examhelpaml

Omkar Bhutra (omkbh878)

9 January 2020

yaml setup for lab 1

LAB 1 stuff

Task 1:

```
data("asia")
asia <- as.data.frame(asia)
#default parameter run of HC
hillClimbingResults = hc(asia)
print(hillClimbingResults)
```

```
##
##   Bayesian network learned via Score-based methods
##
##   model:
##    [A][S][T][L|S][B|S][E|T:L][X|E][D|B:E]
##   nodes:                                 8
##   arcs:                                  7
##     undirected arcs:                     0
##     directed arcs:                       7
##   average markov blanket size:           2.25
##   average neighbourhood size:            1.75
##   average branching factor:              0.88
##
##   learning algorithm:                    Hill-Climbing
##   score:                                 BIC (disc.)
##   penalization coefficient:              4.258597
##   tests used in the learning procedure:  77
##   optimized:                             TRUE
```

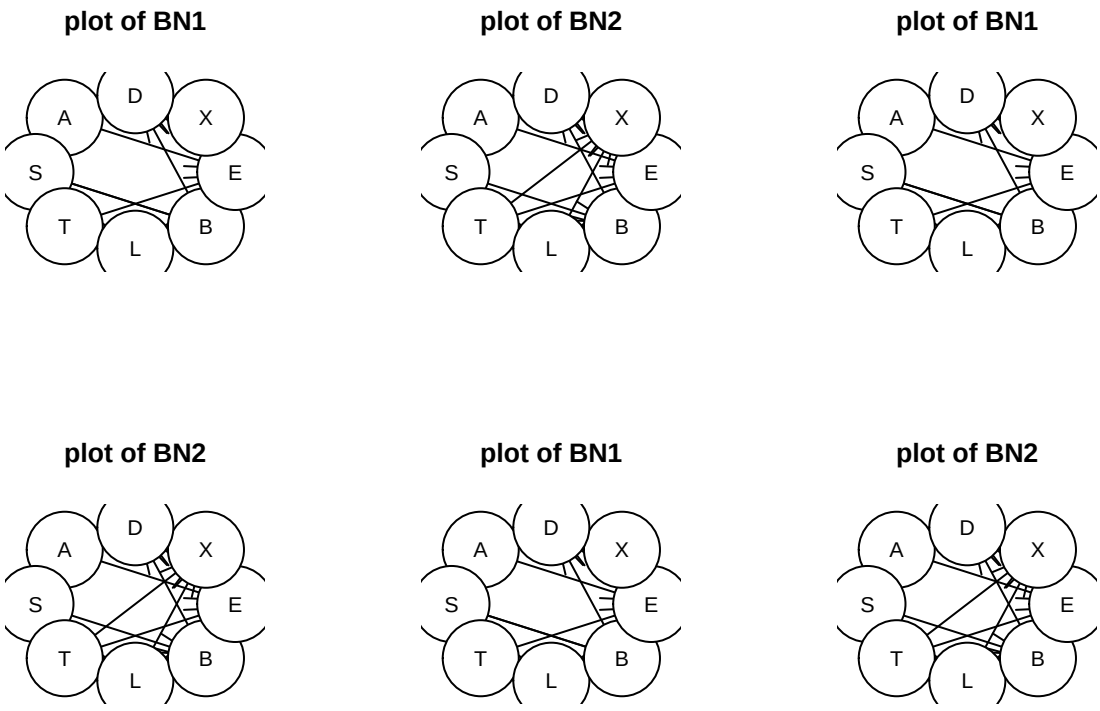
```
par(mfrow = c(2,3))

hillclimb <- function(x){
  hc1 <- hc(x,restart = 15,score = "bde", iss = 3)
  hc2 <- hc(x,restart = 10,score = "bde", iss = 5)
  hc1dag <- cpdag(hc1)
  plot(hc1dag, main = "plot of BN1")
  hc1arcs <- vstructs(hc1dag, arcs = TRUE)
  # print(hc1arcs)
  arcs(hc1)
  hc2dag <- cpdag(hc2)
  plot(hc2dag, main = "plot of BN2")
  hc2arcs <- vstructs(hc2dag, arcs = TRUE)
```

```
  # print(hc2arcs)
  arcs(hc2)
  print(all.equal(hc1dag,hc2dag))
}
for(i in 1:3){
  hillclimb(asia)
}
```

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

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




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

Non-equivalant solutions are produced using different starting parameters such as number of restarts, scoring algorithm, imaginary sample size etc. Hill climbing algorithm is a deterministic one, and hence generates different results on different input parameters. Hill climbing leads to a local maxima of the objective function which makes two different results possible with the same code. Adding imaginary sample size affects the relationships between the nodes such that we get more edges. Number of restarts increases the possibility of getting the same result.

