732A96 Advanced Machine Learning

LAB 1: GRAPHICAL MODELS

Johannes Hedström

STIMA
Department of Computer and Information Science
Linköpings universitet

2024-10-24

Contents

```r
# packages
#if (!requireNamespace("BiocManager", quietly = TRUE))
#install.packages("BiocManager")
#BiocManager::install("RBGL")
#BiocManager::install("Rgraphviz")
#BiocManager::install("gRain")

library(bnlearn)
library(gRain)
library(knitr)
```

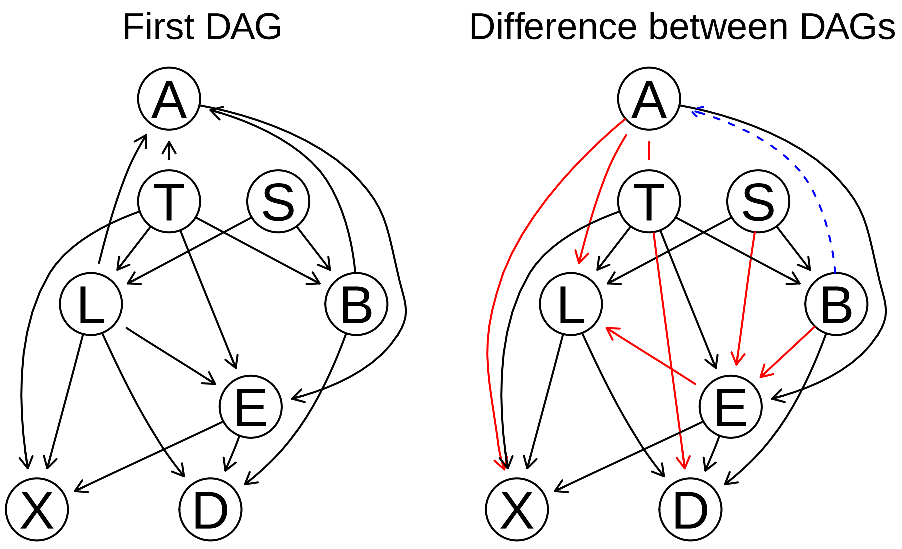
1 Question 1

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.

```r
# loading data
data("asia")

print(head(asia,5)) # table of the data to get a view of it
```

```
##    A   S   T  L   B   E   X   D
## 1 no yes  no no yes  no  no yes
## 2 no yes  no no  no  no  no  no
## 3 no  no yes no  no yes yes yes
## 4 no  no  no no yes  no  no yes
## 5 no  no  no no  no  no  no yes
```

```r
# hill climbing with score = bde for Baysian drichlet equivalent(uniform)
# iss for imaginary sample size
# 10 random restarts
set.seed(12345) # seed to get same results

# 2 different runs
r1 <- hc(asia,restart = 10 ,score='bde', iss=20)
r2 <- hc(asia,restart = 10 ,score='bde', iss=50)

dag1 <- cpdag(r1) # finding class and v structure
dag2 <- cpdag(r2)

# plotting grpahs
par(mfrow= c(1,2))
graphviz.compare(dag1,dag2, shape='circle',main = c('First DAG','Difference between DAGs'))
```



```r
arc1 <- arcs(dag1) # extracting arcs
arc2 <- arcs(dag2)

# checking if they are equal
all.equal(dag1,dag2)
```

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

The bayesian networks are considered non equivalent of they represent different sets of conditional independencies.

The algorithm starts with an empty DAG, then tries to add, remove or reverse any edge th G that improves the BDEu score the most!

