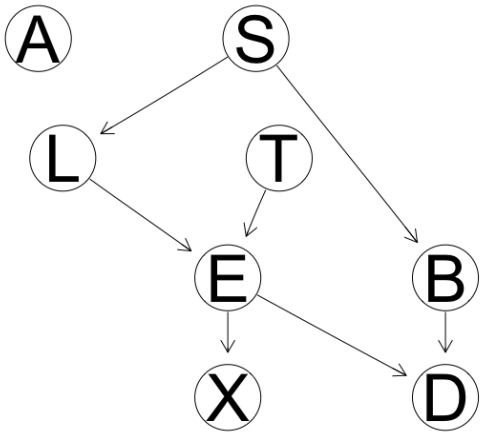
TDDE15 – Lab 1: Graphical models

hanma409 – Hannes Bengtsson

Exercise 1

By using different network scores, the hill-climbing algorithm can return non-equivalent Bayesian network (BN) structures. In the two examples below I've used the Bayesian Information Criterion score (BIC) and the Akaike Information Criterion score (AIC) respectively. In the graphs below we note that we get a different set of unshielded colliders and adjacencies when we use different network scores for the hill-climbing algorithm. Thus, the two DAGs below represent non-equivalent Bayesian network structures. The hill-climbing algorithm is a heuristic, therefore it might get stuck in local optimums, which is why it can result in different BN:s as in the examples below.



BN from hill-climbing using BIC

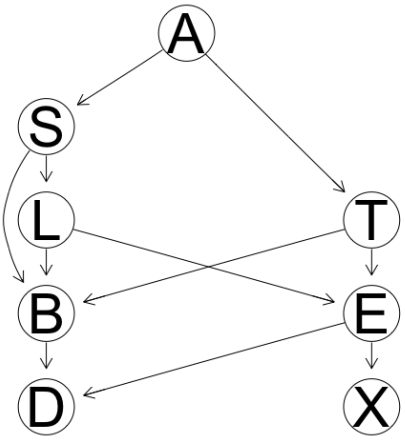
Unshielded colliders

- T → E ← L
- B → D ← E

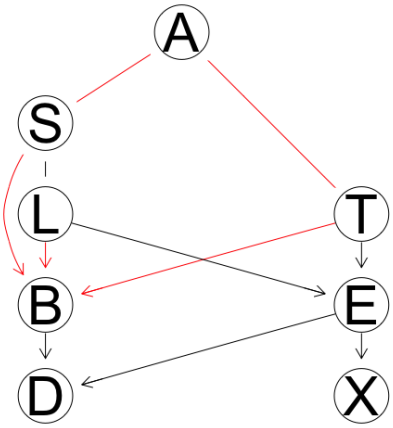
BN from hill-climbing using AIC

Unshielded colliders

- S → B ← T



- T → B ← L
- T → E ← L
- B → D ← E



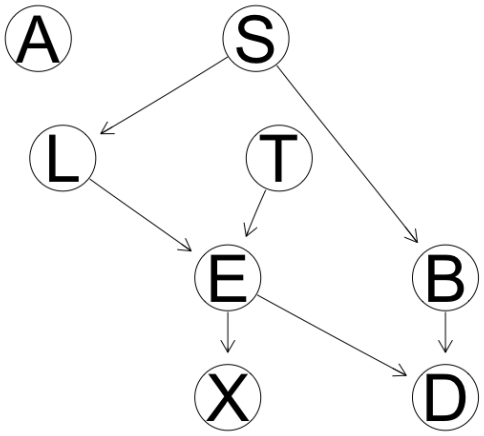
Comparison of the underlying structure for the two Bayesian Networks

- Comparison of the completed partially directed acyclic graphs for the BN above
- Edges that differ between the two are highlighted in red

Exercise 2

To learn the structure for the BN from 80% of the Asia dataset I chose to use the hill-climbing algorithm (without altering any arguments). To learn the parameters for the BN I used *bn.fit()*. To classify the remaining 20% of the Asia dataset in two classes: *S = yes* and *S = no*, I computed the posterior probability distribution for *S* for each case and classified it in the most likely class. I used exact inference to do so. The confusion matrix for the classification is shown below:

Actual / Predicted	*S = yes*	*S = no*
S = yes	366	121
S = no	176	337

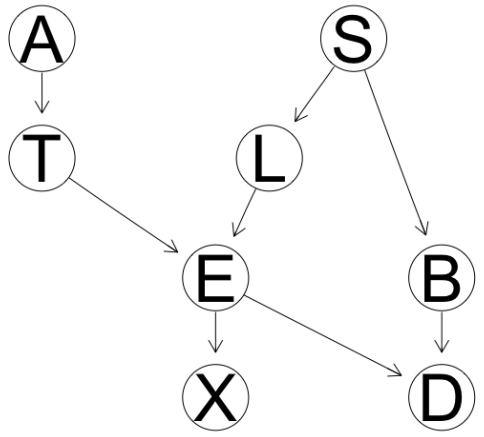


BN from 80% of Asia dataset

- Using the hill-climbing algorithm

By repeating the process for the true Asia BN we get the confusion matrix for the classification of S below:

