Lab_1

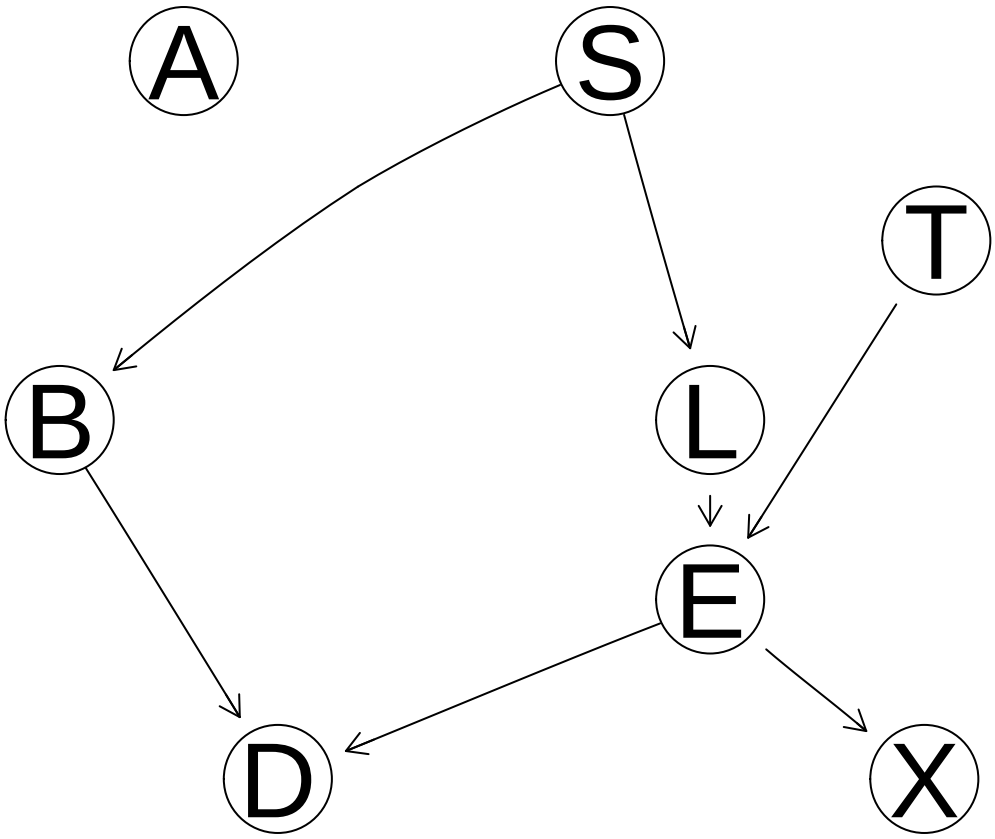
Shipeng Liu

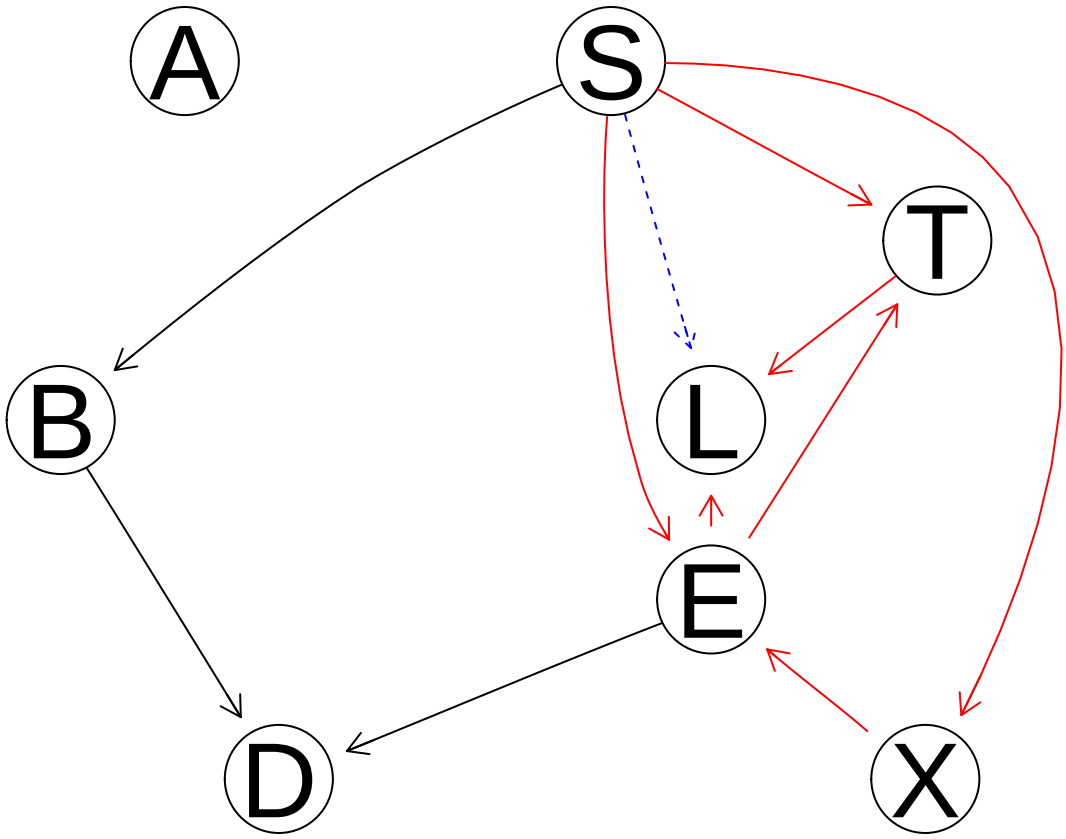
2023-09-08

(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.

```
set.seed(12345)

showDifference=function(g1,g2){
  cat("  Is two graph equivalent?",all.equal(cpdag(g1),cpdag(g2)))
  graphviz.compare(g1,g2)
  print(cpdag(g1))
  print(cpdag(g2))
}

data("asia")


# Different initial graph
dag_1=hc(asia)
init_dag=random.graph(names(asia),num=1,method = "ordered")
dag_2=hc(asia,start = init_dag)

showDifference(dag_1,dag_2)
```

```
##   Is two graph equivalent? Different number of directed/undirected arcs
```


