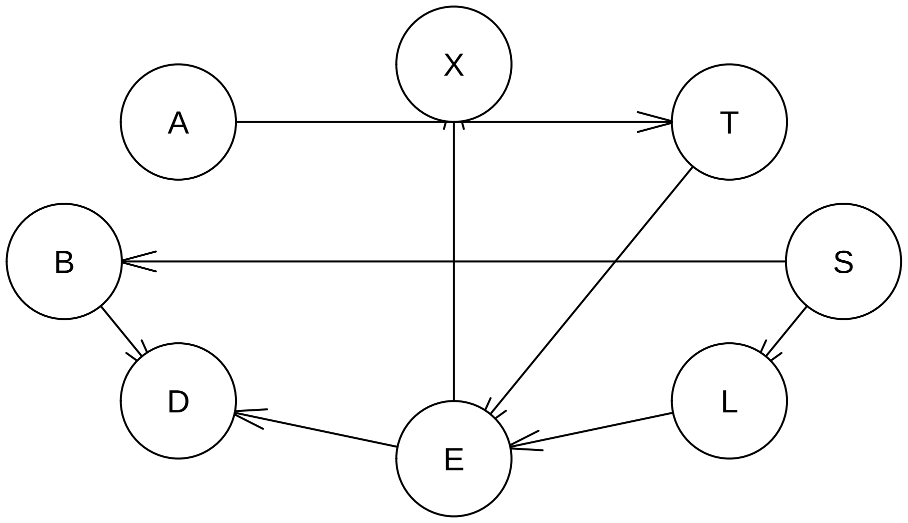
Lab1

Christian von Koch

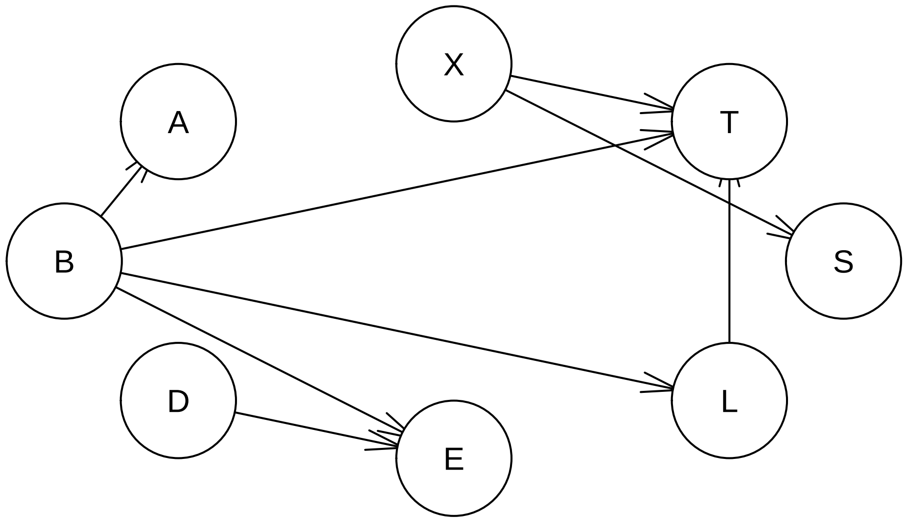
2020-09-11

Assignment 1

```r
data("asia")
init_dag1 = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
init_dag2 = model2network("[B][D][X][E|D:B][L|B][T|L:B:X][S|X][A|B]")
plot(init_dag1)
```



```r
plot(init_dag2)
```



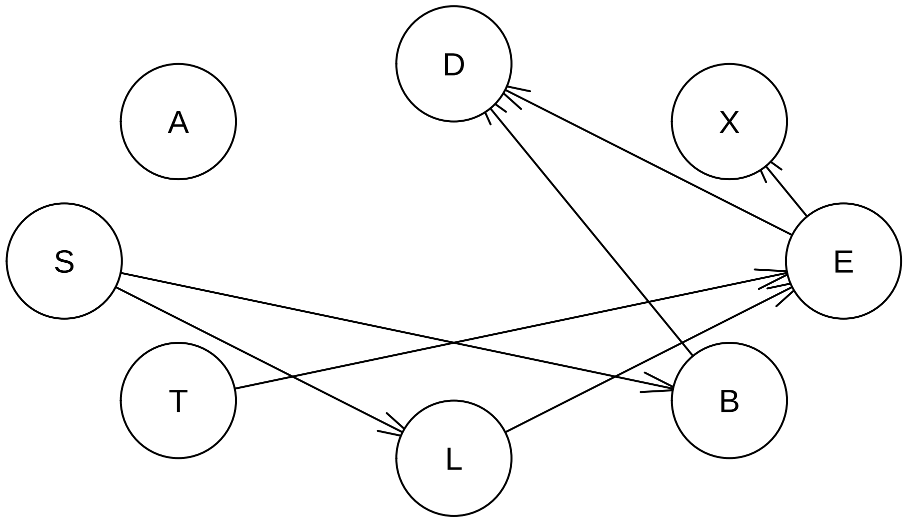
```
random_restarts=c(1,10,1000)
scores=c("aic", "bic", "bde")
iss=c(1,10,100)

# Testing different initial structures
model1_init=hc(asia, start=init_dag1)
model2_init=hc(asia, start=init_dag2)
all.equal(model1_init, model2_init)
```