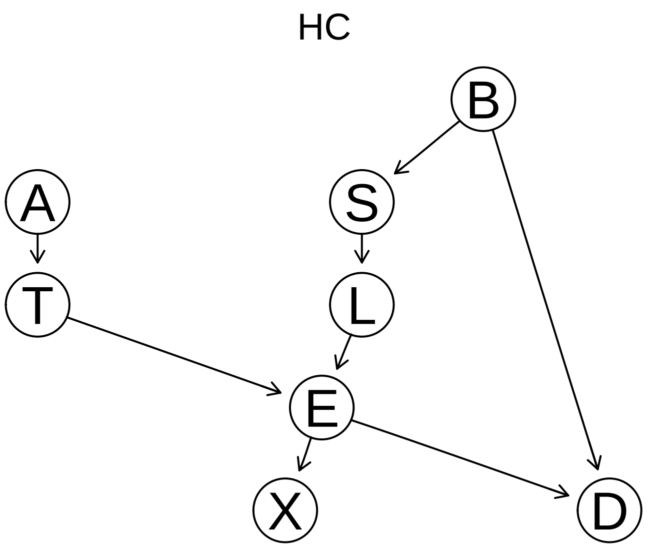
Hill climbing algorithm is a greedy search that relies on the initial conditions, which means that different parameters can likely lead to different local optimas. And that's why the algorithm produces non identical structures. A higher iss means more parameters which then means a more complex model, so a lower iss is then more regularization on the model!

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

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.

```
set.seed(12345) # seed for same results when knitted
idx <- sample(1:5000,5000*0.8) # picking out my training data
train <- asia[idx,]
test <- asia[-idx,]

# hill climbing alg, 100 restarts, doesnt need same faithful assumptions as other algorithms
struct <- hc(train,restart = 100 ,score='bde')

graphviz.plot(struct, main='HC', shape='circle')
```



```
fit_1 <- bn.fit(struct, train, method='mle') # fitting the parameters with MLE
                                            # change to bayes if quastion ask for bayes params values
preds <- c()# empty vector
fit_g <- as.grain(fit_1)
fit_g <- compile(fit_g)

# looping over all the rows in the test data

for(r in 1:nrow(test)){
  # all nodes except 'S' and setting the current row to the state
  ev <- setEvidence(fit_g , nodes = colnames(test[-2]), states=as.vector(unlist(test[r,-2])))

  # marginal prob for S[r]
  qg <- querygrain(ev,node='S', type='marginal')

  # predicting S
  preds[r] <- ifelse(qg$S[1] > qg$S[2],'No','Yes')
}


cf <- table(preds,test$S)/length(preds)
knitr::kable(cf, caption='Confusion matrix')
```