Task 2:

Network structure is trained by the hill climbing algorithm using the BDE score. bn.fit is used to learn the paramets using maximum liklihood estimation. To predict S in the test data , evidence in the network is set to the values of the test data excluding S, using setEvidence. querygrain is used to find the probabilty, maximum of the values is taken to find the misclassification rate.

```
#setting up training and testing datasets
id <- sample(x = seq(1, dim(asia)[1], 1),
             size = dim(asia)[1]*0.8,
             replace = FALSE)
asia.train <- asia[id,]
asia.test <- asia[-id,]

bnprediction = as.numeric()
#fitting a model using Hill Climbing
bnmodel <- hc(asia.train,score = "bde",restart = 50)
bnmodelfit <- bn.fit(bnmodel,asia.train,method = 'mle') #max liklihood estimation
bngrain <- as.grain(bnmodelfit)
comp <- compile(bngrain)

bnmodel_true <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
bnmodelfit_true <- bn.fit(bnmodel_true, asia.train)
bngrain_true <- as.grain(bnmodelfit_true)
comp_true <- compile(bngrain_true)

bnpredict <- function(bntree, data, obs_variables, pred_variable) {
  for (i in 1:dim(data)[1]) {
    x <- NULL
    for (j in obs_variables) {
      x[j] <- if(data[i, j] == "yes") "yes" else "no"
    }
    evidence <- setEvidence(object = bntree,nodes = obs_variables,states=x)
    prob_dist_fit  <- querygrain(object = evidence,nodes = pred_variable)$S
    bnprediction[i] <- if (prob_dist_fit["yes"] >= 0.5) "yes" else "no"
  }
  return(bnprediction)
}

# Predict S from Bayesian Network and test data observations
bnprediction <- bnpredict(bntree = comp,
                          data = asia.test,
                          obs_variables = c("A", "T", "L", "B", "E", "X", "D"),
                          pred_variable = c("S"))
bnprediction_true <- bnpredict(bntree = comp_true,
                               data = asia.test,
                               obs_variables = c("A", "T", "L", "B", "E", "X", "D"),
                               pred_variable = c("S"))

# Calculate confusion matricies
confusion_matrix_fit <- table(bnprediction, asia.test$S)
print(confusion_matrix_fit)
```

```
##
```

```
## bnprediction  no yes
##          no  349 112
##          yes 158 381
```

```
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit))/sum(confusion_matrix_fit)))
```

```
## [1] "Misclassification rate: 0.27"
```

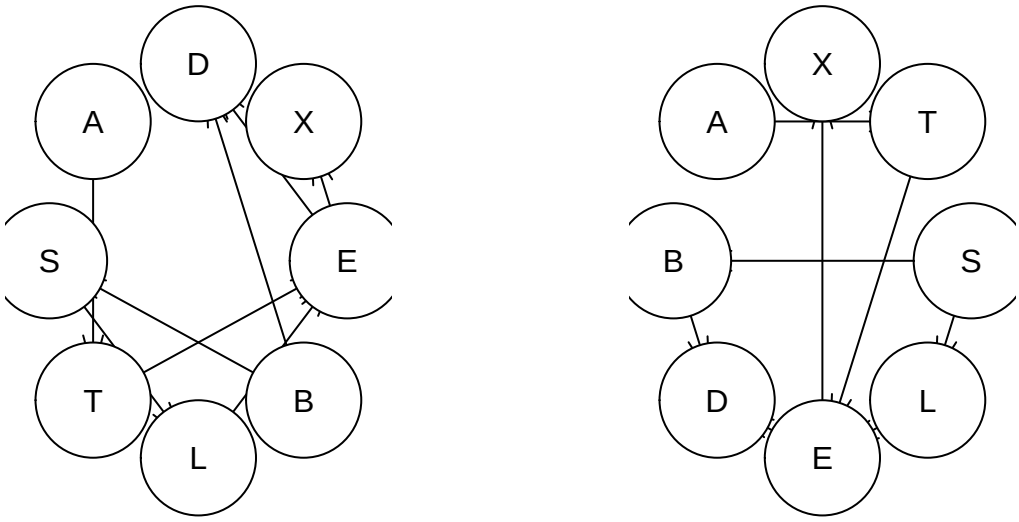
```
confusion_matrix_true <- table(bnprediction_true, asia.test$S)
print(confusion_matrix_true)
```

```
##
## bnprediction_true  no yes
##               no  349 112
##               yes 158 381
```

```
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_true))/sum(confusion_matrix_true)))
```

```
## [1] "Misclassification rate: 0.27"
```

```
par(mfrow=c(1,2))
plot(bnmodel)
title("hill climb network")
plot(bnmodel_true)
title("True network")
```



```r
par(mfrow=c(1,1))
```

