Advanced Machine Learning Lab 1

Mikael Montén

2024-10-22

```
library("RBGL")
library("Rgraphviz")
library("gRain")
library("bnlearn")
library("tidyverse")
data(asia)
asia
```

Dataset used for all labs consists of 5000 observations and the following structure.

Table 1: asia dataset

A	S	T	L	B	E	X	D
no	yes	no	no	yes	no	no	yes
no	yes	no	no	no	no	no	no
no	no	yes	no	no	yes	yes	yes
no	no	no	no	yes	no	no	yes
no	no	no	no	no	no	no	yes

Question 1

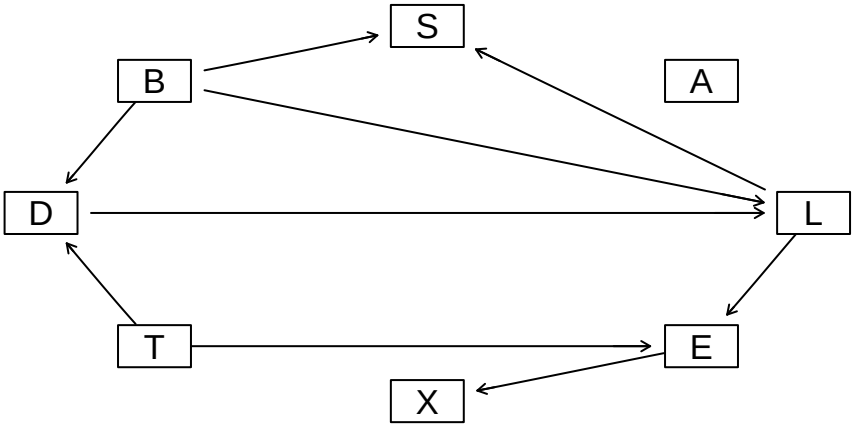
Show that multiple runs of the hill-climbing algorithm can return non-equivalent Bayesian network (BN) structures. Explain why this happens. Recall from the lectures that the concept of non-equivalent BN structures has a precise meaning.

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.

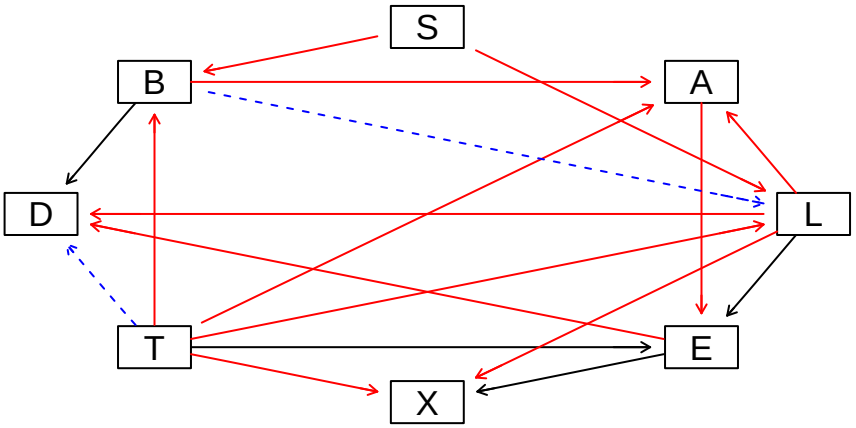
```
library("RBGL")
library("Rgraphviz")
library("gRain")
library("bnlearn")
library("tidyverse")
# different initial structures
# true one according to documentation of asia dataset
d1 = model2network("[A][S][T|A:S][L|T][B|L][D|B][E|D][X|E]")
d2 = model2network("[B][D|B][A|D][S|A:B][T|S:D][L|T:A][E|L:B][X|E:S]")

set.seed(13)
n1 = hc(x=asia,start=d2,score="bic",restart=10)
```

```
n2 = hc(x=asia,score="bde",restart=25,iss=20)
n3 = hc(x=asia,start=d1,score="aic",restart=30)
```

