732A96 Advanced Machine Learning: Lab 1

Rebin Hosini, rebho 150

2017-09-04

Question 1

From a bayesian perspecitve the bayesian network can be decomposed in to following formula:

$$p(G, \theta|Data) \propto p(G|Data)p(\theta|G, Data)$$

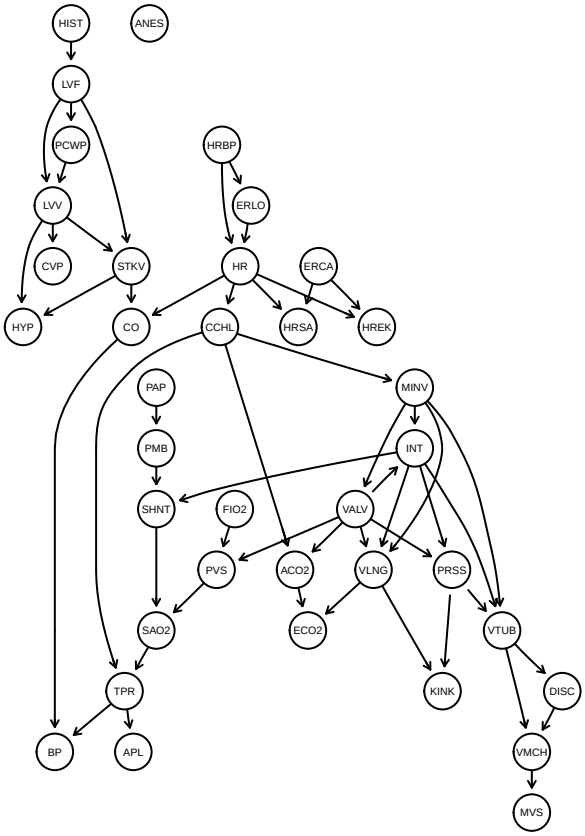
where $p(G|Data)$ is commonly refered to as structural learning and $p(\theta|G, Data)$ is refered to as paratmeter learning. The first step before estimating the parameters is the structural learning which will be performed in this exersize. The main purpose of the structural learning is to find a DAG that maximizes $p(G|Data)$ [1].

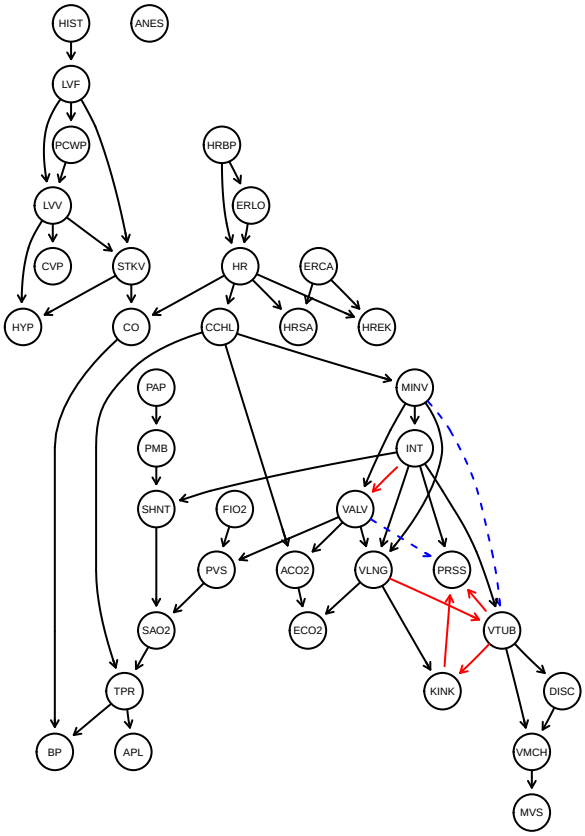
The *hc* function in *bnlearn* package is called the hill climbing algorithm. In a bayesian network, this algoritm will start with an initial graph structure and then calculate the score based on it. The hill climbing algorithm moves in the direction of the increasing score. That is, It checks the neighbours of the the inital score and chooses the path which will increase the score. The algorithm stops when both neigbours to current score give decreasing scores. One common problem with the hill climbin algorithm is that it only finds the local optimum. In this exersise we identify the v-structures which tells us about where unshielded colliders exist and this is something that helps us identify non-equivalent DAGS. We can observe the red arcs in the right hand Figure below which are the ones that changes when we compare two hill-climbing algorithms with the same settings. Since this might not occur in the first observation we have a while loop that stops once the two different hill-climbing algorithms differ. The path that is chosen from the initial graph is random and this is the reason that we can get different structures.

```
#Libraries
library(bnlearn)
library(gRain)
library(Rgraphviz)
library(knitr)

#Comparing arcs, For this case we are going to use the alarm data
countinue <- TRUE
while(countinue){
  hc1 <- hc(alarm, restart = 10)
  hc2 <- hc(alarm, restart = 10)
  continue <- ifelse(all.equal(vstructs(hc1), vstructs(hc2)) == TRUE, TRUE, FALSE)
  if (continue !=TRUE){
    par(mfrow = c(1,2))
    graphviz.compare(hc1, hc2)
    par(mfrow = c(1,1))
    break
  }
}
```