A
S
T
B
L
E
D
X

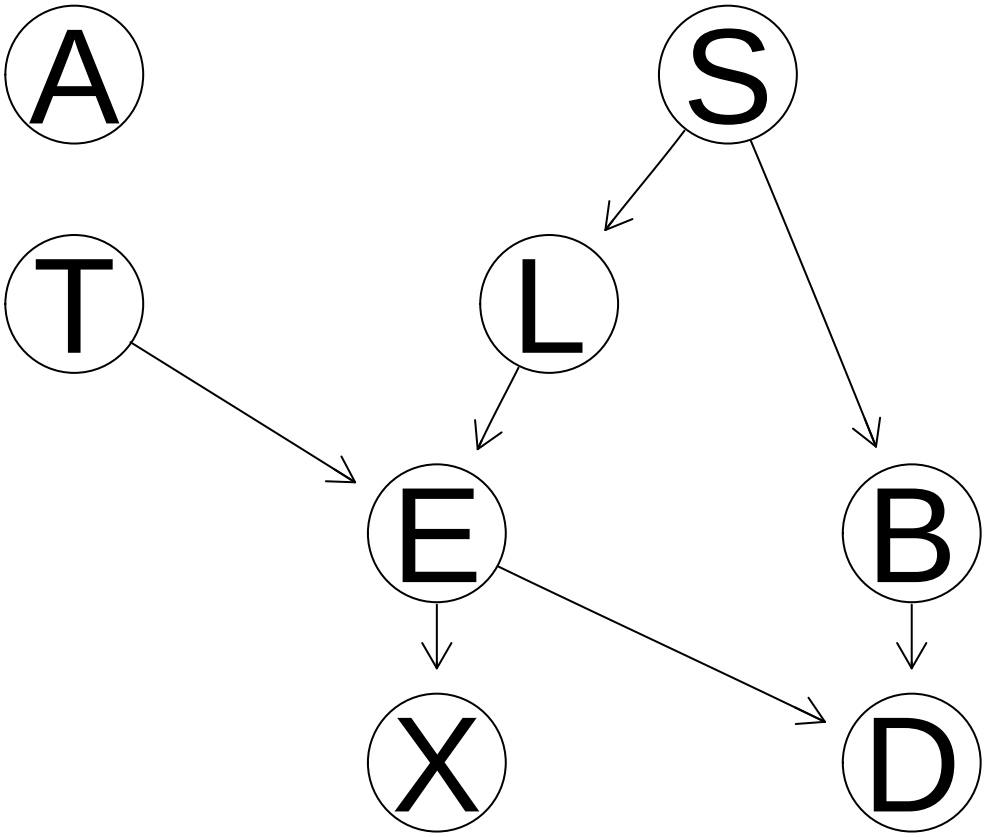


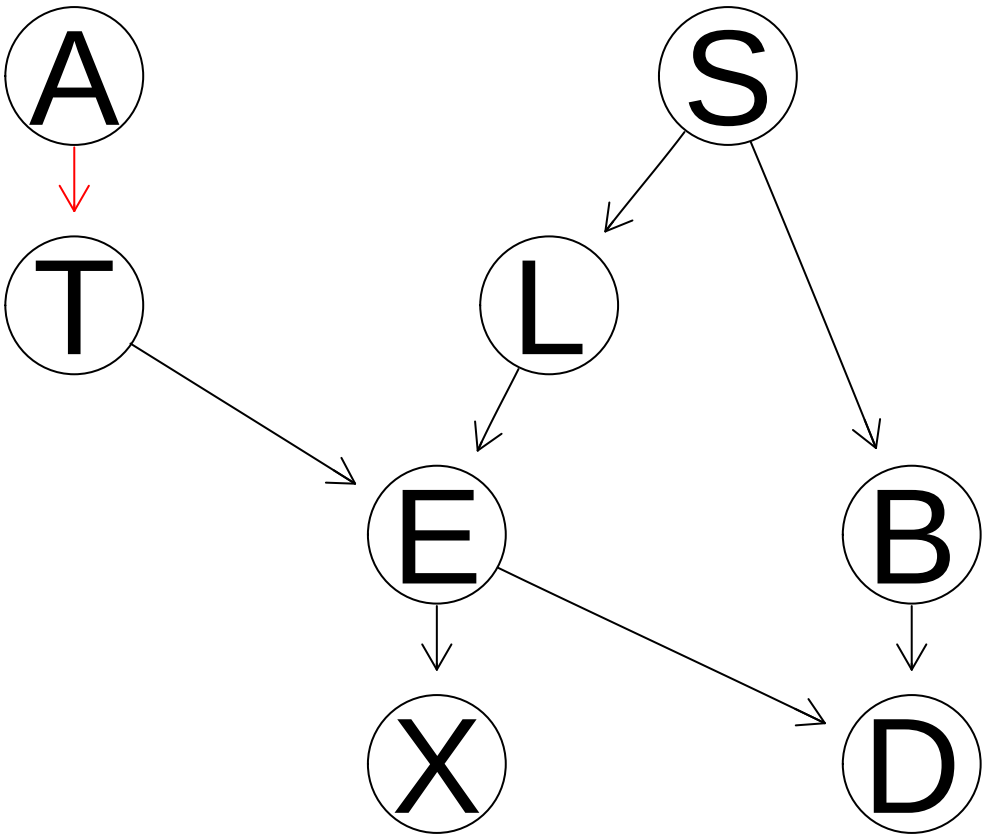

```
##
##   Bayesian network learned via Score-based methods
##
##   model:
##     [partially directed graph]
##   nodes:                                8
##   arcs:                                 7
##     undirected arcs:                    2
##     directed arcs:                      5
##   average markov blanket size:          2.25
##   average neighbourhood size:           1.75
##   average branching factor:             0.62
##
##   learning algorithm:                   Hill-Climbing
##   score:                                BIC (disc.)
##   penalization coefficient:             4.258597
##   tests used in the learning procedure: 77
##   optimized:                            TRUE
##
##
##   Bayesian network learned via Score-based methods
##
##   model:
##     [partially directed graph]
##   nodes:                                8
##   arcs:                                 10
```

```
##     undirected arcs:                       8
##     directed arcs:                         2
##   average markov blanket size:             2.75
##   average neighbourhood size:              2.50
##   average branching factor:                0.25
##
##   learning algorithm:                      Hill-Climbing
##   score:                                   BIC (disc.)
##   penalization coefficient:                4.258597
##   tests used in the learning procedure:    141
##   optimized:                               TRUE
```

```
# Different iss
dag_3=hc(asia,score='bde',iss=1)
dag_4=hc(asia,score='bde',iss=2)

showDifference(dag_3,dag_4)
```

```
##   Is two graph equivalent? Different number of directed/undirected arcs
```






```
##
##   Bayesian network learned via Score-based methods
##
##   model:
##     [partially directed graph]
##   nodes:                                8
##   arcs:                                 7
##     undirected arcs:                    2
##     directed arcs:                      5
##   average markov blanket size:          2.25
##   average neighbourhood size:           1.75
##   average branching factor:             0.62
##
##   learning algorithm:                   Hill-Climbing
##   score:                                Bayesian Dirichlet (BDe)
##   graph prior:                          Uniform
##   imaginary sample size:                1
##   tests used in the learning procedure: 77
##   optimized:                            TRUE
##
##
##   Bayesian network learned via Score-based methods
##
##   model:
##     [partially directed graph]
##   nodes:                                8
```