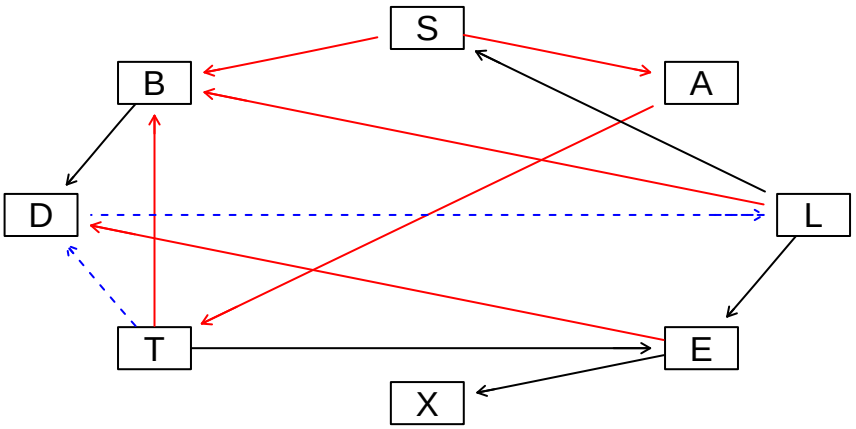
BIC, 10 restarts, unequivalent initial structure



BDE, 25 restarts, 20 imaginary sample size



AIC, 30 restarts, 'true' initial structure



The three graphs created all have different initializations to them. First one is evaluated using BIC with 10 restarts and an initial structure that is unequivalent to the true on according to documentation for the

dataset. The second one is evaluated with BDE without an initial structure given, 25 restarts and an imaginary sample size of 20. The third one is evaluated using AIC, 30 restarts and using the true structure from the dataset documentation. The probability of finding the global optimum is increased by i.e. increasing the number of random restarts. Altering the score also potentially affects the performance of the algorithm. In the 2nd graph the use of iss (equivalent sample size in BDeu) is used, which is a parameter that controls the strength of the prior distribution over the BN parameters. A higher equivalent sample size will result in a stronger prior, which can help to prevent overfitting.

Simply looking at these graphs it's easy to see they are three non-equivalent BNs. The precise definition of a non-equivalent BN structure is, a network that represents a set of conditional independences among variables that can't be represented by any other DAG structure. This means two non-equivalent BN structures encode different sets of conditional independence assumptions.

Two BN structures are equivalent if and only if they have the same skeleton and the same v-structures. The skeleton of a BN is the undirected graph that results from removing the directions of the edges in the BN. To show the non-equivalent structure formally we use the following functions,

- The V-structs denotes if the differents arc are Markov Equivalent or not. Markov Equivalence means two nodes has a common child node without an edge between them.

```
knitr::kable(vstructs(n1))
```

X	Z	Y
T	E	L
T	D	B

```
knitr::kable(vstructs(n2))
```

X	Z	Y
L	A	B
S	L	T
S	B	T
L	D	B
B	D	E

```
knitr::kable(vstructs(n3))
```

X	Z	Y
S	B	T
T	B	L
T	E	L
B	D	E

Since these all contain unique v-structs for each graph, it further shows that the structures are non-equivalent. Furthermore, as the graph shows, there a different number of arcs between each other.

```
all.equal(n1,n3)
```

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

```
all.equal(n1,n2)
```

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

```r
all.equal(n2,n3)
```

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

The hill climbing algorithm is a local search algorithm that starts with an initial candidate solution and iteratively improves it by moving to neighboring candidate solutions that have better scores. This is done by trying to add, remove, or reverse any edge in G and see what yields an improvement in score. Since it is a local search algorithm, it may get stuck in a local optimum and fail to find the global optimum, due to the large search space of BN structures. This means that multiple runs of the algorithm can return different BN structures, even if they are initialized with the same parameters.

Question 2

Learn a BN from 80 % of the Asia dataset. The dataset is included in the bnlearn package. 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.