The Misclassification rate remain the same for both the hill climb network and the true network. However, The state of "Different number of directed/undirected arcs" persists.

Task 3:

Using the markov blanket, we predict the results again, it is expected to output the same result.

```r
markov_blanket = mb(bnmodel, c("S"))
markov_blanket_true = mb(bnmodel_true, c("S"))
bnpredict_mb <- bnpredict(comp, asia.test, markov_blanket, c("S"))
bnpredict_mb_true <- bnpredict(comp_true, asia.test, markov_blanket_true, c("S"))
confusion_matrix_fit <- table(bnpredict_mb, asia.test$S)
confusion_matrix_fit_true <- table(bnpredict_mb_true, asia.test$S)
print(confusion_matrix_fit)
```

```
##
## bnpredict_mb  no yes
##          no  349 112
##          yes 158 381
```

```r
print(confusion_matrix_fit_true)
```

```
##
## bnpredict_mb_true  no yes
##               no  349 112
##               yes 158 381
```

```r
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit))/sum(confusion_matrix_fit)))
```

```
## [1] "Misclassification rate: 0.27"
```

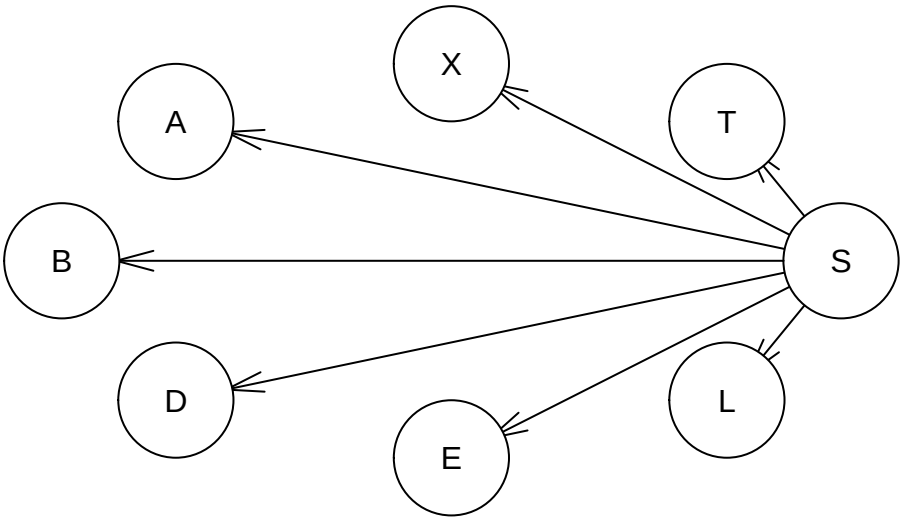
```r
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit_true))/sum(confusion_matrix_fit_
```

```
## [1] "Misclassification rate: 0.27"
```

Task 4:

The naiveBayes structure implies that all variables are independant of S, this implies that there is a diffrent true structure used.

```r
naiveBayes = model2network("[S][A|S][B|S][D|S][E|S][X|S][L|S][T|S]")
plot(naiveBayes, main = "Naive Bayes")
```




```
naiveBayes.fit <- bn.fit(naiveBayes, asia.train)
result_naive <- bnpredict(compile(as.grain(naiveBayes.fit)), asia.test, c("A", "T", "L", "B", "E", "X",
# Calculate confusion matricies
confusion_matrix_naive_bayes <- table(result_naive, asia.test$S)
print(confusion_matrix_naive_bayes)
```

```
##
## result_naive  no yes
##          no  369 181
##          yes 138 312
```

```
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_naive_bayes))/sum(confusion_matrix_na
```

```
## [1] "Misclassification rate: 0.319"
```

Task 5:

In Task 2, Same results are observed for the trained structure and the true structure. Same results are also obtained in Task 3 as before, this is expected due to the reduction of the node from the rest of the structure. The reduced number of predictors still provides the same misclassification rate although some information loss is expected. In Task 4, Naive bayes is not a predictor for the data as the structure used is different as the variables are inferred to be independant of S, but all are assumed to have an effect on S.

```r
knitr::opts_chunk$set(echo = TRUE)
library("dplyr")
library("ggplot2")
library("gRain")
library("bnlearn")
data("asia")
asia <- as.data.frame(asia)
#default parameter run of HC
hillClimbingResults = hc(asia)
print(hillClimbingResults)
par(mfrow = c(2,3))

hillclimb <- function(x){
  hc1 <- hc(x,restart = 15,score = "bde", iss = 3)
  hc2 <- hc(x,restart = 10,score = "bde", iss = 5)
  hc1dag <- cpdag(hc1)
  plot(hc1dag, main = "plot of BN1")
  hc1arcs <- vstructs(hc1dag, arcs = TRUE)
  # print(hc1arcs)
  arcs(hc1)
  hc2dag <- cpdag(hc2)
  plot(hc2dag, main = "plot of BN2")
  hc2arcs <- vstructs(hc2dag, arcs = TRUE)
  # print(hc2arcs)
  arcs(hc2)
  print(all.equal(hc1dag,hc2dag))
}
for(i in 1:3){
  hillclimb(asia)
}
#setting up training and testing datasets
id <- sample(x = seq(1, dim(asia)[1], 1),
             size = dim(asia)[1]*0.8,
             replace = FALSE)
asia.train <- asia[id,]
asia.test <- asia[-id,]

bnprediction = as.numeric()
#fitting a model using Hill Climbing
bnmodel <- hc(asia.train,score = "bde",restart = 50)
bnmodelfit <- bn.fit(bnmodel,asia.train,method = 'mle') #max liklihood estimation
bngrain <- as.grain(bnmodelfit)
comp <- compile(bngrain)

bnmodel_true <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
bnmodelfit_true <- bn.fit(bnmodel_true, asia.train)
bngrain_true <- as.grain(bnmodelfit_true)
comp_true <- compile(bngrain_true)

bnpredict <- function(bntree, data, obs_variables, pred_variable) {
  for (i in 1:dim(data)[1]) {
    x <- NULL
    for (j in obs_variables) {
```

```
      x[j] <- if(data[i, j] == "yes") "yes" else "no"
    }
    evidence <- setEvidence(object = bntree,nodes = obs_variables,states=x)
    prob_dist_fit  <- querygrain(object = evidence,nodes = pred_variable)$S
    bnprediction[i] <- if (prob_dist_fit["yes"] >= 0.5) "yes" else "no"
  }
  return(bnprediction)
}

# Predict S from Bayesian Network and test data observations
bnprediction <- bnpredict(bntree = comp,
                          data = asia.test,
                          obs_variables = c("A", "T", "L", "B", "E", "X", "D"),
                          pred_variable = c("S"))
bnprediction_true <- bnpredict(bntree = comp_true,
                               data = asia.test,
                               obs_variables = c("A", "T", "L", "B", "E", "X", "D"),
                               pred_variable = c("S"))

# Calculate confusion matricies
confusion_matrix_fit <- table(bnprediction, asia.test$S)
print(confusion_matrix_fit)
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit))/sum(confusion_matrix_fit)))

confusion_matrix_true <- table(bnprediction_true, asia.test$S)
print(confusion_matrix_true)
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_true))/sum(confusion_matrix_true)))

par(mfrow=c(1,2))
plot(bnmodel)
title("hill climb network")
plot(bnmodel_true)
title("True network")
par(mfrow=c(1,1))
markov_blanket = mb(bnmodel, c("S"))
markov_blanket_true = mb(bnmodel_true, c("S"))
bnpredict_mb <- bnpredict(comp, asia.test, markov_blanket, c("S"))
bnpredict_mb_true <- bnpredict(comp_true, asia.test, markov_blanket_true, c("S"))
confusion_matrix_fit <- table(bnpredict_mb, asia.test$S)
confusion_matrix_fit_true <- table(bnpredict_mb_true, asia.test$S)