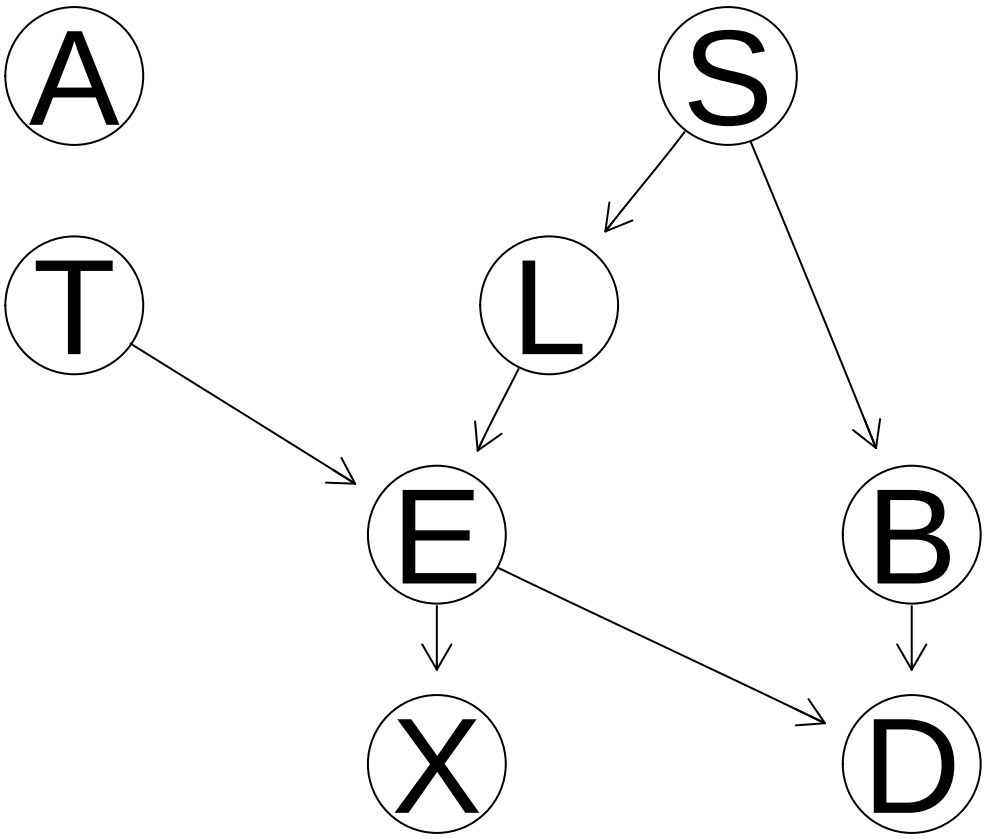
```
##   arcs:                                 8
##     undirected arcs:                    3
##     directed arcs:                      5
##   average markov blanket size:          2.50
##   average neighbourhood size:           2.00
##   average branching factor:             0.62
##
##   learning algorithm:                   Hill-Climbing
##   score:                                Bayesian Dirichlet (BDe)
##   graph prior:                          Uniform
##   imaginary sample size:                2
##   tests used in the learning procedure: 84
##   optimized:                            TRUE
```