Table 1: Confusion matrix

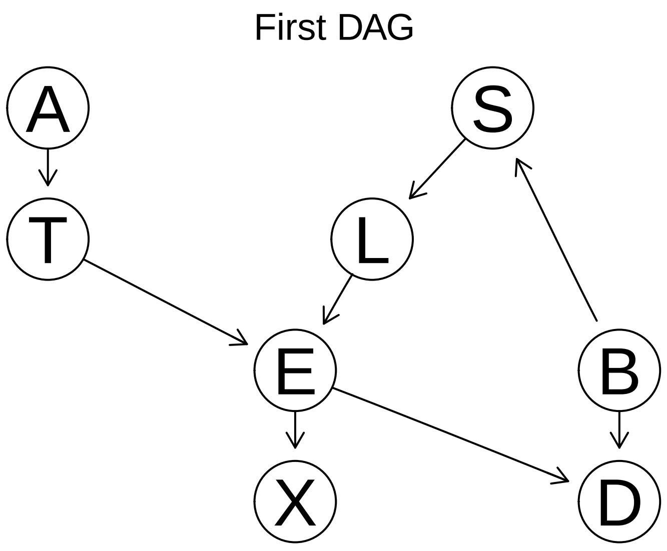
	no	yes
No	0.337	0.121
Yes	0.176	0.366

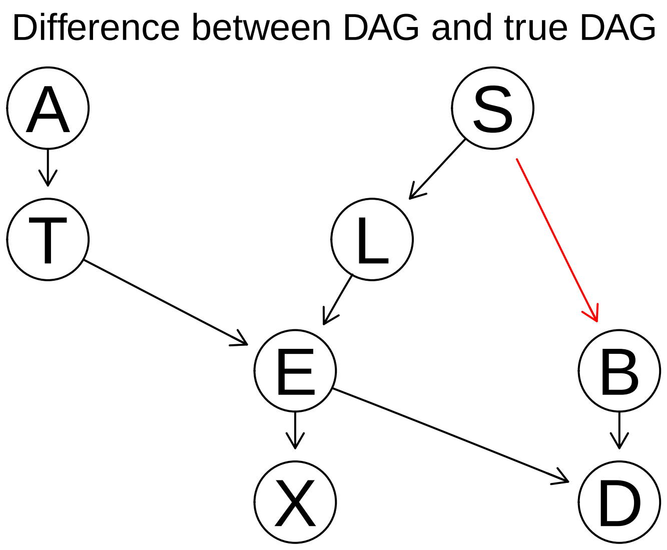
```
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]") # true dag

fit_1 <- bn.fit(dag, train, method='mle') # fitting the parameters with MLE

preds <- c()# empty vector
fit_g <- as.grain(fit_1)
fit_g <- compile(fit_g)

# looping over all the rows in the test data

for(r in 1:nrow(test)){
  # all nodes except 'S' and setting the current row to the state
  ev <- setEvidence(fit_g , nodes = colnames(test[-2]), states=as.vector(unlist(test[r,-2])))

  # marginal prob for S[r]
  qg <- querygrain(ev,node='S', type='marginal') # posterior distribution (remove type = marginal for th

  # predicting S
  preds[r] <- ifelse(qg$S[1] > qg$S[2],'No','Yes')
}


tb<- table(preds,test$S)/length(preds)
knitr::kable(tb, caption='Confusion matrix')
```

Table 2: Confusion matrix

	no	yes
No	0.337	0.121
Yes	0.176	0.366

```
graphviz.compare(struct,dag, shape='circle',main = c('First DAG','Difference between DAG and true DAG'))
```





The Dag I found with my hc algorithm is equivalent in probability with the true Dag and that's why i get the same prediction for both dags.

The conditional probabilities estimated are with mle.

3 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. Hint: You may want to use the function mb from the bnlearn package.

```r
fit_1 <- bn.fit(struct, train, method='mle') # fitting the parameters with MLE

preds <- c()# empty vector
fit_g <- as.grain(fit_1)
fit_g <- compile(fit_g)

# looping over all the rows in the test data
r=1
for(r in 1:nrow(test)){
  #  Markov blanket to get the nodes that are parents plus children plus parent of children
  mb <- mb(fit_1, node='S')
  # using the mb nodes as evidence
  ev <- setEvidence(fit_g , nodes = mb, states=as.vector(unlist(test[r,mb])))

  # marginal prob for S[r]
  qg <- querygrain(ev,node='S', type='marginal')

  # predicting S
  preds[r] <- ifelse(qg$S[1] > qg$S[2],'No','Yes')
}

tb<- table(preds,test$S)/length(preds)
knitr::kable(tb, caption='Confusion matrix')
```

Table 3: Confusion matrix

	no	yes
No	0.337	0.121
Yes	0.176	0.366

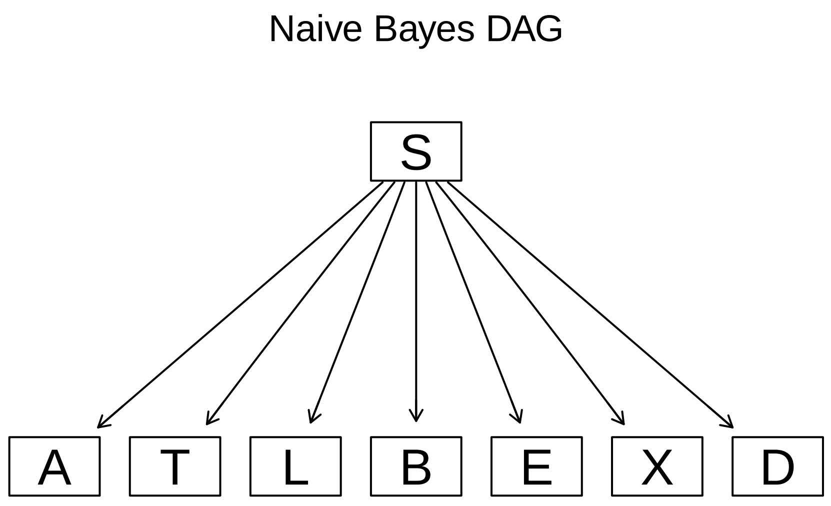
I still get the same results as before even though we are using less information to classify S, This is because given that you know the values of the Markov blanket, S is independent of all other informations.

