Advanced Machine Learning

Hugo Morvan (hugmo418)

2024-09-14

Lab 1: Graphical Models

Question 1

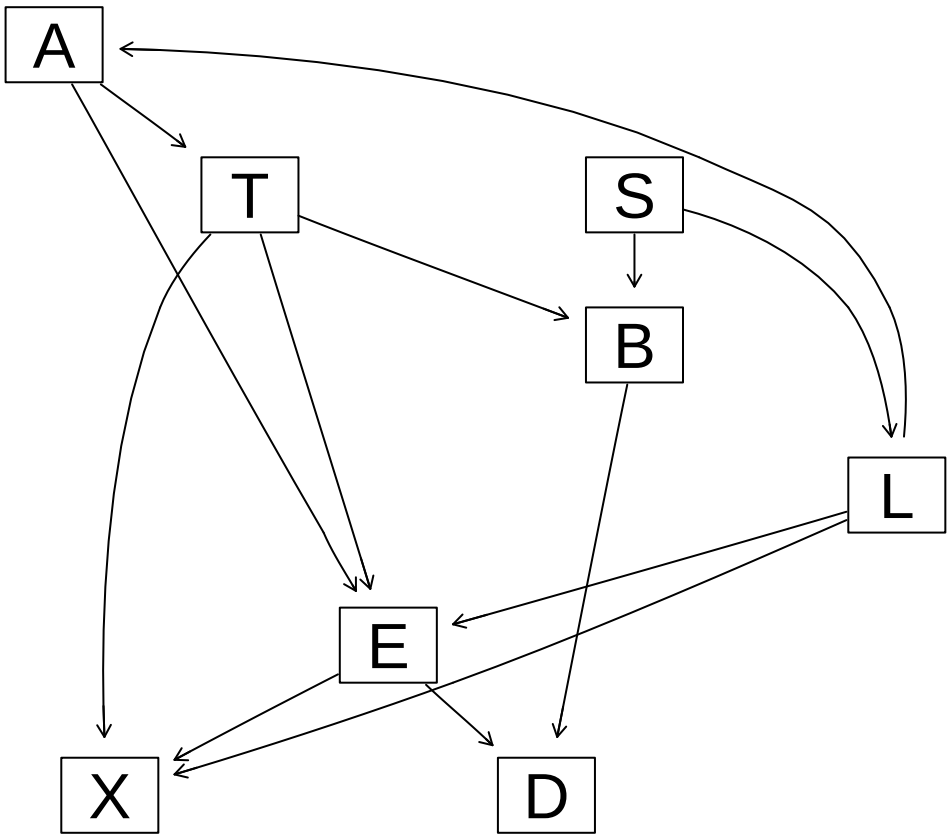
Show that multiple runs of the hill-climbing algorithm can return non-equivalent Bayesian network (BN) structures. Explain why this happens. Use the Asia dataset which is included in the bnlearn package. To load the data, run data("asia"). Recall from the lectures that the concept of non-equivalent BN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

