PGM_Lab1_Groupreport

Maria(marse306),Jasleen(jasma846),vinay(vinbe289),Tejashree(tejma768)

September 20, 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").

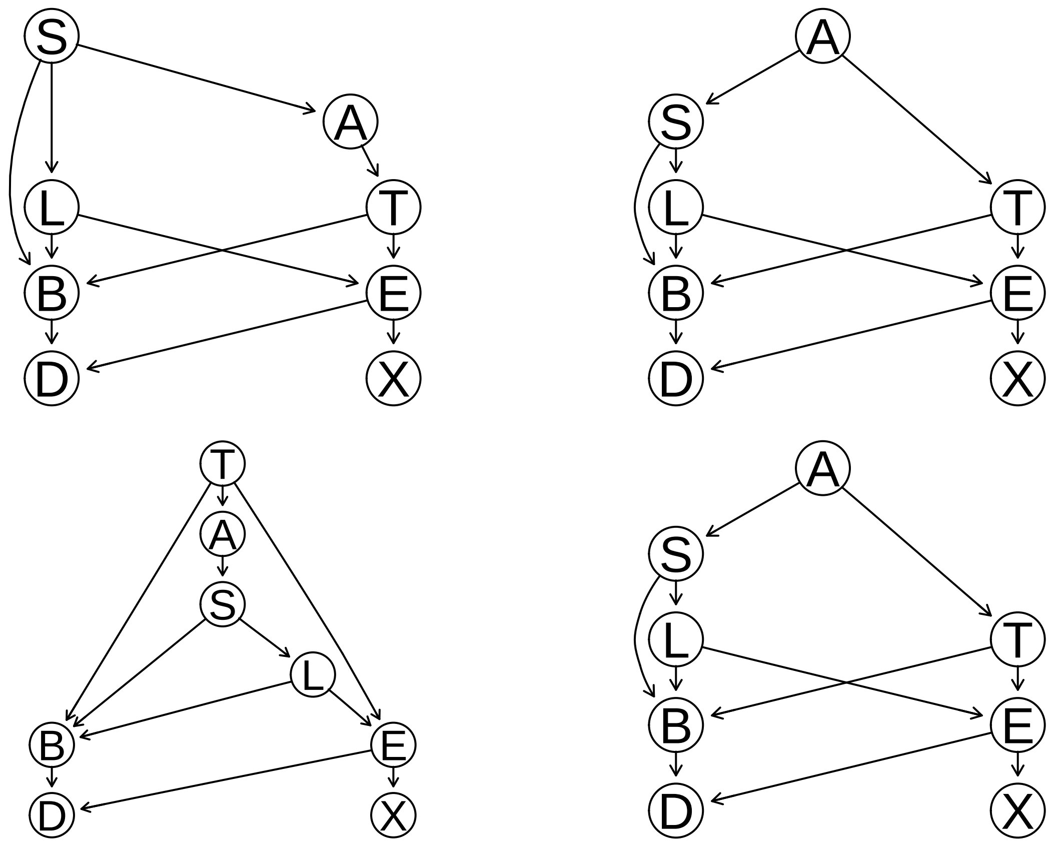
```
bn = list()
arcsBN = list()
vstructsBN = list()
```

```
set.seed(12345)
for(i in 1:4){
  bn[[i]] = hc(asia , restart = 4, score ="aic" )
}

#plot
par(mfrow = c(2,2))
graphviz.plot(bn[[1]])
```

```
## Loading required namespace: Rgraphviz
```

```
graphviz.plot(bn[[2]])
graphviz.plot(bn[[3]])
graphviz.plot(bn[[4]])
```



Results for first HC run

```r
#extract quantities of interest from object
arcs(bn[[1]])
```

```
##       from to
##  [1,] "B"  "D"
##  [2,] "E"  "X"
##  [3,] "S"  "B"
##  [4,] "T"  "E"
##  [5,] "E"  "D"
##  [6,] "S"  "L"
##  [7,] "T"  "B"
##  [8,] "A"  "T"
##  [9,] "L"  "B"
## [10,] "S"  "A"
## [11,] "L"  "E"
```

```r
#cpdag represents the equivalance class
cpdag(bn[[1]])
```

```
##
##   Bayesian network learned via Score-based methods
##
##   model:
##     [partially directed graph]
##   nodes:                                 8
##   arcs:                                  11
##     undirected arcs:                    3
##     directed arcs:                      8
##   average markov blanket size:          3.50
##   average neighbourhood size:           2.75
##   average branching factor:             1.00
##
##   learning algorithm:                   Hill-Climbing
##   score:                                AIC (disc.)
##   penalization coefficient:             1
##   tests used in the learning procedure: 189
##   optimized:                            TRUE
```

```
#vstructs returns a matrix according to value of arcs
#construct moral graph as well
vstructs(bn[[1]])
```

