AML lab1

Zijie Feng

2019/9/12

Q(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").

Data `asia` includes eight variables about lung diseases and visits to Asia. Based on different combinations of arguments, we will create many different Bayesian networks and thereby choosing the best one.

```
############### lab1q1 ######################
rm(list=ls())
set.seed(12345)
data("asia")
nodes <- colnames(asia)
nodes
```

```
## [1] "A" "S" "T" "L" "B" "E" "X" "D"
```

```
scores <- c("loglik","aic","bic","bdla","bdj","bde","bds","mbde")
rr <- seq(1,100,20)  # number of restart
iss <- seq(1,10,5)    # set to a very small value to reduce the
                     # relative weight of the prior distribution
min.score <- -999999
best.i <- 0
best.j <- 0
best.k <- 0

for (i in 1:length(scores)) {
  for (j in 1:length(rr)) {

    if(i<6){
      score <- bnlearn::score(hc(asia, score = scores[i],
                                 restart = rr[j]), asia)
        if(score>min.score){
          min.score <- score
          best.i <- i
          best.j <- j
        }
    }
    else{
      for(k in 1:length(iss)){
        score <- bnlearn::score(hc(asia, score = scores[i],
                                   restart = rr[j],iss=iss[k]), asia)
        if(score>min.score){
          min.score <- score
          best.i <- i
          best.j <- j
          best.k <- k
```

```
        }
      }
    }

  }
}
```

```
## minimal scores:  -11107.29
```

```
## score function:  bic
```

```
## restart number:  1
```

```
## imaginary sample size:  0
```

To be more specific, the hill-climbing algorithm will give us non-equivalent BN structures each time, since such algorithm can only provide us the local optimization.

new Bayesian network structures

Here we create another two BNs with `bde` score and different `restart`. It shows that their arcs are different although their scores are the same.

```
set.seed(54321)
mdl1a <- hc(asia, score = "bde")
score(mdl1a, asia)
```

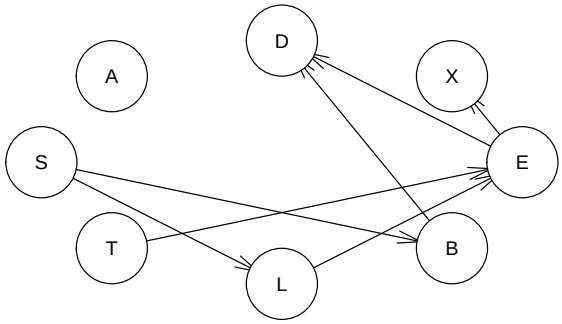
```
## [1] -11107.29
```

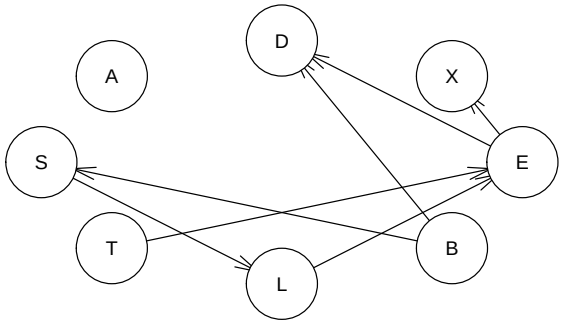
```
set.seed(54321)
mdl1b <- hc(asia, score = "bde", restart = 100)
score(mdl1b, asia)
```

```
## [1] -11107.29
```

```
all.equal(mdl1a, mdl1b)
```

```
## [1] "Different arc sets"
```



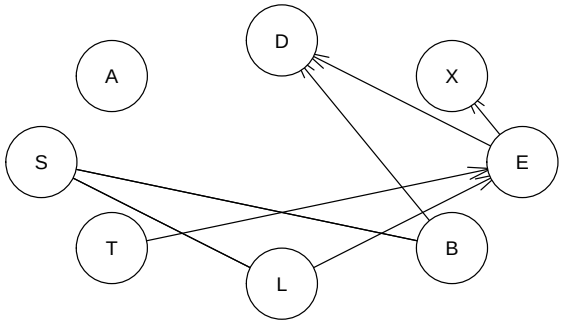


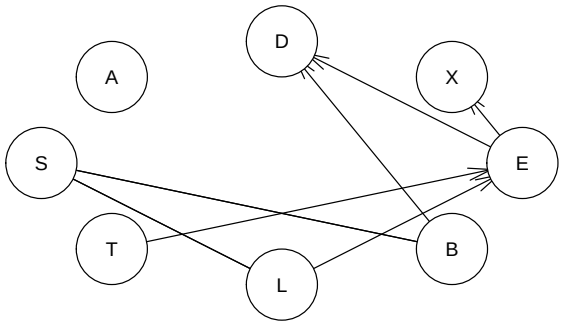
After cpdag:

However, their new structures after `cpdag` are the same.

```
cp1a <- cpdag(mdl1a)
cp1b <- cpdag(mdl1b)
all.equal(cp1a, cp1b)
```

```
## [1] TRUE
```





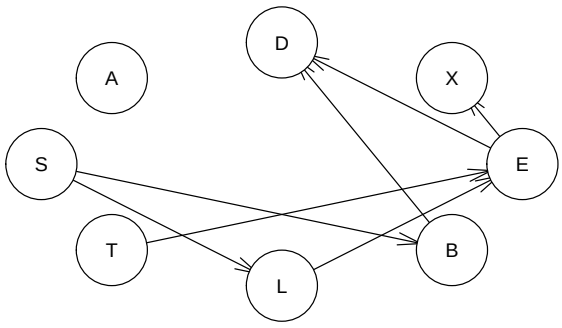
Q(2)

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 S yes and S 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

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

```
############### lab1q2 ####################
idx   <- sample(1:5000, 4000)
train <- asia[idx,]
test  <- asia[-idx,]

bn.hc <- hc(train, score = scores[best.i], restart = rr[best.j])
plot(bn.hc)
```




```
## The best BN structure we have: [A][S][T][L|S][B|S][E|T:L][X|E][D|B:E]
```

Set evidences to the Bayesian network structure and do classification to the test data.

```
#prediction function
pred <- function(mdl, idx){
  bnfit <- bn.fit(mdl, train)
  grain <- as.grain(bnfit)

  res <- matrix(ncol=1, nrow=dim(test)[1])
  for(i in 1:dim(test)[1]){
    state <- c()
    for(j in 1:dim(test)[2]){
      if(test[i,j]=="no"){
        state <- c(state, "no")
```

```
      }else{
        state <- c(state, "yes")
      }
    }
    mdl <- setFinding(grain, nodes=colnames(test)[idx], states=state[idx])
    q =  querygrain(mdl, nodes ="S", type="marginal")$S
    res[i] = ifelse(q[1]<q[2],"yes","no")
  }
  res
}

res.hc <- pred(bn.hc, c(1,3:8))
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
res.true <-pred(dag, c(1,3:8))

table(res.hc, res.true)
```

```
##       res.true
## res.hc  no yes
##    no  460   0
##    yes   0 540
```

```
prop.table(table(res.hc, realS=test$S))
```

```
##       realS
## res.hc    no   yes
##    no  0.334 0.126
##    yes 0.151 0.389
```

The first confusion matrix shows the relation between the prediction of the true Asia Bayesian network and the Bayesian network based on hill-climbing algorithm. Although such two BN structures are different, their predictions are totally same. However, we know that the actual BN's correct rate is not very high. So the difference between such two structures is acceptable.

The second confusion matrix shows the relation between our prediction and the fact of S in test set.

Q(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.

```
############### lab1q3 ####################
mb(bn.hc, "S")
```

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

```
res.mb <- pred(bn.hc, c(4,5))
prop.table(table(res.mb, realS=test$S))
```

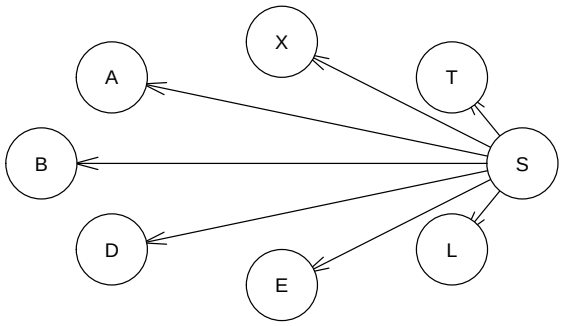
```
##       realS
## res.mb    no   yes
##    no  0.334 0.126
##    yes 0.151 0.389
```

Q(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.

```
############### lab1q4 ####################
## naive.bayes function from bnlearn package
# mdl <- naive.bayes(test, "S")
# plot(mdl)
# p <- predict(mdl, test)
# table(p, test$S)

nb <- model2network("[A|S][S][T|S][L|S][B|S][E|S][X|S][D|S]")
plot(nb)
```



The naive Bayesian classifier is based on an assumption that all the nodes/variables are independent to each other given on the nodes we want to predict.

```
##       realS
## res.nb    no   yes
##    no  0.364 0.196
##    yes 0.121 0.319
```

Q(5)

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

The result of Q(3) is the same as in Q(2), because the probability of S only depends on the the connected nodes shown in our DN structure. The remaining nodes are independent with S absolutely.