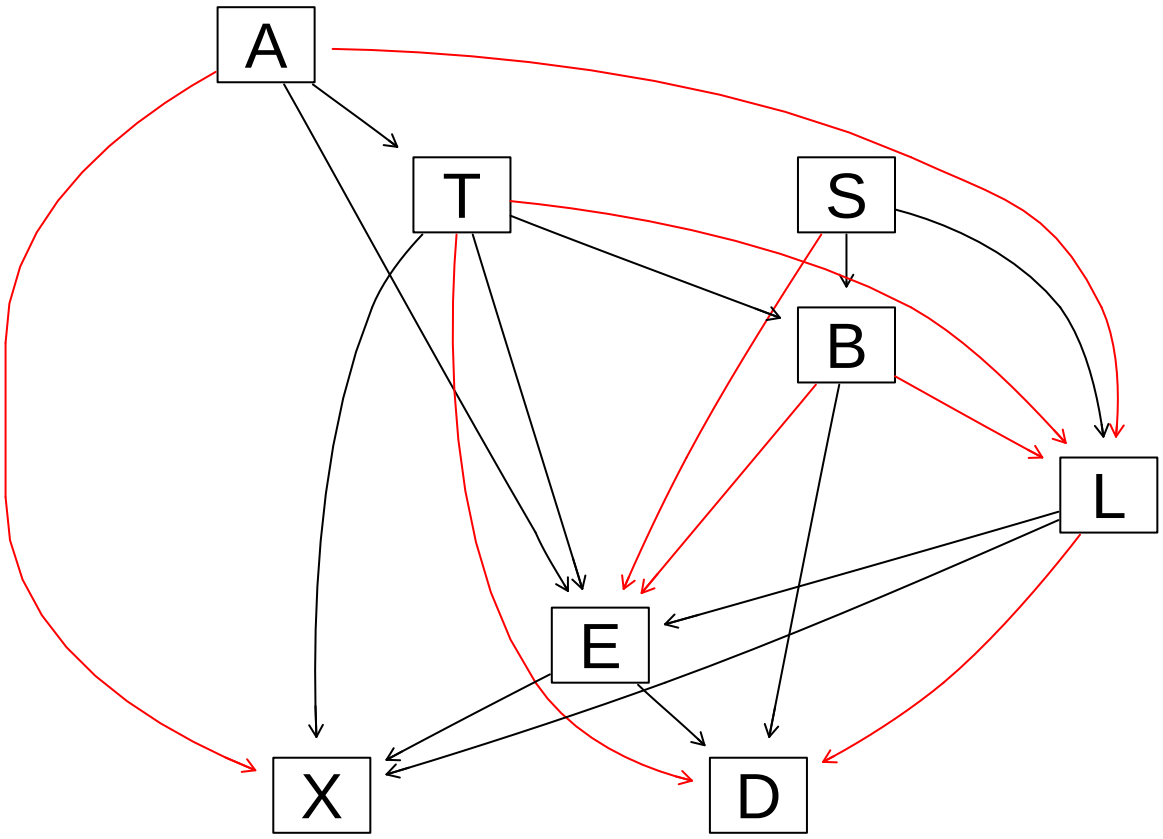
Hint: Check the function hc in the bnlearn package. Note that you can specify the initial structure, the number of random restarts, the score, and the equivalent sample size (a.k.a imaginary sample size) in the BDeu score. You may want to use these options to answer the question. You may also want to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

```
library(RBGL)
library(Rgraphviz)
library(gRain)
library(bnlearn)
library(caret)
```

```
#Dataset
data("asia")
```

```
set.seed(123)

#Learning a first graph with Hill Climb with imaginary sample size = 10
graph_a = hc(asia, restart = 10, score = "bde", iss = 10)
#Learning a second graph with Hill Climb with imaginary sample size = 100
graph_b = hc(asia, restart = 10, score = "bde", iss = 100)

#Plotting the graphs
par(mar=c(0.5,0.5,0.5,0.5))
graphviz.compare(graph_a, graph_b)
```


A
T
S
B
L
E
X
D




```
#Checking for equivalence
print(all.equal(graph_a, graph_b))
```

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

```
#Comparing the BDeu scores
score_a = score(graph_a, asia, type = "bde")
cat("Graph A BDeu score:",score_a,"\n")
```

```
## Graph A BDeu score: -11132.15
```

```
score_b = score(graph_b, asia, type = "bde")
cat("Graph B BDeu score:",score_b,"\n")
```

```
## Graph B BDeu score: -11467.13
```

By changing the parameter iss (imaginary sample size), we get graphs that are not equivalent. The BDeu score is also different for the two graphs.

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]").