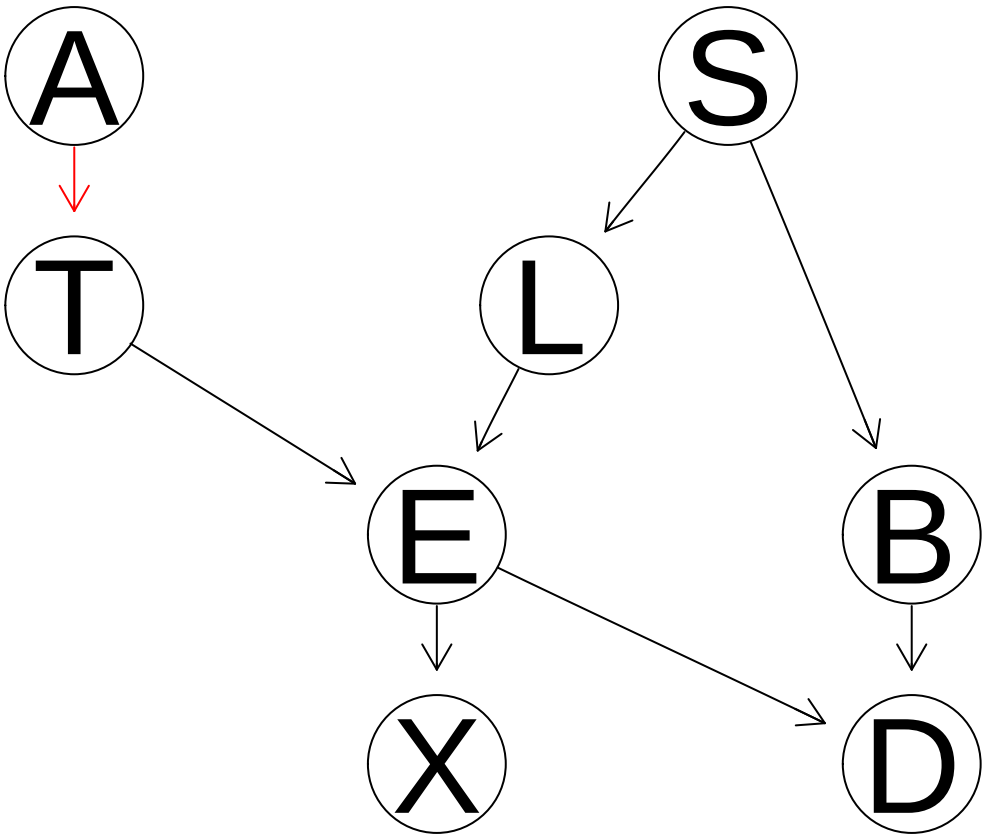
Given different initial graphs, or different iss using BDeu score, the learnt graphs are not equivalent(Have different adjacencies and unshielded colliders). It's because we choose diffierent initial hyperparameters and Hill-Climbing only promise to find the local minimum.

