[2],1,2)
  if(test[i,2]=="yes"){tabel[y,2]<-tabel[y,2]+1}
  else{tabel[y,1]<-tabel[y,1]+1}
}
accuracy_true = (sum(diag(tabel))/ 1000) * 100
```

```
## Confusion matrix for True Asia BN :
```

```
##      [,1] [,2]
## [1,]  322  120
## [2,]  146  412
```

```
## Accuracy for True Asia BN :
```

```
## [1] 73.4
```

Bayesian networks aims to model conditional dependence, and causation, by representing conditional dependence by edges in a directed graph. In BN, the edges tells us about the conditional dependence.

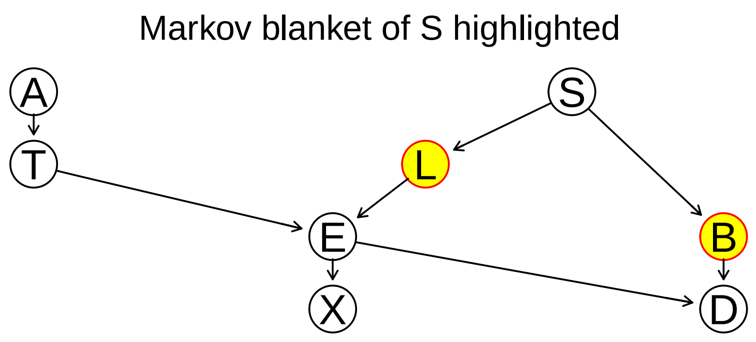
Question 3

In the previous exercise, you classified the variable S given observations for all the rest of the variables. Now, you are asked to classify S given observations only for the so-called Markov blanket of S, i.e. its parents plus its children plus the parents of its children minus S itself. Report again the confusion matrix.

Hint: You may want to use the function mb from the bnlearn package.

```
# Learning the equivalence class of a directed acyclic graph (DAG) from data using the
# Grow-Shrink Algorithm
# res = gs(train)

# learning by HC algo
```

```r
hc0 = hc(train, score="bde")

highlight.opts <- list(nodes = c("L","B"), col = "red", fill = "yellow")

graphviz.plot(hc0, highlight = highlight.opts, main="Markov blanket of S highlighted")
```



```r
train_fit <- bn.fit(hc0, train)

# Markov blanket of S
mb(train_fit, node=c("S"))
```

```
## [1] "L" "B"
```

```r
# compile
hc1 <- as.grain(train_fit)
hc2 <- compile(hc1)

tabel<-matrix(c(0,0,0,0),nrow=2,ncol=2)

for(i in 1:1000){
  z<-NULL
  for(j in c("L","B")){
    if(test[i,j]=="no"){
      z<-c(z,"no")
    }
    else{
      z<-c(z,"yes")
    }
  }
  hc3<-setFinding(hc2,nodes=c("L","B"),states=z) # Enter the evidence
  x<-querygrain(hc3,c("S")) # network with the findings queried
  y <- ifelse(x$S[1]>x$S[2],1,2)
  if(test[i,2]=="yes"){tabel[y,2]<-tabel[y,2]+1}
  else{tabel[y,1]<-tabel[y,1]+1}
}
accuracy_3 = (sum(diag(tabel))/ 1000) *100
```

```
## Confusion matrix:

##      [,1] [,2]
## [1,]  322  120
## [2,]  146  412

## Accuracy:
```

```
## [1] 73.4
```

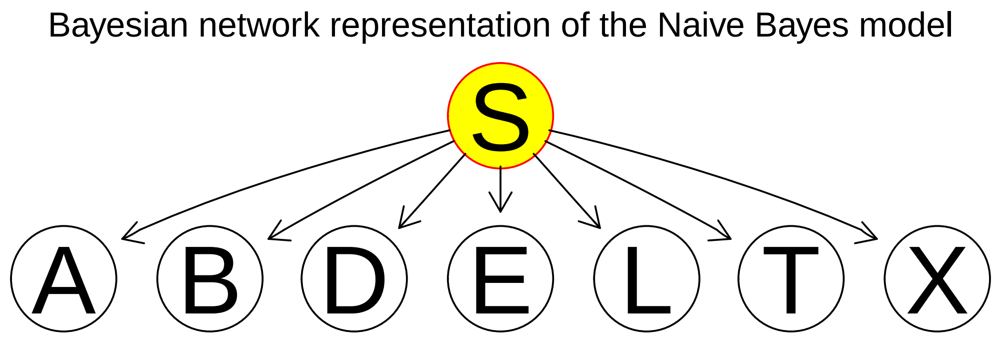
The Markov blanket for a node "S" in a Bayesian network is the set of nodes composed of S's parents, S's children, and S's children's other parents. It is needed to predict the behavior of the "S" node and its children. S is conditionally independent of the other nodes in the graph.

Question 4

Repeat the exercise (2) using a naive Bayes classifier, i.e. the predictive variables are independent given the class variable. See p. 380 in Bishop's book or Wikipedia for more information on the naive Bayes classifier. Model the naive Bayes classifier as a BN. You have to create the BN by hand, i.e. you are not allowed to use the function naive.bayes from the bnlearn package.

Hint: Check http://www.bnlearn.com/examples/dag/ to see how to create a BN by hand.