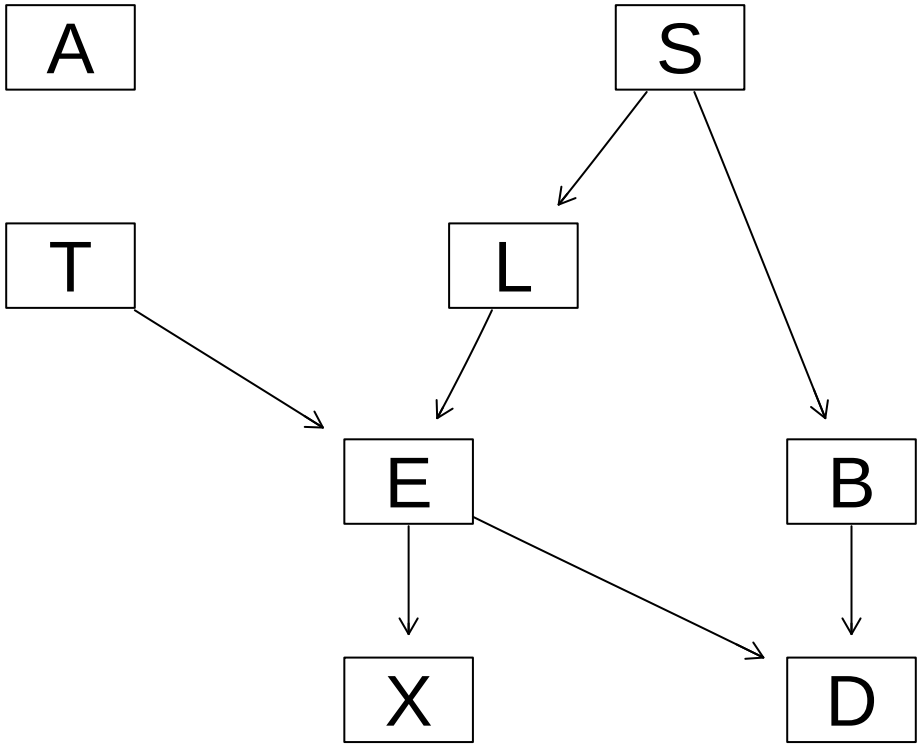
Hint: You already know two algorithms for exact inference in BNs: Variable elimination and cluster trees. There are also approximate algorithms for when the exact ones are too demanding computationally. For exact inference, you may need the functions bn.fit and as.grain from the bnlearn package, and the functions compile, setEvidence and querygrain from the package gRain. For approximate inference, you may need the functions cpquery or cpdist, prop.table and table from the bnlearn package.

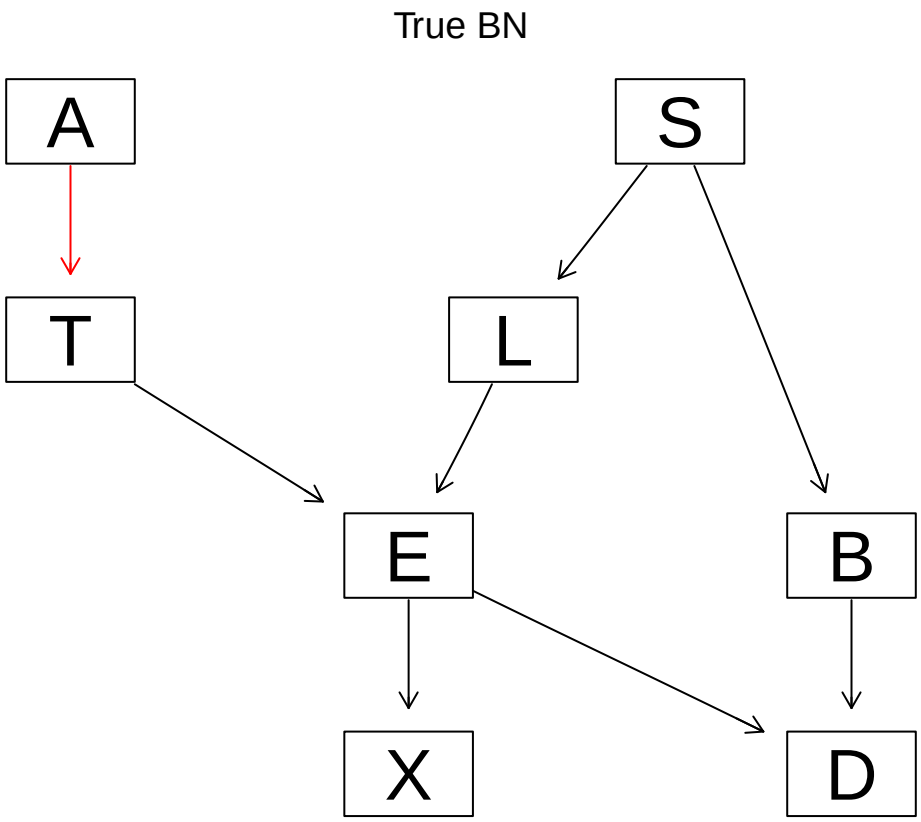
```r
set.seed(123)
#Splitting the data set between Train and Test datasets
n = nrow(asia)
train = asia[1:round(0.8*n),]
test = asia[(round(0.8*n)+1):n,]

#Learn the Structure using default Hill Climb
g = hc(train)
# 8 nodes, only 4 directed arcs, no undirected arcs.
par(mar=c(0,0,0,0))

#Learn the parameters
fitted = bn.fit(g, train, method = "mle")
#convert bn.fit object to grain object:
asiabn = as.grain(fitted)
asiabn = compile(asiabn)
summary(asiabn)
```

```
## Independence network: Compiled: TRUE Propagated: FALSE
##  Nodes : Named chr [1:8] "A" "S" "T" "L" "B" "E" "X" "D"
##  - attr(*, "names")= chr [1:8] "A" "S" "T" "L" ...
##  Number of cliques:                6
##  Maximal clique size:              3
##  Maximal state space in cliques:   8
```

```r
#querygrain : obtain the conditional distribution of a set of variables - possibly (and typically) give

#True dag
dag_true = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
graphviz.compare(fitted, dag_true, main=c("Learned BN", "True BN"))
```

