Lab1

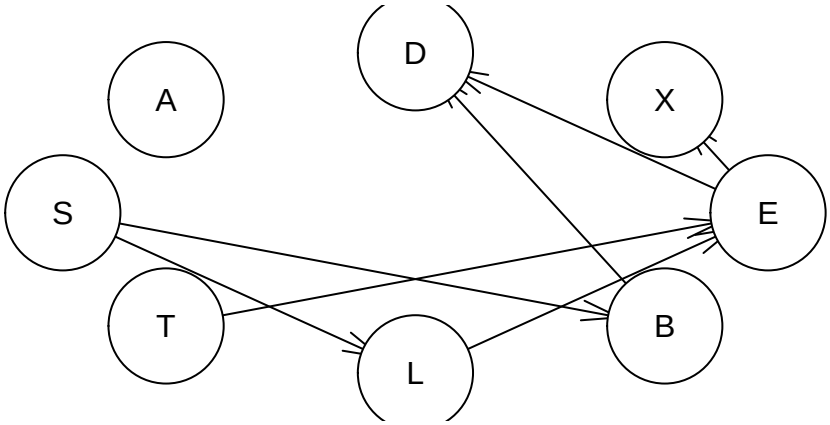
Milda Poceviciute

18 September 2018

Question 1

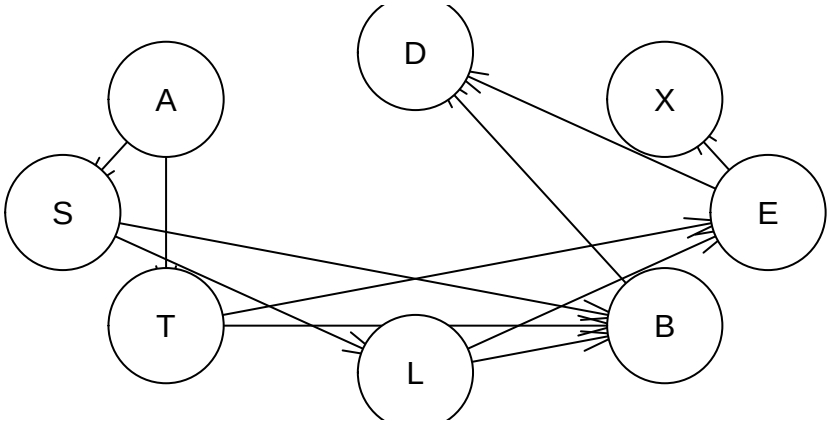
I use different approaches to generate Bayesian Networks with the hill-climbing algorithm. In some cases I specify starting graphs (i.e., generate a random and specific graphs), use different restart and score settings. The resulting structures are plotted below:

BN1: default settings (score=BIC)



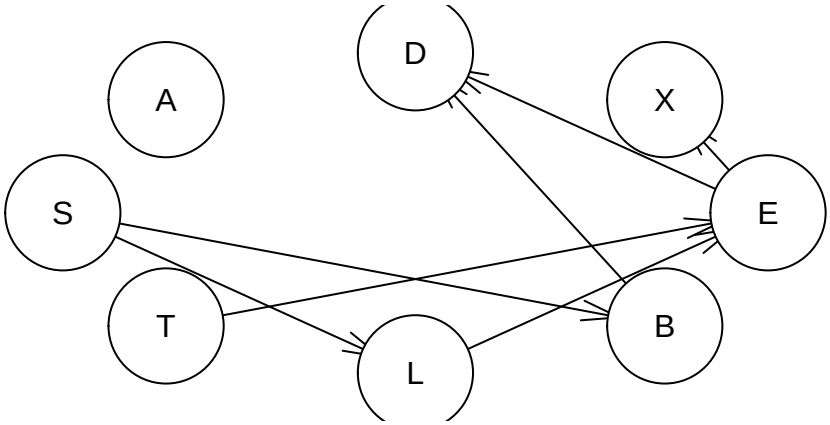
[A][S][T][L|S][B|S][E|T:L][X|E][D|B:E]