print(confusion_matrix_fit)
print(confusion_matrix_fit_true)
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit))/sum(confusion_matrix_fit)))
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit_true))/sum(confusion_matrix_fit_t
naiveBayes = model2network("[S][A|S][B|S][D|S][E|S][X|S][L|S][T|S]")
plot(naiveBayes, main = "Naive Bayes")

naiveBayes.fit <- bn.fit(naiveBayes, asia.train)
result_naive <- bnpredict(compile(as.grain(naiveBayes.fit)), asia.test, c("A", "T", "L", "B", "E", "X",
# Calculate confusion matricies
confusion_matrix_naive_bayes <- table(result_naive, asia.test$S)
print(confusion_matrix_naive_bayes)
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_naive_bayes))/sum(confusion_matrix_na
```

```r
# Question 1: PGMs

# Learn a BN from the Asia dataset, find a separation statement (e.g. B _|_ E | S, T) and, then, check
# it corresponds to a statistical independence.
library(bnlearn)
library(gRain)
set.seed(123)
data("asia")
hc3<-hc(asia,restart=10,score="bde",iss=10)
plot(hc3)
hc4<-bn.fit(hc3,asia,method="bayes")
hc5<-as.grain(hc4)
hc6<-compile(hc5)
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("yes","yes","yes"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("yes","yes","no"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("yes","no","yes"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("yes","no","no"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("no","yes","yes"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("no","yes","no"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("no","no","yes"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("no","no","no"))
querygrain(hc7,c("B"))

# Sample DAGs at random and, then, check which of them coincide with their CPDAGs, i.e. the CPDAGs have
# undirected edge (recall that a CPDAG has an undirected edge if and only if there are two DAGs in the
# equivalence class that differ in the direction of that edge).

#  The exact ratio according to the literature is 11.2
library(bnlearn)
set.seed(123)
ss<-50000
x<-random.graph(c("A","B","C","D","E"),num=ss,method="melancon",every=50,burn.in=30000)

y<-unique(x)
z<-lapply(y,cpdag)

r=0
for(i in 1:length(y)) {
  if(all.equal(y[[i]],z[[i]])==TRUE)
    r<-r+1
}
length(y)/r
# PGMs

#source("https://bioconductor.org/biocLite.R")
#biocLite("RBGL")
```

```r
library(bnlearn)
library(gRain)
set.seed(567)
data("asia")
ind <- sample(1:5000, 4000)
tr <- asia[ind,]
te <- asia[-ind,]

nb0<-empty.graph(c("S","A","T","L","B","E","X","D"))
arc.set<-matrix(c("S", "A", "S", "T", "S", "L", "S", "B", "S", "E", "S", "X", "S", "D"),
                ncol = 2, byrow = TRUE, dimnames = list(NULL, c("from", "to")))
arcs(nb0)<-arc.set
#nb<-naive.bayes(tr,"S")
plot(nb0)

totalError<-array(dim=6)
k<-1
for(j in c(10,20,50,100,1000,2000)){
  nb<-bn.fit(nb0,tr[1:j,],method="bayes")
  nb2<-as.grain(nb)
  nb2<-compile(nb2)

  error<-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(te[i,j]=="no"){
        z<-c(z,"no")
      }
      else{
        z<-c(z,"yes")
      }
    }

    nb3<-setFinding(nb2,nodes=c("A","T","L","B","E","X","D"),states=z)
    x<-querygrain(nb3,c("S"))

    if(x$S[1]>x$S[2]){
      y<-1
    }
    else{
      y<-2
    }

    if(te[i,2]=="no"){
      error[y,1]<-error[y,1]+1
    }
    else{
      error[y,2]<-error[y,2]+1
    }
  }
  totalError[k]<-(error[1,2]+error[2,1])/1000
  k<-k+1
```

```
}
totalError

nb0<-empty.graph(c("S","A","T","L","B","E","X","D"))
arc.set<-matrix(c("A", "S", "T", "S", "L", "S", "B", "S", "E", "S", "X", "S", "D", "S"),
                ncol = 2, byrow = TRUE, dimnames = list(NULL, c("from", "to")))
arcs(nb0)<-arc.set
#nb<-naive.bayes(tr,"S")
plot(nb0)

totalError<-array(dim=6)
k<-1
for(j in c(10,20,50,100,1000,2000)){
  nb<-bn.fit(nb0,tr[1:j,],method="bayes")