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

Results for Second HC run

```
arcs(bn[[2]])
```

```
##       from to
##  [1,] "L"  "E"
##  [2,] "E"  "X"
##  [3,] "S"  "B"
##  [4,] "T"  "E"
##  [5,] "T"  "B"
##  [6,] "A"  "T"
##  [7,] "L"  "B"
##  [8,] "A"  "S"
##  [9,] "B"  "D"
## [10,] "S"  "L"
## [11,] "E"  "D"
```

```
cpdag(bn[[2]])
```

```
##
##   Bayesian network learned via Score-based methods
##
##   model:
##     [partially directed graph]
##   nodes:                                8
##   arcs:                                 11
##     undirected arcs:                    3
##     directed arcs:                      8
##   average markov blanket size:          3.50
##   average neighbourhood size:           2.75
##   average branching factor:             1.00
##
##   learning algorithm:                   Hill-Climbing
##   score:                                AIC (disc.)
##   penalization coefficient:              1
##   tests used in the learning procedure:  175
##   optimized:                             TRUE
```

```
vstructs(bn[[2]])
```

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

```
#check equality
print("Comparing different runs of HC")
```

```
## [1] "Comparing different runs of HC"
```

```
print(all.equal(bn[[1]], bn[[2]]))
```

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

```
print(all.equal(bn[[2]], bn[[3]]))
```

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

```
print(all.equal(bn[[3]], bn[[4]]))
```

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

We had 4 runs of hc algorithm and they are not equal. The reason could be because the hc algorithm is a greedy search algorithm and stops when it reaches a local optimum. Also the HC algorithm starts at different locations at different runs and hence the maxima reached differs.

Question 2

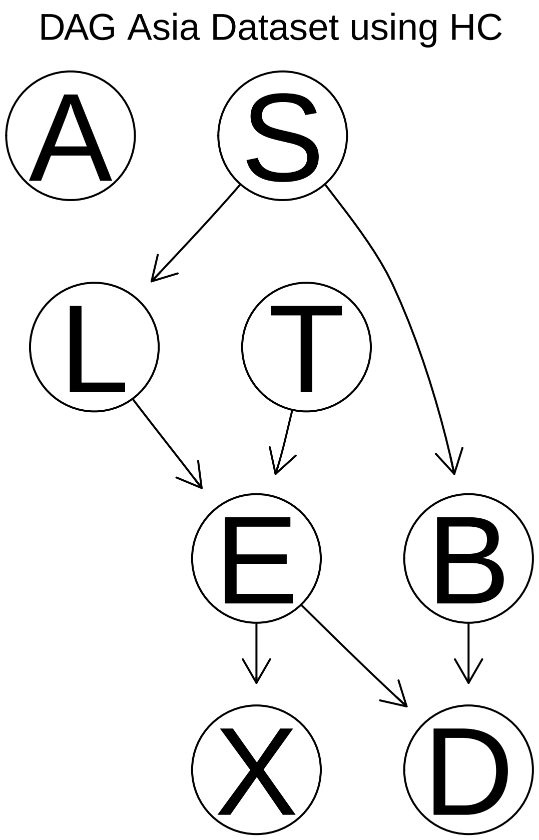
Learn a BN from 80% of the Asia dataset. The dataset is included in the bnlearn package. To load the data, run data(asia). Learn both the structure and the parameters. Use any learning algorithm and settings that you consider appropriate.Use the BN learned to classify the remaining 20 % of the Asia dataset 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]).