```r
# split data
n=dim(asia)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.8))
train=asia[id,]
test=asia[-id,]

# creating network by hand
dag = model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")
highlight.opts <- list(nodes = c("S"), col = "red", fill = "yellow")

graphviz.plot(dag, main = "Bayesian network representation of the Naive Bayes model",
              highlight = highlight.opts)
```



```r
hc1<-bn.fit(dag,train)
hc2<-as.grain(hc1)
hc2<-compile(hc2)
# compile = moralize, triangulate, find RIP ordering, form initial clique potentials

tabel<-matrix(c(0,0,0,0),nrow=2,ncol=2) # confusion matrix
for(i in 1:1000){
  z<-NULL
  for(j in c("A","T","L","B","E","X","D")){
    if(test[i,j]=="no"){
      z<-c(z,"no")
    }
    else{
      z<-c(z,"yes")
    }
  }
  hc3<-setFinding(hc2,nodes=c("A","T","L","B","E","X","D"),states=z) # Enter the evidence
  x<-querygrain(hc3,c("S")) # network with the findings queried
  y <- ifelse(x$S[1]>x$S[2],1,2)
  if(test[i,2]=="yes"){tabel[y,2]<-tabel[y,2]+1}
  else{tabel[y,1]<-tabel[y,1]+1}
}
accuracy_4 = (sum(diag(tabel))/ 1000) *100
```

```
## Confusion matrix for Naive bayes classifier :
```

```
##       [,1] [,2]
## [1,]   349  188
## [2,]   119  344
```

```
## Accuracy for Naive bayes classifier :
```

```
## [1] 69.3
```

Accuracy using naive.bayes function for comparing above results:

```
# to compare the above
# naive bayes parameter learning

bn = naive.bayes(train, "S")
fitted = bn.fit(bn, train)
pred = predict(fitted, test)
confus_matrix = table(pred, test[, "S"])
accuracy_naive = ((confus_matrix[1,1] + confus_matrix[2,2]) / 1000) * 100
```

```
## Accuracy using naive.bayes function:
```

```
## [1] 69.3
```

In naive bayes we use conditional independence assumptions. “S” is a parent node and others are children. “S” has an outgoing arc for each explanatory variable. Here all features are conditionally independent given class label (“S”).

Question 5

Explain why you obtain the same or different results in the exercises (2-4).

Measure	True.BN	Exercise.2	Exercise.3	Exercise.4
Accuracy	73.4	73.4	73.4	69.3

From the above table, it is evident that in exercise 2 and 3, we got the same accuracy when compared to True Asia BN. This is because the L and B nodes are the children of S node in the graph for both exercises and the prediction will be the same.

In exercise 4, using naive bayes, all features are conditionally independent given “S” node (here all nodes are independent and use the posterior probability of the target variable (“S”) for classification). Adding all nodes to the network makes the network dense and noiser. Because nodes “B” and “L” are enough to classify “S”. Hence, the accuracy is 0.693 which is lesser than the true Asia BN classification. So, the naive bayes classifier prediction seems to be not good than other classification.

Reference

1. http://www.bnlearn.com
2. https://cran.r-project.org/web/packages/gRain/vignettes/gRain-intro.pdf
3. http://gauss.inf.um.es/umur/xjurponencias/talleres/J3.pdf

Appendix

```r
knitr::opts_chunk$set(echo = TRUE)

library(bnlearn)

# load data
data("asia")

# ... choose an algorithm and learn the structure of the network from the data...

hc1 = hc(asia, score = "bde", restart = 5, iss=100)
hc2 = hc(asia, score = "bde", restart = 5, iss=10)

# iss set to a very small value to reduce the relative weight of the prior distribution

# ... plot it...

#source("http://bioconductor.org/biocLite.R")
#biocLite(c("graph", "RBGL", "Rgraphviz"))
#install.packages("gRain", dependencies=TRUE)

library(gRain)
library(Rgraphviz)

par(mfrow=c(1,2))
graphviz.plot(hc1, main="BN1 with iss=100")
graphviz.plot(hc2, main="BN2 with iss=10")