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

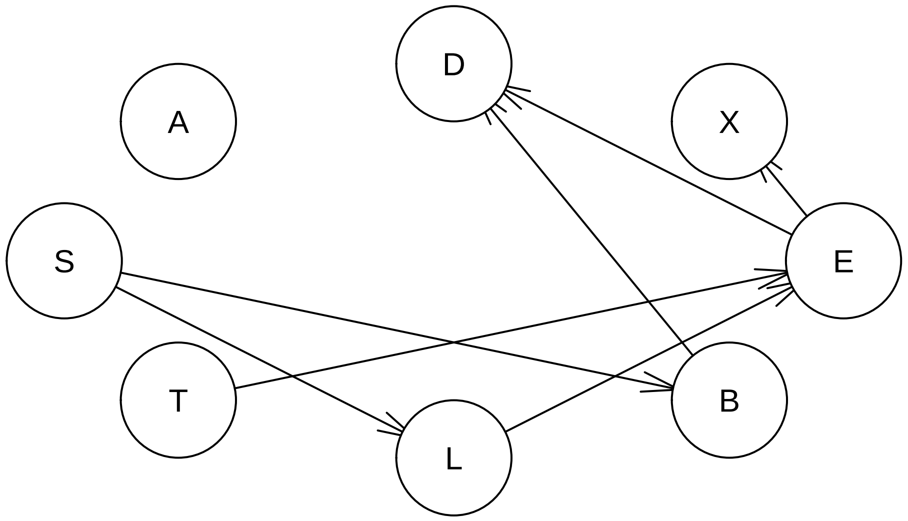
```
# Changed the resulting network
```

It is clear that different initial structures inputted to the algorithm can result in different non-equivalent Bayesian Networks outputted by the Hill Climbing Algorithm. This is due to the fact that the hill climbing algorithm can terminate at different *local* optima points, i.e. the algorithm is not guaranteed to find the *global* optimum. By choosing different starting points for the algorithm it is obvious that the algorithm might end up in different *local* optima outputs.

```
# Testing different random restarts
model1_random=hc(asia, restart=random_restarts[1])
model2_random=hc(asia, restart=random_restarts[2])
model3_random=hc(asia, restart=random_restarts[3])
plot(model3_random)
```



```r
plot(model2_random)
```



```r
all.equal(model1_random, model2_random)
```

```
## [1] TRUE
```

```r
all.equal(model1_random, model3_random)
```

```
## [1] TRUE
```

```r
all.equal(model2_random, model3_random)
```

```
## [1] TRUE
```

```r
# Changed the resulting network
```

By choosing different values of random restarts we tell the algorithm to do the algorithm a specified number of times since the different paths which the algorithm can choose for each step can be many and are chosen randomely. This is due to the fact that many different proposed networks from a certain point in the algorithm can result in the same overall score for that specific network after which the algorithm chooses a structure randomely from the set of structures with equal (and highest) scores. By defining the number of random restarts the algorithm might end up in different *local* optima, i.e. different networks with different scores after which the algorithm outputs the network with the highest score. As seen above the network modelled with 1000 random restarts was different in comparison to the models which only used 1 and 10 restarts respectively. This particular model also had a *higher* score than the other ones which also can be confirmed by comparing to the true network.

```r
# Testing different score models
model1_score=hc(asia, score=scores[1])
model2_score=hc(asia, score=scores[2])
model3_score=hc(asia, score=scores[3])
```

```
all.equal(model1_score, model2_score)
```

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

```
all.equal(model1_score, model3_score)
```

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

```
all.equal(model2_score, model3_score)
```

```
## [1] TRUE
```

```
# Fundamentally changed the resulting network
```