Learned BN






```
true_bn = bn.fit(dag_true, train, method="mle")
true_bn = as.grain(true_bn)
true_bn = compile(true_bn)

#Classify the remaining 20% of the Asia dataset
preds = rep(NaN, dim(test)[1])
preds_true = rep(NaN, dim(test)[1])

for(i in 1:dim(test)[1]){
  #For the learned structure:
  #Set the evidences to the i-th row in the data
  nodes = c("A","T","L","B","D","E","X")
  states = as.vector(unlist(c(test[i,nodes])))
  evidences = setEvidence(asiabn, nodes, states)
  #Probability, given the evidences
  prob = querygrain(evidences, nodes = "S")
  #Predict using maximum probability label
  preds[i] = names(which.max(c(prob$S[1],prob$S[2])))

  #For the true dag:
  evidences2 = setEvidence(true_bn, nodes, states)
  prob2 = querygrain(evidences2, nodes = "S")
  preds_true[i] = names(which.max(c(prob2$S[1],prob2$S[2])))
}
#Report the confusion matrix for learned structure
print("--- Learned Structure results: ---")
```

```
## [1] "--- Learned Structure results: ---"
```

```
preds = as.factor(preds)
confmat = confusionMatrix(preds, test$S)
print(confmat$table)
```

```
##           Reference
## Prediction  no yes
##        no  358 120
##        yes 147 375
```

```
print(confmat$overall[1])
```

```
## Accuracy
##    0.733
```

```
#Compare your results with those of the true Asia BN
print("--- True Structure results: ---")
```

```
## [1] "--- True Structure results: ---"
```

```
preds_true = as.factor(preds_true)
confmat2 = confusionMatrix(preds_true, test$S)
print(confmat2$table)
```

```
##           Reference
## Prediction  no yes
##        no  358 120
##        yes 147 375
```

```
print(confmat2$overall[1])
```

```
## Accuracy
##    0.733
```

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.

```
#Learn the parameters
fitted = bn.fit(g, train, method = "mle")
#Extract the Markov blanket
Smb = mb(fitted, "S")
print(Smb)
```

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

```
#convert bn.fit object to grain object:
mbbn = as.grain(fitted)
mbbn = compile(mbbn)
summary(mbbn)
```

```
## Independence network: Compiled: TRUE Propagated: FALSE
##  Nodes : Named chr [1:8] "A" "S" "T" "L" "B" "E" "X" "D"
##  - attr(*, "names")= chr [1:8] "A" "S" "T" "L" ...
##  Number of cliques:                 6
##  Maximal clique size:               3
##  Maximal state space in cliques:    8
```

```
#querygrain : obtain the conditional distribution of a set of variables - possibly (and typically) give

#Classify the remaining 20% of the Asia dataset
preds = rep(NaN, dim(test)[1])

for(i in 1:dim(test)[1]){
  nodes = Smb #nodes of interest are the ones in my S-Markov blanket
  states = as.vector(unlist(c(test[i,Smb])))
  evidences = setEvidence(mbbn, nodes, states)
  prob = querygrain(evidences, nodes = "S")
  preds[i] = names(which.max(c(prob$S[1],prob$S[2])))
}
#Report the confusion matrix
preds = as.factor(preds)
confmat = confusionMatrix(preds, test$S)
print(confmat$table)
```

```
##          Reference
## Prediction  no yes
##        no  358 120
##        yes 147 375
```