BN2: score = aic



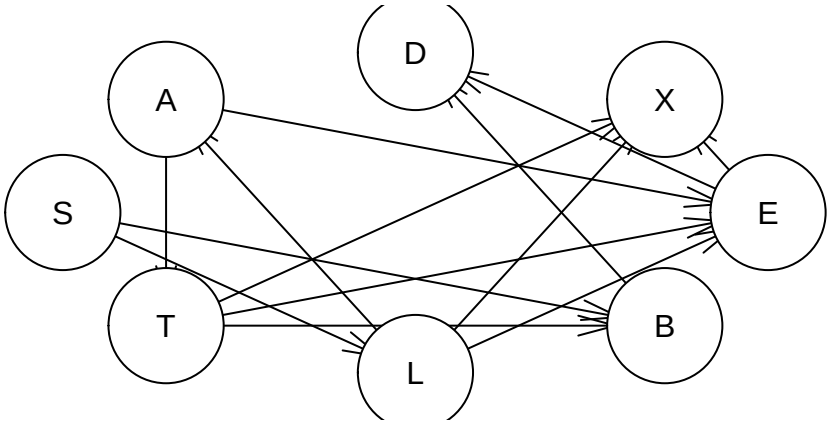
[A][S|A][T|A][L|S][B|S:T:L][E|T:L][X|E][D|B:E]

BN3: score=bde, restart=2



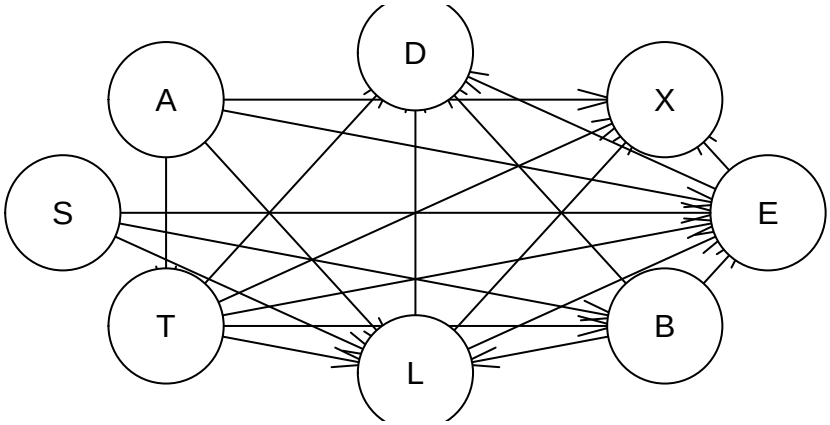
[A][S][T][L|S][B|S][E|T:L][X|E][D|B:E]

BN4: score=bde, restart=2, iss=10



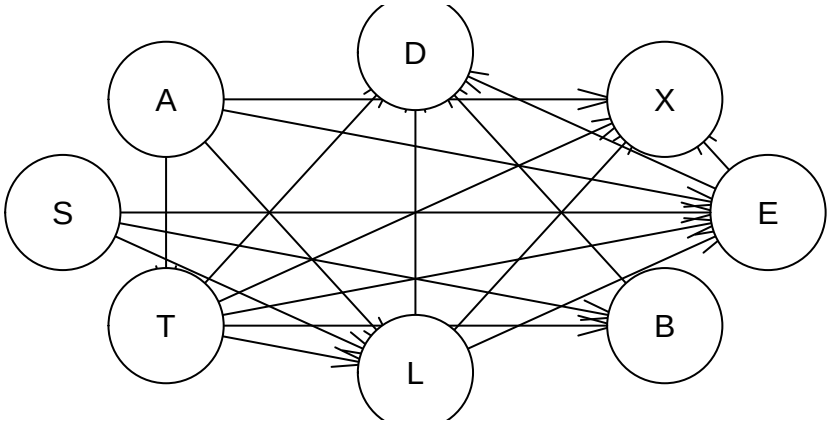
[S][L|S][A|L][T|A][B|S:T][E|A:T:L][X|T:L:E][D|B:E]

BN5: score=bde, restart=2, iss=100



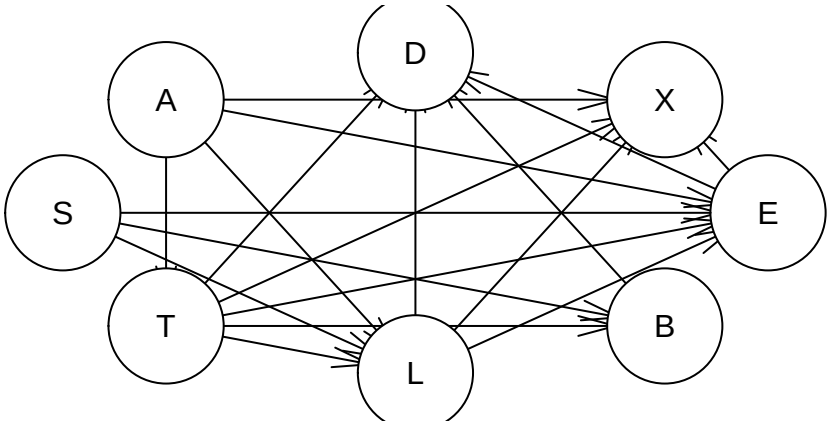
[A][S][T|A][B|S:T][L|A:S:T:B][E|A:S:T:L:B][X|A:T:L:E][D|T:L:B:E]