By choosing different score models we see from the output above that the HC algorithm might output different non-equivalent BNs. This is because by choosing different score models the networks are evaluated differently for each step, which evidently can result in the algorithm to be terminated at different *local* optimas, i.e. output different non-equivalent Bayesian Networks.

```
# Testing different imaginary sample sizes
model1_iss=hc(asia, score="bde", iss=iss[1])
model2_iss=hc(asia, score="bde", iss=iss[2])
model3_iss=hc(asia, score="bde", iss=iss[3])
all.equal(model1_iss, model2_iss)
```

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

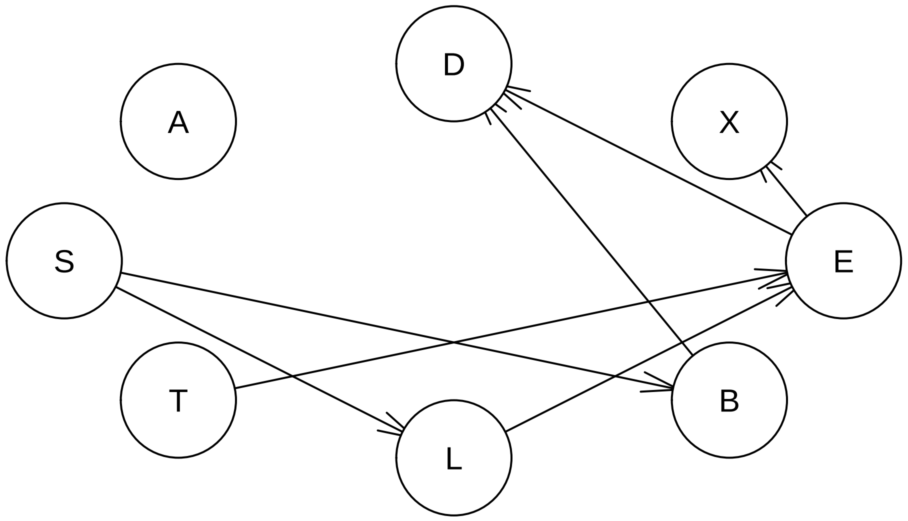
```
all.equal(model1_iss, model3_iss)
```

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

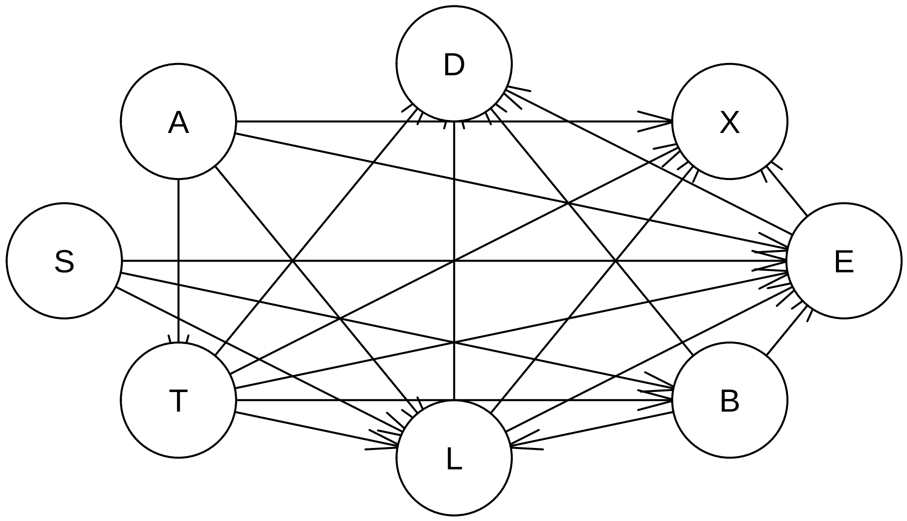
```
all.equal(model2_iss, model3_iss)
```

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

```
plot(model1_iss)
```



```r
plot(model3_iss)
```



It is evident from the above result that different sizes of imaginary sample size also might result in different outputted BNs by the HC algorithm. This is due to the fact that the imaginary sample size represent how much regularization that should be applied when running the algorithm. By specifying a high imaginary sample size the model outputted are more complex and contains more parameters which might result in an overfitted model. A low imaginary sample size on the other hand corresponds to high regularization applied to the model which results in a more sparse model outputted by the algorithm. Above, this is shown by the outputs when using imaginary sample size 1 and 100 where it is evident that the second output which used ISS of 100 is much more complex and contains more edges (which means more parameters) in comparison to the more sparse model shown in the first output.