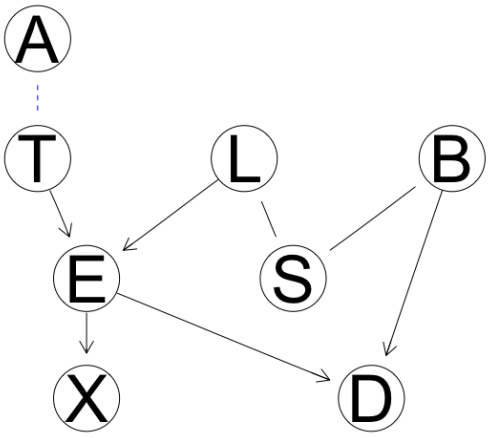
Actual / Predicted	*S = yes*	*S = no*
S = yes	366	121
S = no	176	337



The true BN from Asia

When comparing the two confusion matrixes we note that they contain the same values. Hence, the two BN:s are equally good at classifying S.

Comparison of the underlying structure for the two Bayesian Networks



When comparing the structure for the two BN:s we note that the true BN contains an edge between node A and T which the other BN lacks. However, given T S is separated from A in the graph and thus S is independent of T in the posterior probability distribution.

Exercise 3

The Markov blanket of node *S* consists of node *L* and node *B*. If we use 80% of the Asia dataset to learn the structure and parameters of the BN but only use these two nodes to classify *S* we get the following confusion matrix:

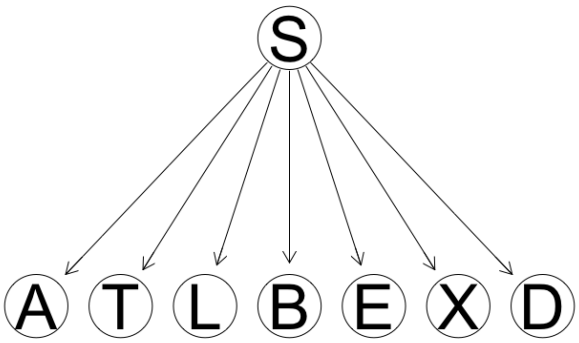
Actual / Predicted	*S = yes*	*S = no*
S = yes	366	121
S = no	176	337

When comparing this confusion matrix with our previous ones we note that it contains the same values. Hence, by using the Markov Blanket for *S*, a classification as good as when we use all nodes of the BN is obtained, as in the cases above. This makes sense if we look at the underlying structure for the two BN:s above. By observing the graphs we note that *S* is separated from the rest of the BN given *L* and given *B*. Hence, given *L* and *B*, *S* is said to be independent of all other nodes in the posterior probability distribution. Which means that the classification of S only depends on node *L* and *B*. Thus, all other nodes can be excluded when classifying *S* since they won't contribute to a more accurate classification of *S*.

Exercise 4

Using a naive Bayes classifier modeled as a BN to predict *S* results in fewer true positives and true negatives compared to the other classifiers above, 307+359 = 666 compared to 366+337 = 703. Hence, using a naive Bayes classifier results in fewer correct predictions of *S* compared to using the other BN:s above.

Actual / Predicted	*S = yes*	*S = no*
S = yes	307	180
S = no	154	359



BN from naive Bayes classifier

We assume that S proceeds all other nodes and that all other nodes are independent

Exercise 5

In exercise 2 and 3 we obtain the same results because if we observe the nodes in *S*:s Markov blanket then S is independent of all other nodes in the Bayesian network. Therefore the additional nodes, nodes that aren't in the Markov blanket of *S,* doesn't tell us anything about class for the variable that we want to predict, *S*. Hence, to predict *S* it's sufficient to use the nodes in its Markov blanket.

When comparing the results from exercise 2 and 3 with that of exercise 4 the results differ. The reason behind this is because of the assumption that *S* proceeds all other nodes and all other nodes are independent when using the naive Bayes classifier. As we can see in previous exercise this assumption doesn't accurately reflects the data's underlying dependencies. In this case the predictions from the naive Bayes classifier therefore results in fewer truthful prediction.

Code

Exercise 1

```
install.packages("bnlearn")
library(bnlearn)
```

```
data("asia")
# Initiate hill climb
hc1 <- hc(asia, score="bic")
hc2 <- hc(asia, score="aic")
install.packages("Rgraphviz")
library(Rgraphviz)
# Plotting the graphs
graphviz.plot(hc1)
graphviz.plot(hc2)
graphviz.plot(cpdag(hc1))
graphviz.plot(cpdag(hc2))
# Checking the structure
vstructs(hc1)
vstructs(hc2)
arcs(hc1)
arcs(hc2)
all.equal(hc1,hc2)
graphviz.compare(cpdag(hc1), cpdag(hc2))
```