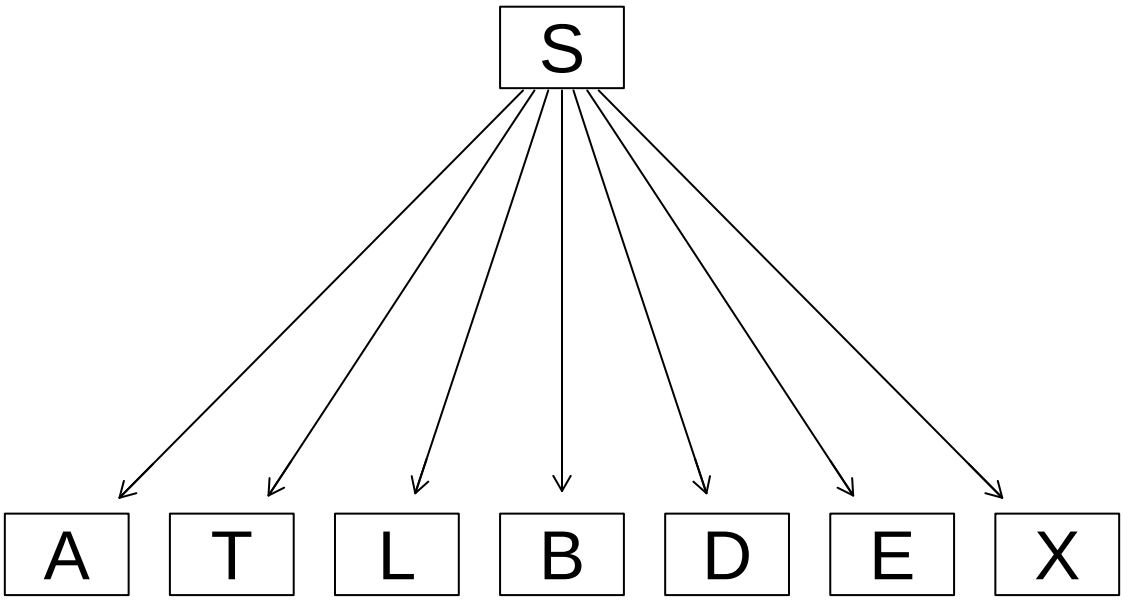
```
print(confmat$overall[1])
```

```
## Accuracy
##    0.733
```

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
#Creating and empty graph containing all the variables
naiveTree = empty.graph(c("S","A","T","L","B","D","E","X"))
#Manually setting the edges to represent a Naive Bayes Classifier
edges = matrix(c("S","A","S","L","S","T","S","B","S","E","S","X","S","D"),
               ncol = 2, byrow = T, dimnames=list(NULL, c("from","to")))
arcs(naiveTree) = edges
graphviz.plot(naiveTree, main = "Naive Bayes tree")
```




```r
fittedBayes = bn.fit(naiveTree, train, method = "mle")
#convert bn.fit object to grain object:
bayesBN = as.grain(fittedBayes)
bayesBN = compile(bayesBN)
summary(bayesBN)
```

```
## Independence network: Compiled: TRUE Propagated: FALSE
##  Nodes : Named chr [1:8] "S" "A" "T" "L" "B" "D" "E" "X"
##  - attr(*, "names")= chr [1:8] "S" "A" "T" "L" ...
##  Number of cliques:                 7
##  Maximal clique size:               2
##  Maximal state space in cliques:    4
```

```r
#querygrain : obtain the conditional distribution of a set of variables - possibly (and typically) gives

#Classify the remaining 20% of the Asia dataset
```

```
preds = rep(NaN, dim(test)[1])

for(i in 1:dim(test)[1]){
  nodes = c("A","T","L","B","D","E","X")
  states = as.vector(unlist(c(test[i,nodes])))
  evidences = setEvidence(bayesBN, nodes, states)
  prob = querygrain(evidences, nodes = "S")
  preds[i] = names(which.max(c(prob$S[1],prob$S[2])))
}
#Report the confusion matrix
preds = as.factor(preds)
confmat = confusionMatrix(preds, test$S)
print(confmat$table)
```

```
##           Reference
## Prediction  no yes
##        no  389 180
##        yes 116 315
```

```
print(confmat$overall[1])
```

```
## Accuracy
##    0.704
```

5

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

The results between question 2 and 3 are the same because the Markov blanket does not changes the results of the prediction of S since it keeps all the useful features (hence removing the features that already did not influence the result).

The results in question 4 differ from the previous 2 because a naive Bayes structure assumes conditional independence between the features given S, which is not the case in the true DAG. Therefore, due to that assumption, the results are slightly lesser.