Assignment 2

```
set.seed(12345)
data("asia")
n=dim(asia)[1]
id=sample(1:n, floor(n*0.8))
train=asia[id,]
test=asia[-id,]

predictNet <- function(juncTree, data, features, target){
  predArray <- matrix(nrow=nrow(data),ncol=1)
  for(i in 1:nrow(data)){
    obsStates <- NULL
    for(p in features){
      if(data[i,p]=="yes"){
```

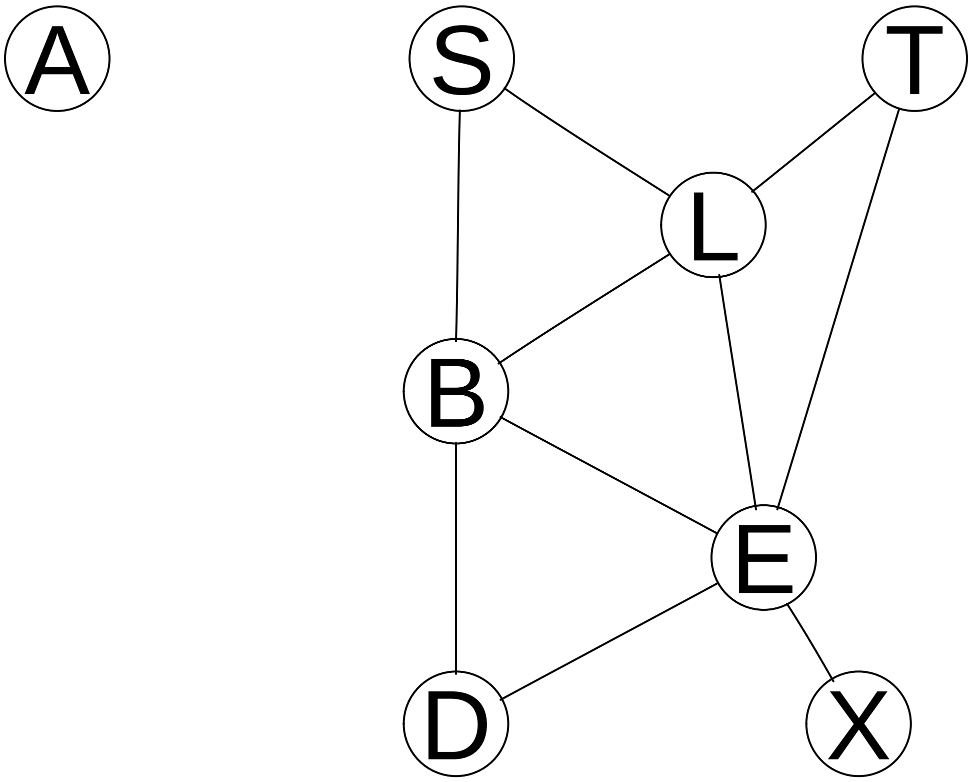
```r
        obsStates <- c(obsStates,"yes")
      } else{
        obsStates <- c(obsStates,"no")
      }
    }


    obsEvidence <- setEvidence(object = juncTree,
                               nodes = features,
                               states = obsStates)
    obsPredProb <- querygrain(object = obsEvidence,
                              nodes = target)$S
    predArray[i] <- if(obsPredProb["yes"]>=0.5) "yes" else "no"
  }
  return(predArray)
}

# Train model
BNmodel=hc(train)
real_dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
fit=bn.fit(BNmodel, train)
fit_real=bn.fit(real_dag, train)
fitTable=as.grain(fit)
fitTable_real=as.grain(fit_real)
junctionTree=compile(fitTable)
junctionTree_real=compile(fitTable_real)

obsVars=c("A", "T", "L", "B", "E", "X", "D")
tarVar=c("S")
predictionTest <- predictNet(junctionTree, test, obsVars, tarVar)
predictionTrue <- predictNet(junctionTree_real, test, obsVars, tarVar)
confTable=table(predictionTest, test$S)
confTable_real=table(predictionTrue, test$S)
confTable
```

```
##
## predictionTest  no yes
##            no  337 121
##            yes 176 366
```