#library(bnviewer)
#viewer(hc1,bayesianNetwork.title="Bayesian Network 1 - Asia")
#viewer(hc2,bayesianNetwork.title="Bayesian Network 2 - Asia")


################## COMPARISON ###########################
# ARCS
all.equal(arcs(hc1),arcs(hc2))

# Score - to verify the goodness of fit of the learned network with respect to
# a particular score function
#score(hc1,asia, type = "bde")

# VSTRUCTS
vstructs(hc1)
vstructs(hc2)

# GRAPHICAL COMPARISON
graphviz.compare(hc1,hc2)

# SKELETONS
all.equal(hc1,hc2)
all.equal(cpdag(hc1),cpdag(hc2))
all.equal(skeleton(hc1),skeleton(hc2))

# counts the number of arcs that are different between two network
paste("Hamming distance between hc1 and hc2:",hamming(hc1,hc2))

# split data

n=dim(asia)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.8))
train=asia[id,]
test=asia[-id,]

# learn structure of asia

hc1 = hc(train, score="bde")
graphviz.plot(hc1,shape = "ellipse", main = "Bayesian Network representation")
train_fit <- bn.fit(hc1, train)

# gRain steps when performing inference:
# • compile: moralize, triangulate, find RIP ordering, form initial clique potentials
# • optional: absorb evidence
# • propagate: transform clique potentials to clique marginals

# querygrain performs those steps internally, no need to do them oneself


# network
hc2 <- as.grain(train_fit)
summary(hc2)

# The compile() method performs the following steps:
# (i) Creates the moral graph.
# (ii) Detects that the moral graph is not triangulated and therefore creates a triangulated graph
# by making fill-ins.
# (iii) Establishes a potential representation by absorbing each CPT into an appropriate
# clique potential; i.e., establish the representation in (8.
# (iv) Creates a junction tree of the cliques.

library(gRbase)
hc2 <- compile(hc2)

tabel<-matrix(c(0,0,0,0),nrow=2,ncol=2)

for(i in 1:1000){
  z<-NULL
  for(j in c("A","T","L","B","E","X","D")){
    if(test[i,j]=="no"){
      z<-c(z,"no")
    }
    else{
      z<-c(z,"yes")
    }
  }
  hc3<-setFinding(hc2,nodes=c("A","T","L","B","E","X","D"),states=z) # Enter the evidence
  x<-querygrain(hc3,c("S")) # network with the findings queried
  y <- ifelse(x$S[1]>x$S[2],1,2)
  if(test[i,2]=="yes"){tabel[y,2]<-tabel[y,2]+1}
  else{tabel[y,1]<-tabel[y,1]+1}
}
accuracy_2 = (sum(diag(tabel))/ 1000) * 100
cat("Confusion matrix for Hill climbing : \n")
tabel

cat("Accuracy for Hill climbing : \n")
accuracy_2

# probability of specific finding
# getFinding(asia.gr)


# approximate inference
set.seed(0)
samples.asia <- cpdist(train_fit, nodes = "S", evidence=(S == "yes"), n=5000)

ep <- prop.table(table(samples.asia))
ep

# cpquery(train_fit, event = (A == "a"), evidence = (L == "a"))

# Comparing

dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
dag

train_fit <- bn.fit(dag, train)

# network
hc2 <- as.grain(train_fit)

hc2 <- compile(hc2)

tabel<-matrix(c(0,0,0,0),nrow=2,ncol=2)

for(i in 1:1000){
  z<-NULL
  for(j in c("A","T","L","B","E","X","D")){
    if(test[i,j]=="no"){
      z<-c(z,"no")
    }
    else{
      z<-c(z,"yes")
    }
  }
  hc3<-setFinding(hc2,nodes=c("A","T","L","B","E","X","D"),states=z) # Enter the evidence
  x<-querygrain(hc3,c("S")) # network with the findings queried
  y <- ifelse(x$S[1]>x$S[2],1,2)
  if(test[i,2]=="yes"){tabel[y,2]<-tabel[y,2]+1}
  else{tabel[y,1]<-tabel[y,1]+1}
}
accuracy_true = (sum(diag(tabel))/ 1000) * 100
cat("Confusion matrix for True Asia BN : \n")
tabel

cat("Accuracy for True Asia BN : \n")
accuracy_true
################################################
m <- moralize(asia.gr$dag)
t <- triangulate(m)

library(gRbase)
getCliques()