Exercise 2

```
# Divide data
n<-dim(asia)[1]
set.seed(12345)
id<-sample(1:n, floor(n*0.8))
train<-asia[id,]
test<-asia[-id,]
# Install gRain
if (!requireNamespace("BiocManager", quietly = TRUE))
install.packages("BiocManager")
BiocManager::install("RBGL", force=TRUE)
BiocManager::install("Rgraphviz")
BiocManager::install("gRain")
library(gRain)
# Learn the structure using Hill Climbing algorithm
bn_structure <- hc(train)
graphviz.plot(bn_structure)
# Estimate parameters of the Bayesian Network
bn_parameters <- bn.fit(bn_structure, train, method="mle")
bn_grain <- as.grain(bn_parameters)
# Initialize an empty vector to store posterior probabilities
posterior_probs <-numeric(nrow(test))
# Find the posterior probabilities
for(i in 1:1000){
evidence <-setEvidence(bn_grain,nodes=c("A","T","L","B","E","X","D"),states=as.character(u
nlist(test[i, -2])))
prob <- querygrain(evidence)
posterior_probs[i] <- prob$S["yes"]
}
# Classify based on the most likely class
```

```
predicted_classes <- ifelse(posterior_probs > 0.5, "yes", "no")
results <- data.frame(ground_truth = test$S, predicted = predicted_classes)
confusion_matrix <- table(results$predicted,results$ground_truth)
# Calculate true/false positives/negatives
true_positive <- confusion_matrix["yes", "yes"]
false_positive <- confusion_matrix["no", "yes"]
true_negative <- confusion_matrix["no", "no"]
false_negative <- confusion_matrix["yes", "no"]
# Print the confusion matrix
confusion_matrix
dag <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
graphviz.plot(dag)
graphviz.compare(cpdag(dag), cpdag(bn_structure))
dag.grain <- as.grain(bn.fit(dag,train))
posterior_probs <-numeric(nrow(test))
for(i in 1:1000){
evidence <-setEvidence(dag.grain,nodes=c("A","T","L","B","E","X","D"),states=as.character
(unlist(test[i, -2])))
prob <- querygrain(evidence)
posterior_probs[i] <- prob$S["yes"]
}
# Classify based on the most likely class
predicted_classes <- ifelse(posterior_probs > 0.5, "yes", "no")
# Create a data frame with ground truth and predicted values
results <- data.frame(ground_truth = test$S, predicted = predicted_classes)
# Calculate the confusion matrix
confusion_matrix <- table(results$predicted,results$ground_truth)
# Calculate true/false positives/negatives
true_positive <- confusion_matrix["yes", "yes"]
false_positive <- confusion_matrix["no", "yes"]
true_negative <- confusion_matrix["no", "no"]
false_negative <- confusion_matrix["yes", "no"]
# Print the confusion matrix
confusion_matrix
```

Exercise 3

```
S_mb <- mb(bn_structure, node="S")
bn_structure <- hc(train)
bn_parameters <- bn.fit(bn_structure, train, method="mle")
bn_grain <- as.grain(bn_parameters)
filtered_test <- test[4:5]
posterior_probs <-numeric(nrow(filtered_test))
for(i in 1:1000){
evidence <-setEvidence(bn_grain,nodes=c(S_mb),states=as.character(unlist(filtered_test
[i,])))
prob <- querygrain(evidence)
posterior_probs[i] <- prob$S["yes"]
}
predicted_classes <- ifelse(posterior_probs > 0.5, "yes", "no")
```

```
results <- data.frame(ground_truth = test$S, predicted = predicted_classes)
confusion_matrix <- table(results$predicted ,results$ground_truth)
true_positive <- confusion_matrix["yes", "yes"]
false_positive <- confusion_matrix["no", "yes"]
true_negative <- confusion_matrix["no", "no"]
false_negative <- confusion_matrix["yes", "no"]
# Print the confusion matrix
confusion_matrix
```

Exercise 4

```
e <- empty.graph(c("A","S","T","L","B","E","X","D"))
class(e)
graphviz.plot(e)
arc <-matrix(c("S", "A", "S", "T", "S", "L", "S", "B","S", "E", "S", "X", "S", "D"),
ncol = 2, byrow = TRUE,
dimnames = list(NULL, c("from", "to")))
arcs(e) <- arc
graphviz.plot(e)
e.grain <- as.grain(bn.fit(e,train))
posterior_probs <-numeric(nrow(test))
for(i in 1:1000){
evidence <-setEvidence(e.grain,nodes=c("A","T","L","B","E","X","D"),states=as.character(un
list(test[i, -2])))
prob <- querygrain(evidence)
posterior_probs[i] <- prob$S["yes"]
}
# Classify based on the most likely class
predicted_classes <- ifelse(posterior_probs > 0.5, "yes", "no")
# Create a data frame with ground truth and predicted values
results <- data.frame(ground_truth = test$S, predicted = predicted_classes)
# Calculate the confusion matrix
confusion_matrix <- table(results$predicted,results$ground_truth)
# Calculate true/false positives/negatives
true_positive <- confusion_matrix["yes", "yes"]
false_positive <- confusion_matrix["no", "yes"]
true_negative <- confusion_matrix["no", "no"]
false_negative <- confusion_matrix["yes", "no"]
# Print the confusion matrix
confusion_matrix
```