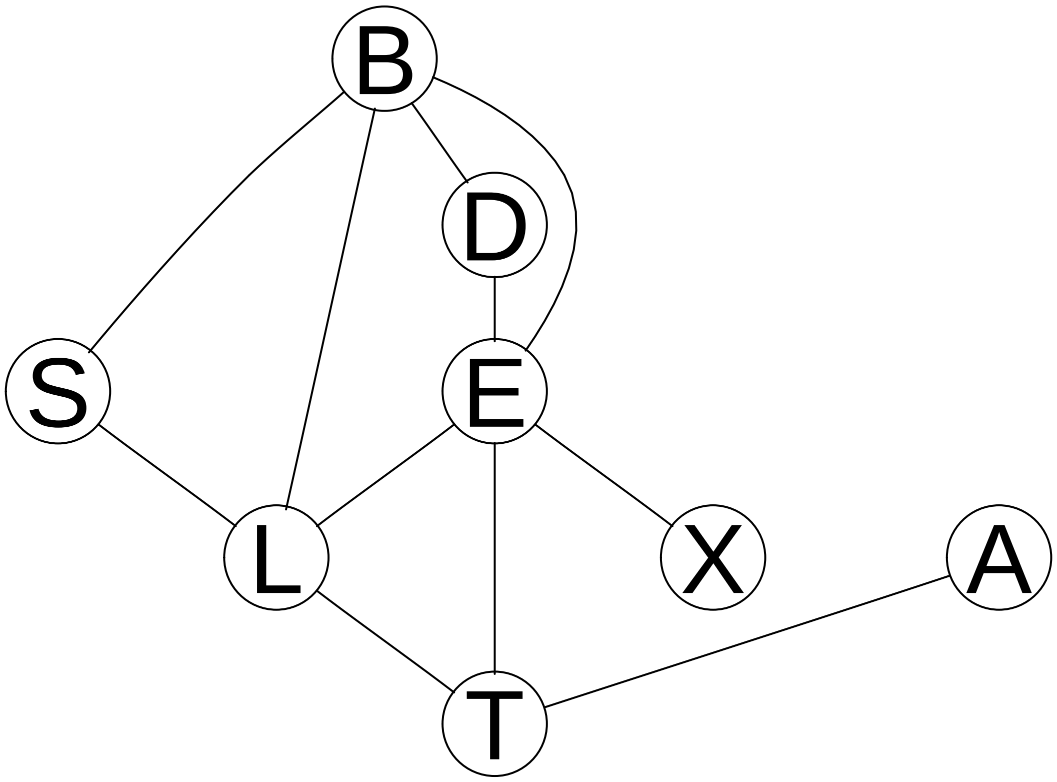
```r
confTable_real
```

```
##
## predictionTrue  no yes
##            no  337 121
##            yes 176 366
```

```r
plot(junctionTree)
```



```r
plot(junctionTree_real)
```



As seen in the above result the trained DAG outputted by the HC algorithm produced the same confusion table as the real DAG. This implies that the two DAGs are equivalent when looking at the target value S. This is because the markov blanket of S is the same in the two graphs which can be seen in the two plots above. The result of this is that the same model in the end is used when specifically looking at the variable S, investigating the conditional probabilities.

Assignment 3

```
markovBlanket=mb(fit_real, node="S")
print(markovBlanket)
```

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

```
predictMarkovBlanket=predictNet(junctionTree, test, markovBlanket, tarVar)
confTable_MB=table(predictMarkovBlanket, test$S)
confTable_MB
```

```
##
## predictMarkovBlanket  no yes
##                  no  337 121
##                  yes 176 366
```

As seen from the results above, the same confusion matrix is once again outputted. This is in line with the previous explanation, i.e. now we only used the markov blanket for S which naturally once again produces the same result as the markov blanket is the only features that matters when calculating the conditional probabilities of S.

Assignment 4

```r
set.seed(12345)
naive_dag=model2network("[S][A|S][B|S][X|S][T|S][L|S][E|S][D|S]")
naive_fit=bn.fit(naive_dag, train)
fitNaive=as.grain(naive_fit)
naive_JunctionTree=compile(fitNaive)
naivePred=predictNet(naive_JunctionTree, test, obsVars, tarVar)
naiveConf=table(naivePred, test$S)
naiveConf
```

```
##
## naivePred  no yes
##       no  359 180
##       yes 154 307
```

As seen above, now the confusion table is different. This is because the markov blanket has now changed as a result of the new naive bayes model used with respect to S. Now the markov blanket contains all variables in comparison to before when only feature B and L were included. This in turn results in a different outputted confusion matrix since the conditional probabilities for S calculated by the model will be different and as a consequence classify the test data differently.

Assignment 5

This has been explained in each of the exercises - the results vary depending on the resulting markov blanket of S from the model used.