The result of Q(4) is worse than previous results, since naive Bayesian network is the most basic BN structure which assumes that all the variables are independent mutually. It might ignore some relation among different variables and thereby affecting our prediction negatively.

Appendix

```
############### lab1 ####################
knitr::opts_chunk$set(echo = TRUE ,out.height = "200px")
library(bnlearn)
# BiocManager::install("RBGL")
library(gRain)
############### lab1q1 ####################
rm(list=ls())
set.seed(12345)
data("asia")
nodes <- colnames(asia)
nodes
scores <- c("loglik","aic","bic","bdla","bdj","bde","bds","mbde")
rr <- seq(1,100,20)  # number of restart
iss <- seq(1,10,5)    # set to a very small value to reduce the
                     # relative weight of the prior distribution
min.score <- -999999
best.i <- 0
best.j <- 0
best.k <- 0

for (i in 1:length(scores)) {
  for (j in 1:length(rr)) {

    if(i<6){
      score <- bnlearn::score(hc(asia, score = scores[i],
                                 restart = rr[j]), asia)
        if(score>min.score){
          min.score <- score
          best.i <- i
          best.j <- j
        }
    }
    else{
      for(k in 1:length(iss)){
        score <- bnlearn::score(hc(asia, score = scores[i],
                                   restart = rr[j],iss=iss[k]), asia)
        if(score>min.score){
          min.score <- score
          best.i <- i
          best.j <- j
          best.k <- k
        }
      }
    }

  }
}
cat("minimal scores: ",min.score,"\n")
cat("score function: ",scores[best.i],"\n")
cat("restart number: ",rr[best.j],"\n")
cat("imaginary sample size: ",best.k,"\n")
set.seed(54321)
```

```r
mdl1a <- hc(asia, score = "bde")
score(mdl1a, asia)

set.seed(54321)
mdl1b <- hc(asia, score = "bde", restart = 100)
score(mdl1b, asia)

all.equal(mdl1a, mdl1b)
plot(mdl1a)
plot(mdl1b)
cp1a <- cpdag(mdl1a)
cp1b <- cpdag(mdl1b)
all.equal(cp1a, cp1b)
plot(cp1a)
plot(cp1b)
############### lab1q2 ####################
idx   <- sample(1:5000, 4000)
train <- asia[idx,]
test  <- asia[-idx,]

bn.hc <- hc(train, score = scores[best.i], restart = rr[best.j])
plot(bn.hc)

cat("The best BN structure we have:",modelstring(bn.hc))
#prediction function
pred <- function(mdl, idx){
  bnfit <- bn.fit(mdl, train)
  grain <- as.grain(bnfit)

  res <- matrix(ncol=1, nrow=dim(test)[1])
  for(i in 1:dim(test)[1]){
    state <- c()
    for(j in 1:dim(test)[2]){
      if(test[i,j]=="no"){
        state <- c(state, "no")
      }else{
        state <- c(state, "yes")
      }
    }
    mdl <- setFinding(grain, nodes=colnames(test)[idx], states=state[idx])
    q =  querygrain(mdl, nodes ="S", type="marginal")$S
    res[i] = ifelse(q[1]<q[2],"yes","no")
  }
  res
}

res.hc <- pred(bn.hc, c(1,3:8))
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
res.true <-pred(dag, c(1,3:8))

table(res.hc, res.true)
prop.table(table(res.hc, realS=test$S))
############### lab1q3 ####################
```

```r
mb(bn.hc, "S")
res.mb <- pred(bn.hc, c(4,5))
prop.table(table(res.mb, realS=test$S))
############### lab1q4 ####################
## naive.bayes function from bnlearn package
# mdl <- naive.bayes(test, "S")
# plot(mdl)
# p <- predict(mdl, test)
# table(p, test$S)

nb <- model2network("[A|S][S][T|S][L|S][B|S][E|S][X|S][D|S]")
plot(nb)
res.nb <- pred(nb, c(1,3:8))
prop.table(table(res.nb, realS=test$S))
```