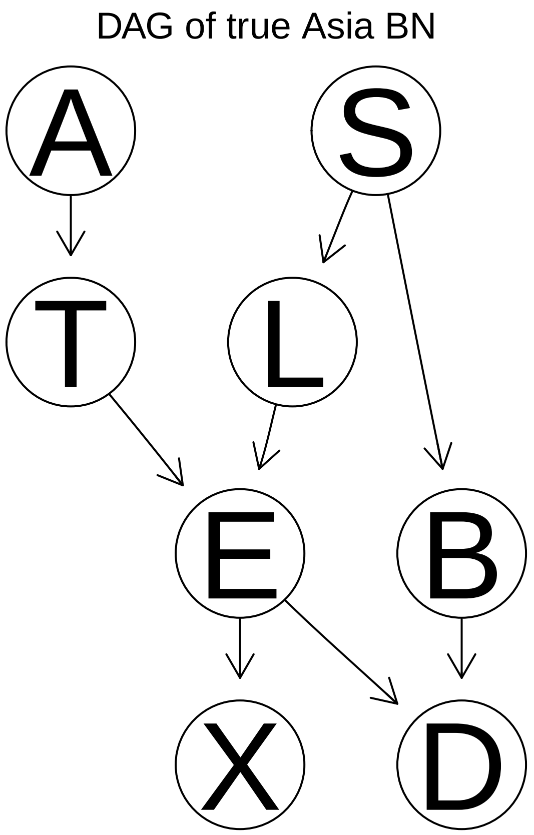
```
#Dividing data 80% and 20%
set.seed(12345)
n=dim(asia)[1]
id=sample(1:n, floor(n*0.8))
train=asia[id,]
test=asia[-id,]
```

```
# Learn Bayesian network
asia.dag = hc(train)
true.asia.dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

par(mfrow = c(1,2))
```

```r
graphviz.plot(asia.dag, main = "DAG Asia Dataset using HC")
graphviz.plot(true.asia.dag, main = "DAG of true Asia BN")
```





```r
inferance = function(dag , train, testdata ,nodes, prednode){

  fit = bn.fit(dag ,train)
  grainobj = as.grain(fit)
  # moralization and triangulation done by compile to satisfy RIP?
  junction = compile(grainobj)
  #plot(junction)
  prob = c()
  pred = c()

  #prediction
  for(i in 1:nrow(testdata)){
    s = as.numeric(testdata[i,])
    St = (s=ifelse(s==1,"no","yes"))

    jFinding= setEvidence(junction, nodes = nodes, states = St)
    prob[i] = querygrain(jFinding , nodes = prednode)

    pred[i] = ifelse(prob[[i]][[1]]>prob[[i]][[2]],"no","yes")
  }

  return(pred)
}

exact.pred = inferance(dag = asia.dag , train , testdata = test[,-2] ,
                       nodes = colnames(test[,-2]), prednode = "S" )
true.pred = inferance(dag = true.asia.dag , train ,testdata = test[,-2] ,
                      nodes = colnames(test[,-2]), prednode = "S" )
```

```
S = test[,2]
mtable = table(exact.pred, S)
mtable
```

```
##           S
## exact.pred  no yes
##        no  322 120
##        yes 146 412
```

```
mtab_true = table(true.pred, S)

accuracy = sum(diag(mtable))/sum(mtable)
accuracy
```

```
## [1] 0.734
```

```
accuracy_true = sum(diag(mtab_true))/sum(mtab_true)
accuracy_true
```

```
## [1] 0.734
```

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.

```
mb.asia.dag = mb(asia.dag , node = "S")
mb.true.asia.dag = mb(true.asia.dag , node = "S")

pred_mbblanket = inferance(asia.dag, train, testdata = test[,mb.asia.dag],
                           nodes = mb.asia.dag, prednode = "S" )
mtable = table(pred_mbblanket,S)
accuracy = sum(diag(mtable))/sum(mtable)
accuracy
```

```
## [1] 0.734
```

```
pred_true_mbblanket = inferance(true.asia.dag, train,
                                testdata = test[,mb.true.asia.dag],
                           nodes = mb.true.asia.dag, prednode = "S" )
mtable = table(pred_true_mbblanket,S)
accuracy = sum(diag(mtable))/sum(mtable)
accuracy
```

```
## [1] 0.734
```