BN6: score=bde, restart=5, iss=50



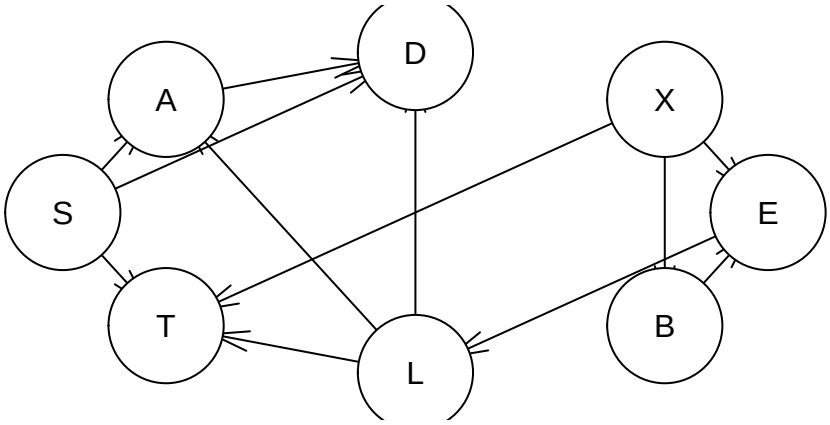
[A][S][T|A][L|A:S:T][B|S:T][E|A:S:T:L][X|A:T:L:E][D|T:L:B:E]

BN7: score=bde, restart=10, iss=50



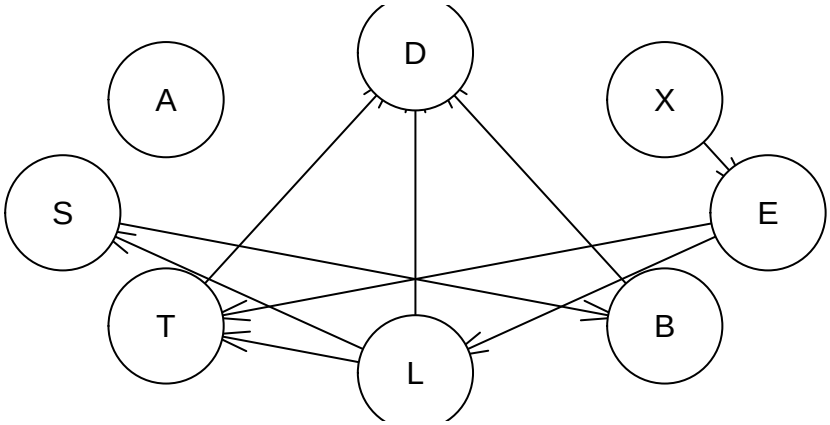
[A][S][T|A][L|A:S:T][B|S:T][E|A:S:T:L][X|A:T:L:E][D|T:L:B:E]

Generated Random Starting Graph



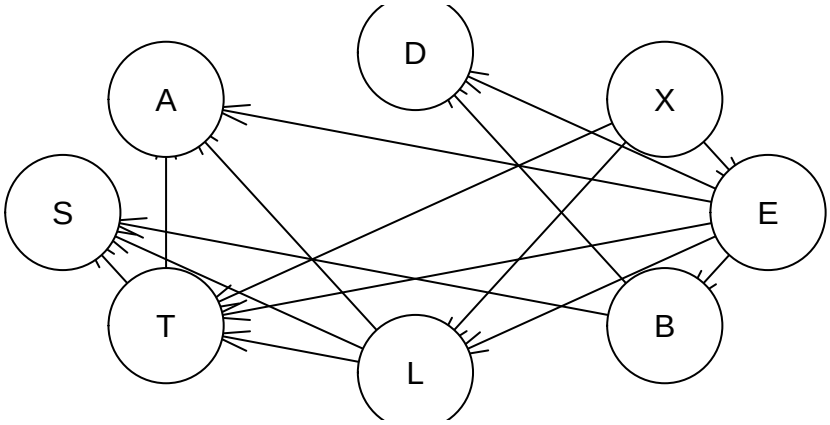
[S][X][B|X][E|B:X][L|E][A|S:L][T|S:L:X][D|A:S:L]