Split data into 80/20 train and validation split

```
# 80% split
index <- sample(1:nrow(asia), 0.8*nrow(asia))
train = asia[index,]
test = asia[-index,]
```

Using Hill Climbing algorithm again, evaluated with AIC and 10 restarts.

```
set.seed(13)
# find skeleton of DAG with hc algorithm
dag_hc = hc(x=train,score="aic",restart=10)
# fit hc dag on train data using MLE, change class for gRain inference and compile
fitted = compile(as.grain(bn.fit(dag_hc,train)))
```

Computing posterior probability distribution for S. Requires the variable to not be evaluated as evidence.

```
set.seed(13)
ev = test[colnames(test) != "S"] # evidence test data
pred_hc = matrix(0,nrow=nrow(ev),ncol=1)

for(i in 1:nrow(ev)){
  # set evidence as value for each variable row-wise, except for S
  evidence = setEvidence(fitted,nodes=c(colnames(ev)),states=as.vector(unlist(ev[i,])))
  # predict by obtaining conditional distribution for each row-wise set of values and classifying
  pred_hc[i] = ifelse((querygrain(evidence, nodes = "S")$S["yes"]>=0.5), "yes","no")
}

conf_hc = table(pred_hc,test$S)
```

Redo the previous steps for the true distribution [A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]

```
set.seed(13)
# true conditional structure for the dataset
dag_t = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
true = compile(as.grain(bn.fit(dag_t,train)))
truth = matrix(0,nrow=nrow(ev),ncol=1)
for(i in 1:nrow(ev)){
  evidence = setEvidence(true,nodes=c(colnames(ev)),states=as.vector(unlist(ev[i,])))
```

```
  truth[i] = ifelse((querygrain(evidence, nodes = "S")$S["yes"]>=0.5), "yes","no")
}

conf_t = table(truth,test$S)
```

Table 5: Confusion matrix Hill-Climbing

	no	yes
no	352	108
yes	154	386

Table 6: Confusion matrix True structure

	no	yes
no	350	106
yes	156	388

The two confusion matrices generated when comparing the predictions from the computed and the true conditional dependencies are very similar. The accuracies are similar between the estimated models as it's only 2 observations differing per class.

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.

We use the same fitted model as in the exercise before, the only difference is that we only use the Markov Blanket variables for predictions.

```
set.seed(13)
blanket = mb(bn.fit(dag_hc,train),"S")
blanket # markov blanket variables are T, L and B
```

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

```
pred_mb = matrix(0,nrow=nrow(ev),ncol=1)
for(i in 1:nrow(ev)){
  evidence = setEvidence(fitted,nodes=c(blanket),states=as.vector(unlist(ev[i,blanket])))
  pred_mb[i] = ifelse((querygrain(evidence, nodes = "S")$S["yes"]>=0.5), "yes","no")
}

conf_mb = table(pred_mb,test$S)
```

The Markov blanket nodes are T, L and B. The Markov blanket of a variable is the set of nodes that are directly connected to it in the BN. It includes the variables parents, its children, and the parents of its children. The Markov blanket shields a variable from the rest of the network, meaning that the variable is conditionally independent of all other variables in the network given its Markov blanket.

Table 7: Confusion matrix Markov Blanket

	no	yes
no	352	108
yes	154	386

The confusion matrix computed using the Markov Blanket of S compared to the Hill-Climb generated structure from last exercise results in the identical confusion matrix. This is possible becuase the Markov blanket of S contains all the information that is needed to predict the value of S.

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.

Do the same steps as before but with an initialization that aligns with the variables being independent given the class variable

```
# if data had no underlying structure and probabilities had to be hardcoded
# all created variables requires the dependencies to be encoded as new dimensions
dag_nb = model2network("[S][X|S][D|S][B|S][E|S][T|S][L|S][A|S]")

fitted = compile(as.grain(bn.fit(dag_nb,train)))
pred_nb = matrix(0,nrow=nrow(ev),ncol=1)

for(i in 1:nrow(ev)){
  evidence = setEvidence(fitted,nodes=c(colnames(ev)),states=as.vector(unlist(ev[i,])))
  pred_nb[i] = ifelse((querygrain(evidence, nodes = "S")$S["yes"]>=0.5), "yes","no")
}

conf_nb = table(pred_nb,test$S)
```

Table 8: Confusion matrix Naive Bayes

	no	yes
no	380	175
yes	126	319

The confusion matrix from Naive Bayes structure provides a worse classification than the previous models. This is due to the Naive Bayes assumption that the perdictive variables are conditionally independent of each other given the class variable, which results in a simple structure where the class variable is the parent of all predictive variables, and there are no other edges in the network. This assumption is often violated in practice (i.e. in this example), but Naive Bayes can still perform well in many cases, especially when the number of predictive variables is large.