Appendix

```r
## 1. Show that multiple runs of the hill-climbing algorithm can return non-equivalent Bayesian network
## Explain why this happens. Use the Asia dataset which is included in the bnlearn package. To load the
## run data("asia").

library(bnlearn)
data("asia")
init_dag1 = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
init_dag2 = model2network("[B][D][X][E|D:B][L|B][T|L:B:X][S|X][A|B]")
plot(init_dag1)
plot(init_dag2)
random_restarts=c(1,10,100)
scores=c("aic", "bic", "bde")
iss=c(1,10,100)

# Testing different initial structures
model1_init=hc(asia, start=init_dag1)
model2_init=hc(asia, start=init_dag2)
plot(model1_init)
plot(model2_init)
all.equal(model1_init, model2_init)

# Changed the resulting network

# Testing different random restarts
model1_random=hc(asia, restart=random_restarts[1])
model2_random=hc(asia, restart=random_restarts[2])
model3_random=hc(asia, restart=random_restarts[3])
plot(model1_random)
plot(model2_random)
plot(model3_random)
all.equal(model1_random, model2_random)
all.equal(model1_random, model3_random)
all.equal(model2_random, model3_random)

# Changed the resulting network

# Testing different score models
model1_score=hc(asia, score=scores[1])
model2_score=hc(asia, score=scores[2])
model3_score=hc(asia, score=scores[3])
plot(model1_score)
plot(model2_score)
plot(model3_score)
all.equal(model1_score, model2_score)
all.equal(model1_score, model3_score)
all.equal(model2_score, model3_score)

# Fundamentally changed the resulting network

# Testing different imaginary sample sizes
model1_iss=hc(asia, score="bde", iss=iss[1])
```

```r
model2_iss=hc(asia, score="bde", iss=iss[2])
model3_iss=hc(asia, score="bde", iss=iss[3])
plot(model1_iss)
plot(model2_iss)
plot(model3_iss)
all.equal(model1_iss, model2_iss)
all.equal(model1_iss, model3_iss)
all.equal(model2_iss, model3_iss)

# Fundamentally changed the resulting network

## 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]").

set.seed(12345)
data("asia")
n=dim(asia)[1]
id=sample(1:n, floor(n*0.8))
train=asia[id,]
test=asia[-id,]

predictNet <- function(juncTree, data, features, target){
  predArray <- matrix(nrow=nrow(data),ncol=1)
  for(i in 1:nrow(data)){
    obsStates <- NULL
    for(p in features){
      if(data[i,p]=="yes"){
        obsStates <- c(obsStates,"yes")
      } else{
        obsStates <- c(obsStates,"no")
      }
    }


    obsEvidence <- setEvidence(object = juncTree,
                               nodes = features,
                               states = obsStates)
    obsPredProb <- querygrain(object = obsEvidence,
                              nodes = target)$S
    predArray[i] <- if(obsPredProb["yes"]>=0.5) "yes" else "no"
  }
  return(predArray)
}
```

```
# Train model
BNmodel=hc(train)
real_dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
fit=bn.fit(BNmodel, train)
fit_real=bn.fit(real_dag, train)
fitTable=as.grain(fit)
fitTable_real=as.grain(fit_real)
junctionTree=compile(fitTable)
junctionTree_real=compile(fitTable_real)

obsVars=c("A", "T", "L", "B", "E", "X", "D")
tarVar=c("S")
predictionTest <- predictNet(junctionTree, test, obsVars, tarVar)
predictionTrue <- predictNet(junctionTree_real, test, obsVars, tarVar)
confTable=table(predictionTest, test$S)
confTable_real=table(predictionTrue, test$S)
confTable
confTable_real

## 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.

markovBlanket=mb(fit_real, node="S")
print(markovBlanket)
predictMarkovBlanket=predictNet(junctionTree, test, markovBlanket, tarVar)
confTable_MB=table(predictMarkovBlanket, test$S)
confTable_MB

## 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

set.seed(12345)
naive_dag=model2network("[S][A|S][B|S][X|S][T|S][L|S][E|S][D|S]")
naive_fit=bn.fit(naive_dag, train)
fitNaive=as.grain(naive_fit)
naive_JunctionTree=compile(fitNaive)
naivePred=predictNet(naive_JunctionTree, test, obsVars, tarVar)
naiveConf=table(naivePred, test$S)
naiveConf

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

## Markov Blanket (MB) consists of the nodes of importance for the target nodes' dependencies in the ne
## If we do not change the MB, the outcome of the prediction on the target node will not change.
```