  nb2<-as.grain(nb)
  nb2<-compile(nb2)

  error<-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(te[i,j]=="no"){
        z<-c(z,"no")
      }
      else{
        z<-c(z,"yes")
      }
    }

    nb3<-setFinding(nb2,nodes=c("A","T","L","B","E","X","D"),states=z)
    x<-querygrain(nb3,c("S"))

    if(x$S[1]>x$S[2]){
      y<-1
    }
    else{
      y<-2
    }

    if(te[i,2]=="no"){
      error[y,1]<-error[y,1]+1
    }
    else{
      error[y,2]<-error[y,2]+1
    }
  }
  totalError[k]<-(error[1,2]+error[2,1])/1000
  k<-k+1
}
totalError

# Discussion
# The NB classifier only needs to estimate the parameters for distributions of the form
```

```
# p(C) and P(A_i|C) where C is the class variable and A_i is a predictive attribute. The
# alternative model needs to estimate p(C) and P(C|A_1,...,A_n). Therefore, it requires
# more data to obtain reliable estimates (e.g. many of the parental combinations may not
# appear in the training data and this is actually why you should use method="bayes", to
# avoid zero relative frequency counters).This may hurt performance when little learning
# data is available. This is actually observed in the experiments above. However, when
# the size of the learning data increases, the alternative model should outperform NB, because
# the latter assumes that the attributes are independent given the class whereas the former
# does not. In other words, note that p(C|A_1,...,A_n) is proportional to P(A_1,...,A_n|C) p(C)
# by Bayes theorem. NB assumes that P(A_1,...,A_n|C) factorizes into a product of factors
# p(A_i|C) whereas the alternative model assumes nothing. The NB's assumption may hurt
# performance. This can be observed in the experiments.
# Question 1.1 (Monty Hall)

library(bnlearn)
library(gRain)

MH<-model2network("[D][P][M|D:P]") # Structure
cptD = matrix(c(1/3, 1/3, 1/3), ncol = 3, dimnames = list(NULL, c("D1", "D2", "D3"))) # Parameters
cptP = matrix(c(1/3, 1/3, 1/3), ncol = 3, dimnames = list(NULL, c("P1", "P2", "P3")))
cptM = c(
  0,.5,.5,
  0,0,1,
  0,1,0,
  0,0,1,
  .5,0,.5,
  1,0,0,
  0,1,0,
  1,0,0,
  .5,.5,0)
dim(cptM) = c(3, 3, 3)
dimnames(cptM) = list("M" = c("M1", "M2", "M3"), "D" =  c("D1", "D2", "D3"), "P" = c("P1", "P2", "P3"))
MHfit<-custom.fit(MH,list(D=cptD,P=cptP,M=cptM))
MHcom<-compile(as.grain(MHfit))

MHfitEv<-setFinding(MHcom,nodes=c(""),states=c("")) # Exact inference
querygrain(MHfitEv,c("P"))
MHfitEv<-setFinding(MHcom,nodes=c("D","M"),states=c("D1","M2"))
querygrain(MHfitEv,c("P"))
MHfitEv<-setFinding(MHcom,nodes=c("D","M"),states=c("D1","M3"))
querygrain(MHfitEv,c("P"))

table(cpdist(MHfit,"P",evidence=TRUE)) # Alternatively, one can use approximate inference
table(cpdist(MHfit,"P",(D=="D1" & M=="M2")))
table(cpdist(MHfit,"P",(D=="D1" & M=="M3")))

# Question 1.2 (XOR)

library(bnlearn)
library(gRain)

xor<-model2network("[A][B][C|A:B]") # Structure
cptA = matrix(c(0.5, 0.5), ncol = 2, dimnames = list(NULL, c("0", "1"))) # Parameters
```

```r
cptB = matrix(c(0.5, 0.5), ncol = 2, dimnames = list(NULL, c("0", "1")))
cptC = c(1,0,0,1,0,1,1,0)
dim(cptC) = c(2, 2, 2)
dimnames(cptC) = list("C" = c("0", "1"), "A" =  c("0", "1"), "B" = c("0", "1"))
xorfit<-custom.fit(xor,list(A=cptA,B=cptB,C=cptC))

for(i in 1:10){
  xordata<-rbn(xorfit,1000) # Sample
  xorhc<-hc(xordata,score="bic") # Learn
  plot(xorhc)
}

# The HC fails because the data comes from a distribution that is not faithful to the graph.
# In other words, the graph has an edge A -> C but A is marginally independent of C in the
# distribution. The same for B and C. And, of course, the same for A and B since there is no
# edge between them. So, the HC cannot find two dependent variables to link with an edge and,
# thus, it adds no edge to the initial empty graph.
```