asia.gr <- compile(asia.gr)
# origpot contains the clique potentials, i.e., initial cliques
asia.gr$origpot[[1]]

asia.gr <- propagate(asia.gr)

q <- querygrain(asia.gr, nodes = c("smoke", "dysp"), type = "joint")
q

set.seed(0)
ep <- cpquery(train_fit, )

set.seed(0)
samples.asia <- cpdist(asia.fit, nodes = c("smoke", "dysp"),
evidence=(asia == "yes"),
n=5000)


ep <- prop.table(table(samples.asia))
ep <- ep['no', 'yes']
gray <- setEvidence(asia.gr, nodes = "asia", states = c("yes"))
q <- querygrain(gray, nodes = c("smoke", "dysp"), type = "joint")
tp <- q['no', 'yes']
abs(ep - tp)


# ... and perform inference to answer any question that interests you!
cpquery(fitted, event = (A == "a"), evidence = (L == "a"))
```

```r
# Learning the equivalence class of a directed acyclic graph (DAG) from data using the
# Grow-Shrink Algorithm
# res = gs(train)

# learning by HC algo

hc0 = hc(train, score="bde")

highlight.opts <- list(nodes = c("L","B"), col = "red", fill = "yellow")

graphviz.plot(hc0, highlight = highlight.opts, main="Markov blanket of S highlighted")
train_fit <- bn.fit(hc0, train)

# Markov blanket of S
mb(train_fit, node=c("S"))

# compile
hc1 <- as.grain(train_fit)
hc2 <- compile(hc1)

tabel<-matrix(c(0,0,0,0),nrow=2,ncol=2)

for(i in 1:1000){
  z<-NULL
  for(j in c("L","B")){
    if(test[i,j]=="no"){
      z<-c(z,"no")
    }
    else{
      z<-c(z,"yes")
    }
  }
  hc3<-setFinding(hc2,nodes=c("L","B"),states=z) # Enter the evidence
  x<-querygrain(hc3,c("S")) # network with the findings queried
  y <- ifelse(x$S[1]>x$S[2],1,2)
  if(test[i,2]=="yes"){tabel[y,2]<-tabel[y,2]+1}
  else{tabel[y,1]<-tabel[y,1]+1}
}
accuracy_3 = (sum(diag(tabel))/ 1000) *100


cat("Confusion matrix: \n")
tabel

cat("Accuracy: \n")
accuracy_3

# split data
n=dim(asia)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.8))
train=asia[id,]
test=asia[-id,]
```

```r
# creating network by hand
dag = model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")
highlight.opts <- list(nodes = c("S"), col = "red", fill = "yellow")

graphviz.plot(dag, main = "Bayesian network representation of the Naive Bayes model",
              highlight = highlight.opts)
hc1<-bn.fit(dag,train)
hc2<-as.grain(hc1)
hc2<-compile(hc2)
# compile = moralize, triangulate, find RIP ordering, form initial clique potentials

tabel<-matrix(c(0,0,0,0),nrow=2,ncol=2) # confusion matrix
for(i in 1:1000){
  z<-NULL
  for(j in c("A","T","L","B","E","X","D")){
    if(test[i,j]=="no"){
      z<-c(z,"no")
    }
    else{
      z<-c(z,"yes")
    }
  }
  hc3<-setFinding(hc2,nodes=c("A","T","L","B","E","X","D"),states=z) # Enter the evidence
  x<-querygrain(hc3,c("S")) # network with the findings queried
  y <- ifelse(x$S[1]>x$S[2],1,2)
  if(test[i,2]=="yes"){tabel[y,2]<-tabel[y,2]+1}
  else{tabel[y,1]<-tabel[y,1]+1}
}
accuracy_4 = (sum(diag(tabel))/ 1000) *100

cat("Confusion matrix for Naive bayes classifier : \n")
tabel

cat("Accuracy for Naive bayes classifier : \n")
accuracy_4
# to compare the above
# naive bayes parameter learning

bn = naive.bayes(train, "S")
fitted = bn.fit(bn, train)
pred = predict(fitted, test)
confus_matrix = table(pred, test[, "S"])
accuracy_naive = ((confus_matrix[1,1] + confus_matrix[2,2]) / 1000) * 100

cat("Accuracy using naive.bayes function: \n")
accuracy_naive
library(kableExtra)

df=data.frame("Measure"="Accuracy", "True BN" = accuracy_true,
              "Exercise 2" = accuracy_2, "Exercise 3" = accuracy_3,
              "Exercise 4" = accuracy_4)

knitr::kable(df)%>%
kable_styling(position = "center") %>%
row_spec(0, bold=TRUE)
```