[1] G = Graph, θ = parameters


HIST
ANES
LVF
PCWP
HRBP
LVV
ERLO
CVP
STKV
HR
ERCA
HYP
CO
CCHL
HRSA
HREK
MINV
PAP
INT
PMB
SHNT
FIO2
VALV
PVS
ACO2
VLNG
PRSS
SAO2
ECO2
VTUB
TPR
KINK
DISC
BP
APL
VMCH
MVS


HIST
ANES
LVF
PCWP
HRBP
LVV
ERLO
CVP
STKV
HR
ERCA
HYP
CO
CCHL
HRSA
HREK
MINV
PAP
INT
PMB
SHNT
FIO2
VALV
PVS
ACO2
VLNG
PRSS
SAO2
ECO2
VTUB
TPR
KINK
DISC
BP
APL
VMCH
MVS

Question 2

In the Figure below we can see that the number of arcs from the hill-climbing algorithm is increasing as a function of the imaginary sample size (ISS) hence the DAGs become more densely connected as we increase the ISS. The reason for it becoming less regularized: \

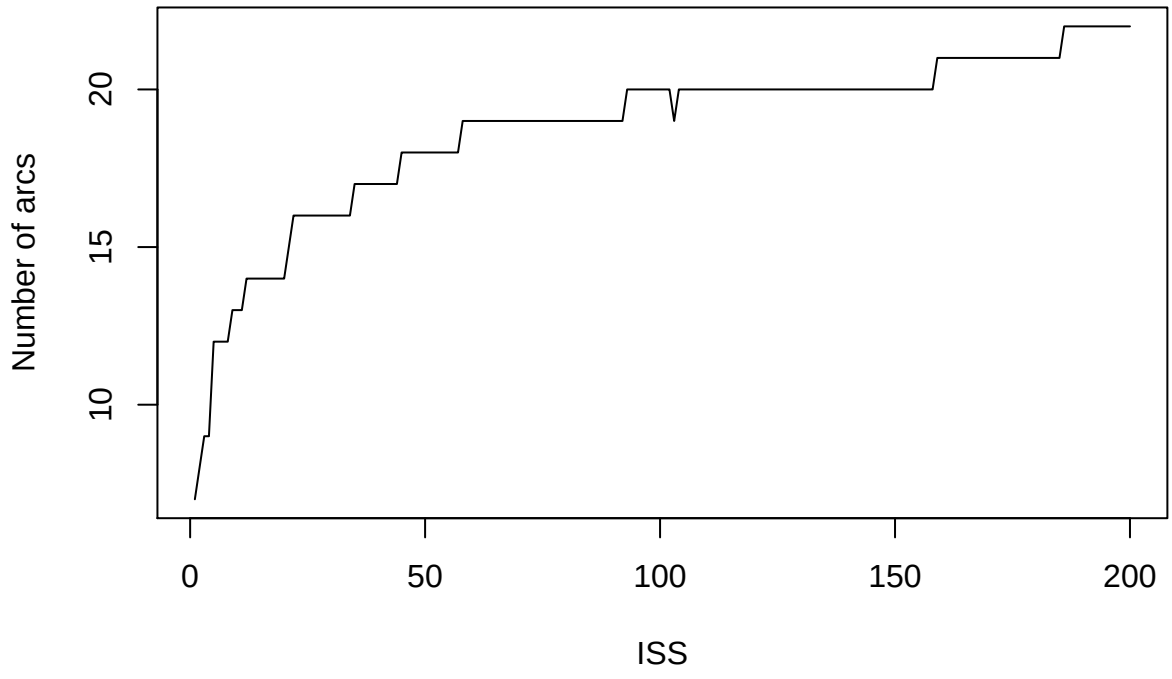
To exlain this we will use a simplified case of the dirchlet, the beta distribution. If we assume a balanced beta prior $beta(\alpha, \beta)$ where $\alpha + \beta = ISS$. The larger the ISS we have the more narrow the beta distribution will become. Therefore, we get quite certain that the prior parameter value is 0.5. If this is the case, we know that:

$$p(A|B = 0) \sim 0.5$$
$$p(A|B = 1) \sim 0.5$$

We can almost say that A is independent of B since $p(B) = 0.5$ and $p(A) = 0.5$ and we will choose drawing an node between the two parameters since we want to include this in our prior. Although, this might lead to an overfitted model. If we instead assume that we have wide beta distribution which means a low ISS we will not be certain that the dependencies are 0.5 and we rather choose the independent version since we do not want to include information we are not certain about in our prior which will lead to less nodes (arcs). In our case we have a dirchlet but we are using beta to explain this in a simpler way. As we get more arcs the regularization will therefore decrease and the other way around.

```
#Increasing iss as a function of number of arcs, that is density of arcs
lhigh <- list()
nArc <- c()
for (i in 1:200){
  lhigh[[i]] <- hc(asia, score = "bde", iss = i)
  nArc[i] <- nrow(arcs(lhigh[[i]]))
}

#Plot
plot(y = nArc, x = 1:200, type = "l", xlab = "ISS", ylab = "Number of arcs")
```



Question 3

Tables with condititonal and unconditional probabilities for both exact and approximate inference are presented below. We can see that they differ a bit. The exact inference is using LS with junction trees and the approximate is using rejection sampling.

In the Figure we can observe the accuracy of the approximate inference. When we add more conditioning we will get more variation. The reason for this is that we are becoming more specific and the accuracy is based on the compatible observations. The general reason for getting variation in the approximate inference is that we are using markov chains.

Table 1: Lung cancer conditional on smoking

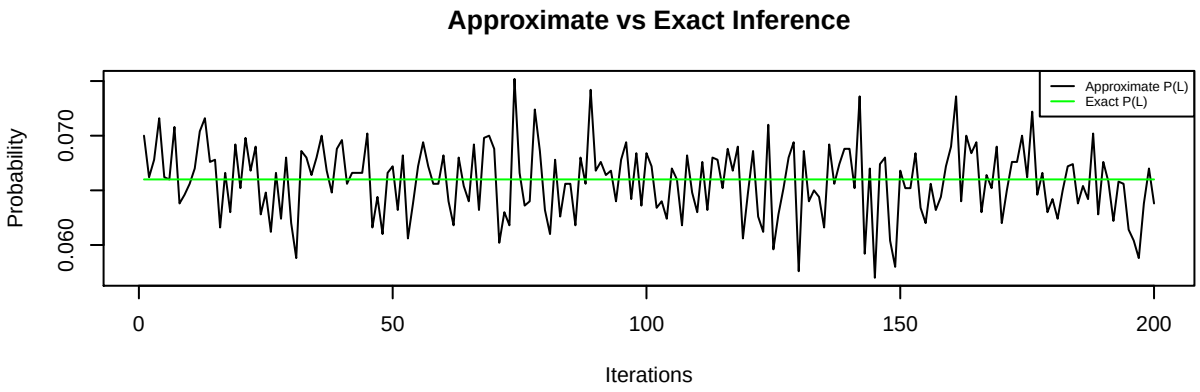
	Lung.cancer	Not.lung.cancer
bnlearn	0.1242461	0.8757539
rGrain	0.1176938	0.8823062

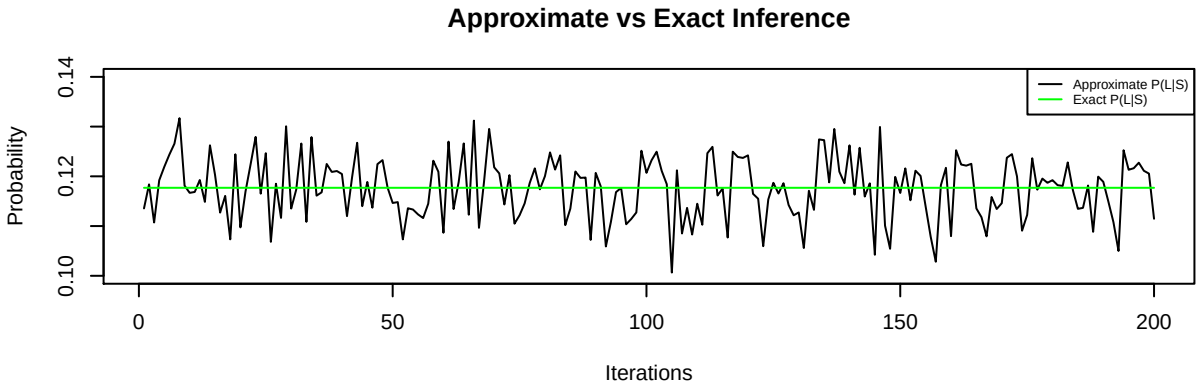
Table 2: Dyspnoea conditional on smoking and visit to asia