Question 5

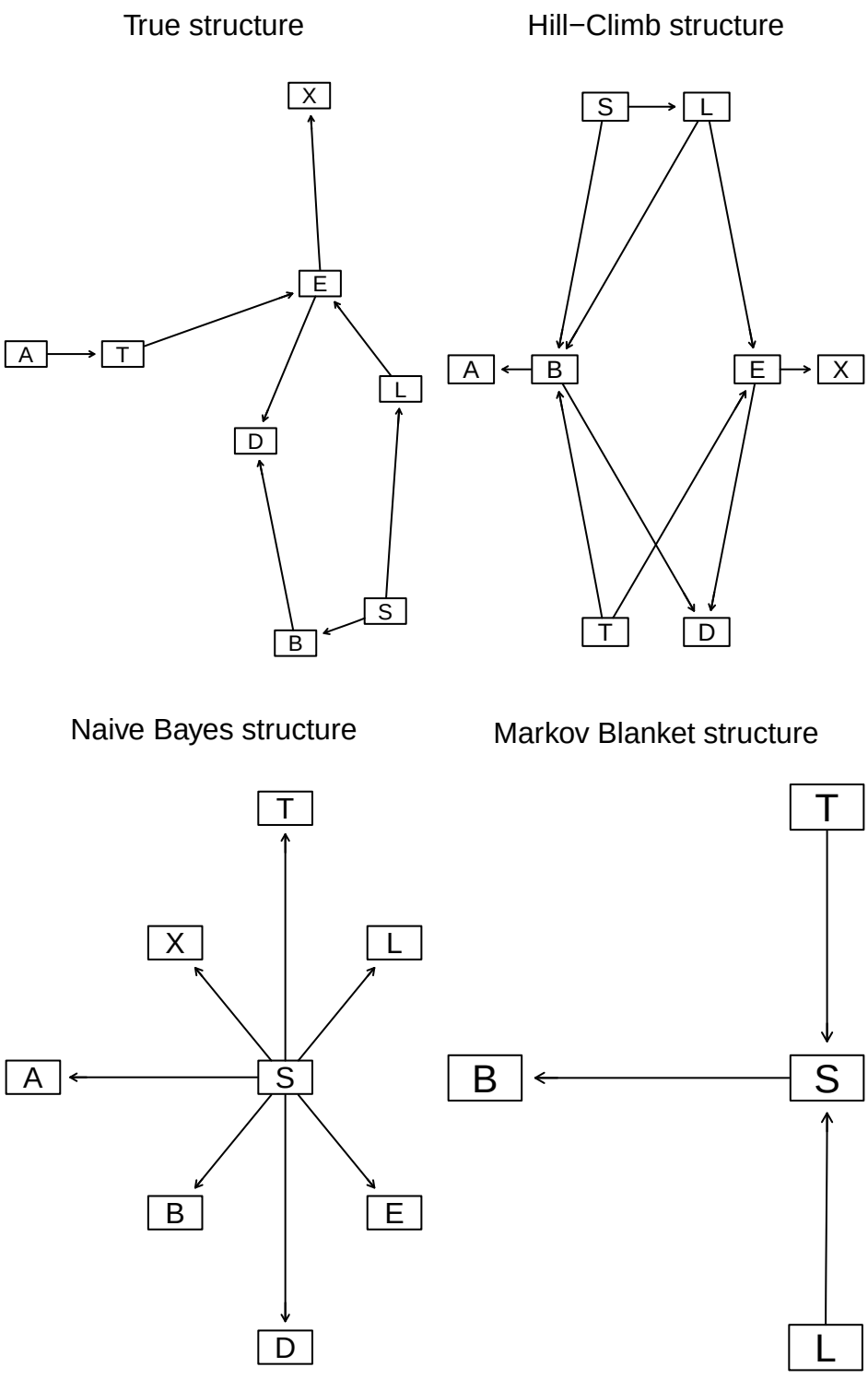
Explain why you obtain the same or different results in the exercises (2-4).