4 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. Hint: Check http://www.bnlearn.com/examples/dag/ to see how to create a BN by hand.

```r
empt_dag <- empty.graph(colnames(asia)) # creating empty graph

# picking out the letters except S
let <- colnames(train[,-2])

# creating a matrix with 'from' s to 'letters'
```

```r
nb_arcs <- as.matrix(data.frame('from'=rep('S',length(let)), 'to'=let))

# creating arcs from s to...
arcs(empt_dag) <- nb_arcs

# plot
graphviz.plot(empt_dag, main='Naive Bayes DAG')
```



Naive Bayes assumes that all values as independent of the other features.

So for example A is independent of all other variables given that you know S, and this is for every other variable.

For this to work for 1 variable we could have the constellation "chain" (A->S->T) or 'fork' (A<-S->T) and the latter works for every variable if S is the parent to all other nodes.

```r
fit_1 <- bn.fit(empt_dag, train, method='mle') # fitting the parameters with MLE

preds <- c()# empty vector
fit_g <- as.grain(fit_1)
fit_g <- compile(fit_g)

# looping over all the rows in the test data
r=1
for(r in 1:nrow(test)){

  # using the mb nodes as evidence
  ev <- setEvidence(fit_g , nodes = colnames(test[,-2]), states=as.vector(unlist(test[r,-2])))

  # marginal prob for S[r]
  qg <- querygrain(ev,node='S', type='marginal')

  # predicting S
  preds[r] <- ifelse(qg$S[1] > qg$S[2],'No','Yes')
}


tb <- table(preds,test$S)/length(preds)
knitr::kable(tb, caption='Confusion matrix')
```

Table 4: Confusion matrix

	no	yes
No	0.359	0.180
Yes	0.154	0.307

```
# accuracy
cat('Naive bayes validation accuracy:',sum(diag(table(preds,test$S)/length(preds))),'\n',
'Validation accuracy from task 2:',sum(diag(cf)))
```

```
## Naive bayes validation accuracy: 0.666
##  Validation accuracy from task 2: 0.703
```

Worse classification results for Naive Bayes than with HC algorithm.

```
# another way of setting up the graph
library(bnlearn)
#library(gRain)

#net <- model2network("[U][C|U][A|C][B|C][D|A:B][Ch|U][Ah|Ch][Bh|Ch][Dh|Ah:Bh]")
#plot(net)

# custom prob
#cptC <- matrix(c(.9,.1,.1,.9), nrow=2, ncol=2)
#dim(cptC) <- c(2,2)
#dimnames(cptC) <- list("C" = c("0", "1"), "U" =  c("0", "1"))

#netfit <- custom.fit(net,list(U=cptU, C=cptC, A=cptA, B=cptB, D=cptD, Ch=cptCh, Ah=cptAh, Bh=cptBh, Dh=
#netcom <- compile(as.grain(netfit))

#querygrain(setEvidence(netcom,nodes=c("D","Ah"),states=c("1","0")),c("Dh"))
```

5 Question 5

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

Two Dags represent the same indipendencies according to the seperation criterion if and only if they have the sa,e adjancies and unshielded colliders. We hat that Dag1 and TRue DAG have the same adjancies and unshielded colliders, therefore they are eqvavilent.

Markov blanket tell us that if we have evidence for T and B, then all other evidence for the other variables is not neded to infer S. Tahts why we get the same results in 2 and 3.

I get the same results in task 2 and 3 as the results of B and T are directly dependent on S, so when using all data in task 2 the variables that affect the probability for S are B and T. The 2 variables i get from using Markov Blanket are B and T, so when using only those variables for the prediction its the same data that depends on S that will be used and therefore the results are the same.

When it comes to task 4 I get different results as the graph results in that all other variable depends on S, not only B an T, and that all other variables are independent of each other given S. Which is a bold/risky assumption unless you have expert knowledge about the domain. And as task 2 gave us the True graph we know that this assumption doesn't correspond to reality.