In the second instance,iss(imaginary sample size) is related to the prior $p(\theta_{x_i|Pa_i=j}|G)$, the latter follows Dirichlet distribution.As we know, the prior distribution will influence the posterior distribution.

(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.

```
asiaSample=sample(nrow(asia),floor(nrow(asia)*0.8))
trainSet=asia[asiaSample,]
testSet=asia[-asiaSample,]
trueDag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
dag=hc(trainSet,score='bde',iss=2)

graphviz.compare(dag,trueDag)
```


A
S
T
L
E
B
X
D



compute the posterior probability distribution of S for each case and classify it in the most likely class

```
bnInference=function(dag,trainSet,testSet,target,mb=0){
  fittedDag=bn.fit(dag,trainSet)
  grainDag=as.grain(fittedDag)
  compiledDag=compile(grainDag)

  if(mb==0){
    requiredNodes=setdiff(colnames(asia),target)
  }else{
    requiredNodes=mb(fittedDag,target)
  }

  predict=function(testInstence){
    evidence=setEvidence(compiledDag,nodes=requiredNodes,states=testInstence)
    return(ifelse(querygrain(evidence,nodes=target)[[1]][[1]]>0.5,"no","yes"))
  }

  res=sapply(as.data.frame(t(testSet[,requiredNodes])),predict)

  return(res)
}
```

```
# Fit the true dag
trueRes=bnInference(trueDag,trainSet,testSet,"S")
trueRes <- factor(trueRes, levels = c("yes", "no"))

# Fit the learnt dag
learntRes=bnInference(dag,trainSet,testSet,"S")
learntRes <- factor(learntRes, levels = c("yes", "no"))
```

Confusion Matrix

True Graph

```
# Confusion matrix
trueDagMatrix <- table(testSet$S, trueRes)
print(trueDagMatrix[,c(2,1)])
```

```
##      trueRes
##        no yes
##   no  323 164
##   yes 120 393
```

Own Result

```
# Confusion matrix
dagMatrix <- table(testSet$S, learntRes)
print(dagMatrix[,c(2,1)])
```

```
##      learntRes
##        no yes
##   no  323 164
##   yes 120 393
```

We have the same result!

(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.

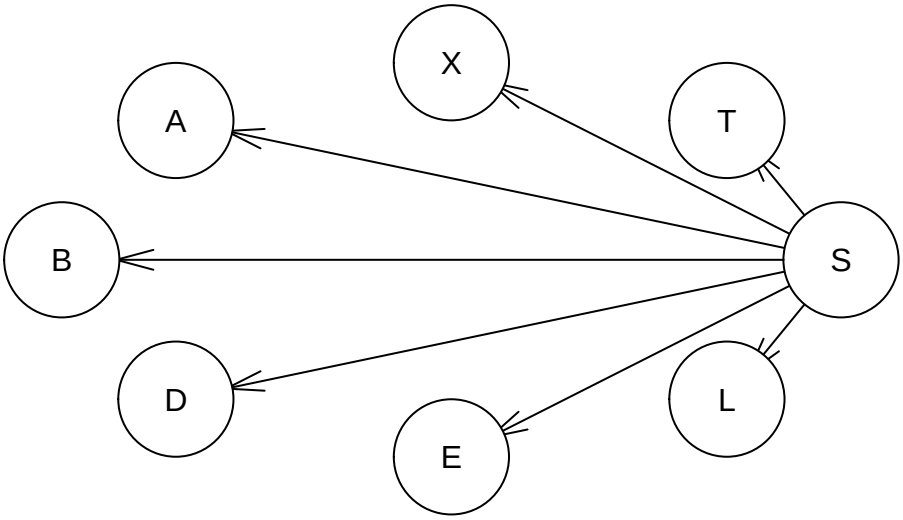
```
mbLearntRes=bnInference(dag,trainSet,testSet,"S",mb=1)
mbLearntRes <- factor(mbLearntRes, levels = c("yes", "no"))
mbDagMatrix <- table(testSet$S, mbLearntRes)
print(mbDagMatrix[,c(2,1)])
```

```
##      mbLearntRes
##        no yes
##   no  323 164
##   yes 120 393
```