Table 9: Classification metrics for the different structures

	Accuracy	Sensitivity	Specificity
True	0.738	0.7854	0.6917
Hill-Climb	0.738	0.7814	0.6957
Markov Blanket	0.738	0.7814	0.6957
Naive Bayes	0.699	0.6457	0.7510

The table of classification metrics above shows how the models perform similarly in most cases, but identical only for two cases. Two DAGs represent the same independenceis according to the separation criterion if and only if they have the same adjacencies and unshielded colliders. The two identical results comes from the Hill-Climb generated structure and the Markov Blanket structure calculated from that. This is due to how dependencies are encoded in the networks when using variable elimination, meaning that the markov blanket variables contain conditions from the non-adjacent to S variables. In practice this could result in a network that is not as expensive to compute.

Despite Naive Bayes having a worse accuracy and sensitivity (by a rather large margin) than the other models, it has higher specificity. This means it is better at predicting no when the true value is no and worse at predicting yes when the true value is yes. The difference between the models is due to how the conditionals are set-up in the Naive Bayes model. This is likely due to the independence assumption simplifying the network dependencies too much and such important conditions are not taken into account.



Looking at the different structures for the graphs it makes sense that they converge to different values due to the node dependencies and thus the order of calculations.

Appendix

```
knitr::opts_chunk$set(echo = TRUE, message = FALSE, warning=FALSE, fig.align = 'center', fig.width = 5,
set.seed(12345)
library("RBGL")
library("Rgraphviz")
library("gRain")
library("bnlearn")
library("tidyverse")
data(asia)
asia
knitr::kable(asia[1:5,],caption="asia dataset")
library("RBGL")
library("Rgraphviz")
library("gRain")
library("bnlearn")
library("tidyverse")
# different initial structures
# true one according to documentation of asia dataset
d1 = model2network("[A][S][T|A:S][L|T][B|L][D|B][E|D][X|E]")
d2 = model2network("[B][D|B][A|D][S|A:B][T|S:D][L|T:A][E|L:B][X|E:S]")

set.seed(13)
n1 = hc(x=asia,start=d2,score="bic",restart=10)
n2 = hc(x=asia,score="bde",restart=25,iss=20)
n3 = hc(x=asia,start=d1,score="aic",restart=30)
par(mfrow=c(3,1))
graphviz.compare(n1,n2,n3,layout="circo",main=c("BIC, 10 restarts, unequivalent initial structure",
                                                "BDE, 25 restarts, 20 imaginary sample size",
                                                "AIC, 30 restarts, 'true' initial structure"))
knitr::kable(vstructs(n1))
knitr::kable(vstructs(n2))
knitr::kable(vstructs(n3))
all.equal(n1,n3)
all.equal(n1,n2)
all.equal(n2,n3)
# 80% split
index <- sample(1:nrow(asia), 0.8*nrow(asia))
train = asia[index,]
test = asia[-index,]
set.seed(13)
# find skeleton of DAG with hc algorithm
dag_hc = hc(x=train,score="aic",restart=10)
# fit hc dag on train data using MLE, change class for gRain inference and compile
fitted = compile(as.grain(bn.fit(dag_hc,train)))
set.seed(13)
ev = test[colnames(test) != "S"] # evidence test data
pred_hc = matrix(0,nrow=nrow(ev),ncol=1)

for(i in 1:nrow(ev)){