Question 4

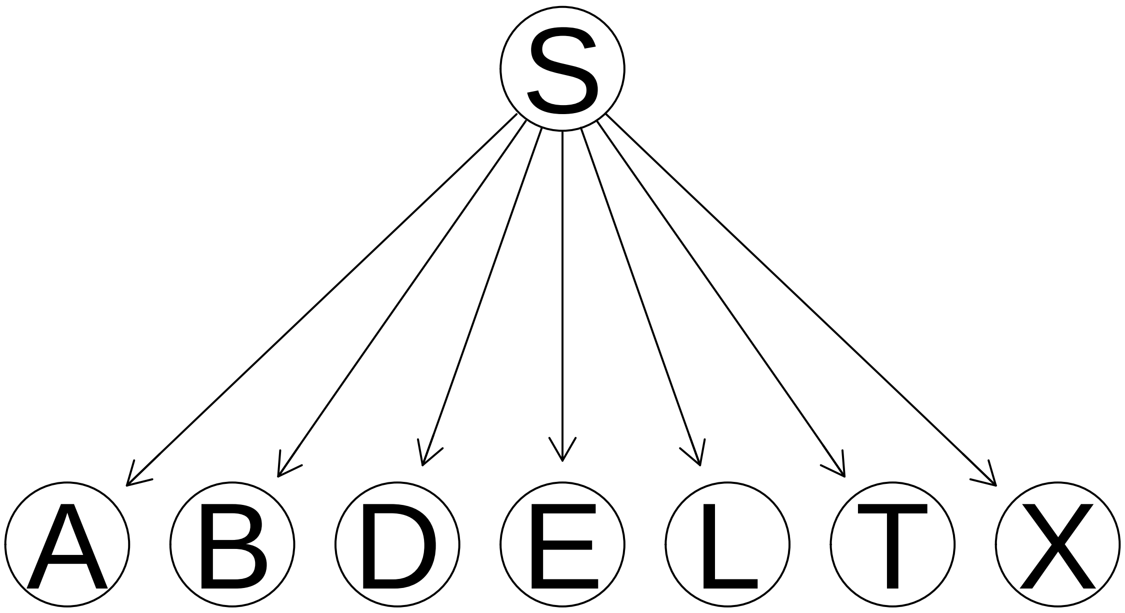
Repeat the exercise (2) using a naive Bayes classifier, i.e. the predictive variables are independent given the class variable. See p. 380 in Bishop's book or Wikipedia for more information on the naive Bayes classifier. Model the naive Bayes classifier as a BN. You have to create the BN by hand, i.e.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

```
# assumption : all nodes other than S are dependent on S and independent of each other
#S is the parent node for all other nodes

naivebayes.dag = model2network("[S][D|S][T|S][L|S][B|S][A|S][X|S][E|S]", debug = FALSE)

#nbcl= naive.bayes(train, training = "S")
#plot(nbcl)
graphviz.plot(naivebayes.dag , main = "Naive Bayes")
```

Naive Bayes



```
nb_pred = inferance(dag = naivebayes.dag , train , testdata = test[,-2] ,
                    nodes = colnames(test[,-2]), prednode = "S" )

S = test[,2]
mtable = table(nb_pred, S)
mtable
```

```
##        S
## nb_pred  no yes
##     no  349 188
##     yes 119 344
```

```
accuracy = sum(diag(mtable))/sum(mtable)
accuracy
```

```
## [1] 0.693
```

Question 5

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

Answer :

For any kind of inferance about the target node, we consider only the markov blanket. Markov blanket for a target node includes its parents, children and any nodes sharing a child. In excercise 2, the target node is S and If we check the markov blanket, it contains only nodes B and L.Hence we can say that given B and L , 'S' is independent of all other nodes. So in case of HC algorithm and true bayesian network, the dependancies of S are same. In excat inferance, a moral graph is created , triangulated, cliques are identified and then we do classification. In the above case the two DAG's will create same moral graph and hence the inferance give same results.

In excercise 3, we are extracting the markov blanket and the exact inferance algorithm is run considering only the markov blanket so even there we get same values of accuracy as in question 2.

In excercise 4 we use naive bayes algorithm where we consider, node S (n the classification node) as the parent of all other nodes and all other features as independent of each other. Here the markov blanket obtained is different from the above markov blanket and hence the network structure consideres S to be directly dependent on all other nodes. So this gives a different result