	Lung.cancer	Not.lung.cancer
bnlearn	0.5555556	0.4444444
rGrain	0.6206461	0.3793539

Table 3: Approximate inference: Conditional and unconditional probabilities

	Lung.cancer	Dyspnoea
Unconditional	0.0694000	0.4788000
Conditional	0.1242461	0.5555556





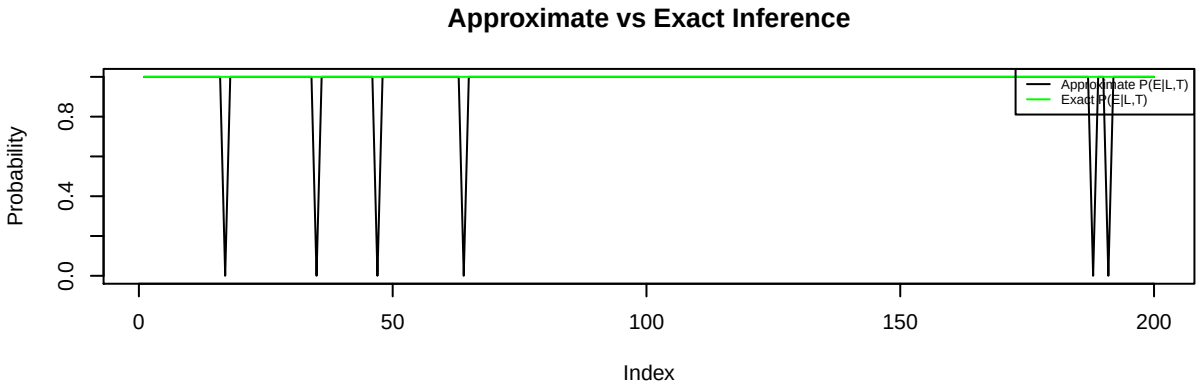


Table 4: Conditional Probabilities for Figure

Event	Evidence
L = Yes	None
L = Yes	S = Yes
E = Yes	L = Yes & T = No

Question 4

In Table 4 we can see the fractions with different values of the parameters *burn.in* and *every*. The parameter *burn.in* is the number of iterations until the algorithm of interest (in our case $ic - dag$) converges into a stationary process. The high value of *every* will result in a network that is diverse. It can be good to perform structure learning in the space of essential graphs since we can focus on unique ones instead of wasting memory and time doing the same operations on same graphs.

```
#Function for generating fractions
fraction <- function(burncoeff = 12, every = 6, n = 29281){
  dags <- random.graph(LETTERS[1:5],
                       num = n, method = "ic-dag",
                       burn.in = burncoeff * length(nodes)^2,
                       every = every)
  uniqueDags <- unique(dags)
  essentialDag <- lapply(uniqueDags, function(x) cpdag(x))
  return(length(unique(essentialDag))/n)
}


#Different values of fractions
fraction12.1 <- fraction(n = 10000)
fraction6.1 <- fraction(burncoeff = 6, n = 10000)
fraction6.6 <- fraction(burncoeff = 6, every = 6, n = 10000)
fraction12.6 <- fraction(every = 6, n = 10000)
fraction6.12 <- fraction(burncoeff = 6, every = 12, n = 10000)
fraction12.12 <- fraction(burncoeff = 12, every = 12, n = 10000)

#Table for comparing fractions
fractionData <- data.frame("burnIn=6" = c(fraction6.1,fraction6.6,fraction6.12),
                           "burnIn=12" = c(fraction12.1,fraction12.6, fraction12.12),
                           row.names = c("every = 1", "every = 6", "every = 12"))


kable(fractionData, caption = "Fractions with different burnIns and every")
```

Table 5: Fractions with different burnIns and every

	burnIn.6	burnIn.12
every = 1	0.4467	0.4538
every = 6	0.4540	0.4396
every = 12	0.4573	0.4589