The confusion matrix is the same as we get in (2), which tells that the markov blanket of S(node L and B) gives all the causal information of S.

(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.

```
naiveBN = model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")
plot(naiveBN)
```



Confusion Matrix

```r
# Fit the naive bayes dag
nbRes=bnInference(naiveBN,trainSet,testSet,"S")
nbRes <- factor(nbRes, levels = c("yes", "no"))

# Confusion matrix
nbMatrix <- table(testSet$S, nbRes)
print(nbMatrix[,c(2,1)])
```

```
##      nbRes
##        no yes
##   no  352 135
##   yes 182 331
```

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

Why the same result in exercise (2)?

In the true graph, the variables A and T are not independent.

For the DAG we learned in score based approach

$$\begin{aligned} P(S|Y \setminus S) &= \frac{P(Y)}{P(Y \setminus S)} \\ &= \frac{\prod_Y P(X_i|Pa_i)}{\sum_S \prod_{Y \setminus S} P(X_i|Pa_i)} \\ &= \frac{P(A)P(T)P(S)P(L|S)P(B|S)P(E|T,L)P(D|B,E)P(X|E)}{\tau_1(L)\tau_2(B)P(A)P(T)P(E|T,L)P(X|E)P(D|E,B)} \\ &= \frac{P(S)P(L|S)P(B|S)}{\tau_1(L)\tau_2(B)} \end{aligned}$$

where Y is the set of all the variable in the directed acyclic graph, τ_1,τ_2 is the marginal distribution of L,B, we get them using Variable Elimination.

For the true DAG

$$\begin{aligned} P(S|Y \setminus S) &= \frac{P(Y)}{P(Y \setminus S)} \\ &= \frac{\prod_Y P(X_i|Pa_i)}{\sum_S \prod_{Y \setminus S} P(X_i|Pa_i)} \\ &= \frac{P(A)P(T|A)P(S)P(L|S)P(B|S)P(E|T,L)P(D|B,E)P(X|E)}{\tau_1(L)\tau_2(B)P(A)P(T|A)P(E|T,L)P(X|E)P(D|E,B)} \\ &= \frac{P(S)P(L|S)P(B|S)}{\tau_1(L)\tau_2(B)} \end{aligned}$$

From the induction formula,the distribution of S given $Y \setminus S$ only relate to the variables B,L and S,Which is also the reason we get the same result in exercise 3.

Why the same result in exercise (3)?

$$\begin{aligned} P(S|U \setminus S) &= \frac{P(U)}{P(U \setminus S)} \\ &= \frac{P(S)P(L|S)P(B|S)}{\tau_1(L)\tau_2(B)} \end{aligned}$$

Where U is a subset of set X,and $U = \{S, L, B\}$ i.e.S and its markov blanket.

Why different result in exercise (4)?

In exercise 4 we use Naive Bayesan model to classify.

$$\begin{aligned} P(S|Y \setminus S) &= \frac{P(Y)}{P(Y \setminus S)} \\ &= \frac{\prod_Y P(X_i|Pa_i)}{\sum_S \prod_{Y \setminus S} P(X_i|Pa_i)} \\ &= \frac{P(S)P(A|S)P(T|S)P(L|S)P(B|S)P(E|S)P(D|S)P(X|S)}{\tau_1(A, P, L, B, E, D, X)} \end{aligned}$$