  # set evidence as value for each variable row-wise, except for S
  evidence = setEvidence(fitted,nodes=c(colnames(ev)),states=as.vector(unlist(ev[i,])))
  # predict by obtaining conditional distribution for each row-wise set of values and classifying
```

```
  pred_hc[i] = ifelse((querygrain(evidence, nodes = "S")$S["yes"]>=0.5), "yes","no")
}

conf_hc = table(pred_hc,test$S)
set.seed(13)
# true conditional structure for the dataset
dag_t = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
true = compile(as.grain(bn.fit(dag_t,train)))
truth = matrix(0,nrow=nrow(ev),ncol=1)
for(i in 1:nrow(ev)){
  evidence = setEvidence(true,nodes=c(colnames(ev)),states=as.vector(unlist(ev[i,])))
  truth[i] = ifelse((querygrain(evidence, nodes = "S")$S["yes"]>=0.5), "yes","no")
}

conf_t = table(truth,test$S)
knitr::kable(conf_hc,caption="Confusion matrix Hill-Climbing")
knitr::kable(conf_t,caption="Confusion matrix True structure")
set.seed(13)
blanket = mb(bn.fit(dag_hc,train),"S")
blanket # markov blanket variables are T, L and B
pred_mb = matrix(0,nrow=nrow(ev),ncol=1)
for(i in 1:nrow(ev)){
  evidence = setEvidence(fitted,nodes=c(blanket),states=as.vector(unlist(ev[i,blanket])))
  pred_mb[i] = ifelse((querygrain(evidence, nodes = "S")$S["yes"]>=0.5), "yes","no")
}

conf_mb = table(pred_mb,test$S)
knitr::kable(conf_mb,caption="Confusion matrix Markov Blanket")
# if data had no underlying structure and probabilities had to be hardcoded
# all created variables requires the dependencies to be encoded as new dimensions
dag_nb = model2network("[S][X|S][D|S][B|S][E|S][T|S][L|S][A|S]")

fitted = compile(as.grain(bn.fit(dag_nb,train)))
pred_nb = matrix(0,nrow=nrow(ev),ncol=1)

for(i in 1:nrow(ev)){
  evidence = setEvidence(fitted,nodes=c(colnames(ev)),states=as.vector(unlist(ev[i,])))
  pred_nb[i] = ifelse((querygrain(evidence, nodes = "S")$S["yes"]>=0.5), "yes","no")
}

conf_nb = table(pred_nb,test$S)
knitr::kable(conf_nb,caption="Confusion matrix Naive Bayes")
acc_truth = (conf_t[1,1]+conf_t[2,2])/nrow(test)
acc_HC_pred = (conf_hc[1,1]+conf_hc[2,2])/nrow(test)
acc_MB_pred = (conf_mb[1,1]+conf_mb[2,2])/nrow(test)
acc_NB_pred = (conf_nb[1,1]+conf_nb[2,2])/nrow(test)

tpr_truth = (conf_t[2,2]/(conf_t[1,2]+conf_t[2,2]))
tpr_HC = (conf_hc[2,2]/(conf_hc[1,2]+conf_hc[2,2]))
tpr_MB = (conf_mb[2,2]/(conf_mb[1,2]+conf_mb[2,2]))
tpr_NB = (conf_nb[2,2]/(conf_nb[1,2]+conf_nb[2,2]))

tnr_truth = (conf_t[1,1]/(conf_t[1,1]+conf_t[2,1]))
```

```
tnr_HC = (conf_hc[1,1]/(conf_hc[1,1]+conf_hc[2,1]))
tnr_MB = (conf_mb[1,1]/(conf_mb[1,1]+conf_mb[2,1]))
tnr_NB = (conf_nb[1,1]/(conf_nb[1,1]+conf_nb[2,1]))

accuracies = matrix(c(acc_truth, acc_HC_pred, acc_MB_pred, acc_NB_pred,
                      tpr_truth, tpr_HC, tpr_MB, tpr_NB,
                      tnr_truth,tnr_HC,tnr_MB,tnr_NB), ncol = 3)
colnames(accuracies) = c("Accuracy", "Sensitivity","Specificity")
rownames(accuracies) = c("True","Hill-Climb","Markov Blanket", "Naive Bayes")

knitr::kable(accuracies,caption="Classification metrics for the different structures", digits = 4)
par(mfrow=c(2,2))
graphviz.plot(dag_t,layout="circo",main="True structure")
graphviz.plot(dag_hc,layout="circo",main="Hill-Climb structure")
graphviz.plot(dag_nb,layout="circo",main="Naive Bayes structure")
graphviz.plot(model2network("[T][L][S|T:L][B|S]"),layout="circo",main="Markov Blanket structure")
```