BN8: with the starting graph



[A][X][E|X][L|E][S|L][T|L:E][B|S][D|T:L:B]

BN9: BN8 + score = bde, restart = 5, iss=10



[X][E|X][L|E:X][B|E][T|L:E:X][D|B:E][A|T:L:E][S|T:L:B]

Hill climbing algorithm finds local optimal solution instead of the global. Hence, having different starting parameters can produce non-equivalent final structures. The direction of the arrows statistically has no difference, so the graphs that have same nodes connected but in opposite directions are still equivalent.

From the plots above, I conclude that BIC (Bayesian Information Criterion) and BDE (Bayesian Dirichlet

equivalent) produced equivalent graphs (BN1 and BN3), but all other structures are non-equivalent. Hence, it is clear that different runs of the hill-climbing algorithm produces non-equivalent Bayesian Network structures.

Question 2

In this question the data is split into training (80%) and testing (20%) data sets. Then a (local) optimal structure is determined using the Hill Climbing method. Afterwards the parameters are learnt using Maximum Likelihood algorithm, and the resulting graph is compiled (the steps of moralization and triangulation are done).

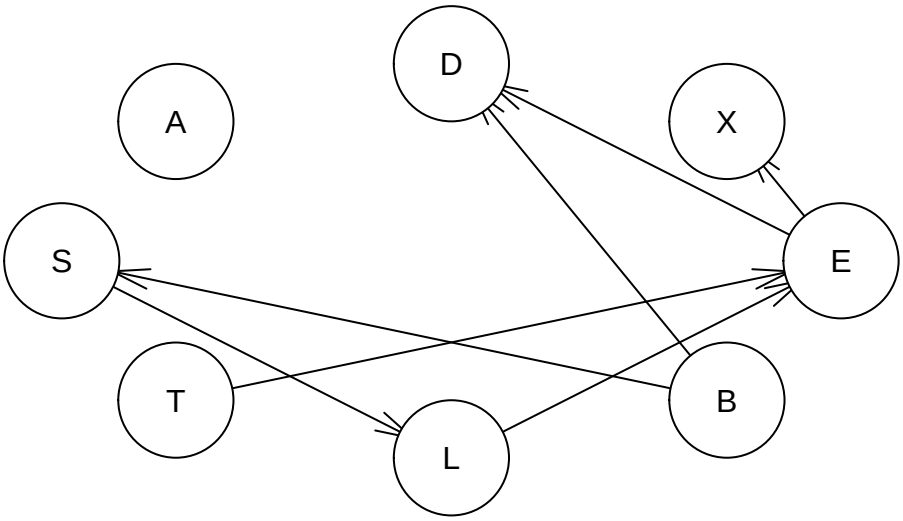
```
# Split data into 80% training and 20% testing data sets
N <- nrow(asia)
ind <- sample(1:N, size = 0.8*N)
train <- asia[ind,]
test <- asia[-ind,]
# Remove column S for predictions from testing data set
cnames2 <- colnames(test[-2])
new_test <- test[,-2]

# Generate a BN for training
train_learn <- hc(train, start = NULL, restart = 10)
summary(train_learn)
```

```
##          Length Class  Mode
## learning  8     -none- list
## nodes     8     -none- list
## arcs     14     -none- character
```

```
plot(train_learn)
title(train_learn, main="BN: default with restart = 10")
```

BN: default with restart = 10



[A][T][B][S|B][L|S][E|T:L][X|E][D|B:E]

```
# Learn parameters using ML
ml_fit <- bn.fit(train_learn, train, method = "mle")
ml_fit
```

```
##
##   Bayesian network parameters
##
##   Parameters of node A (multinomial distribution)
##
## Conditional probability table:
##       no     yes
## 0.99175 0.00825
##
##   Parameters of node S (multinomial distribution)
##
## Conditional probability table:
##
##      B
## S            no       yes
##   no  0.7124552 0.2879453
##   yes 0.2875448 0.7120547
##
##   Parameters of node T (multinomial distribution)
##
## Conditional probability table:
##       no     yes
```

```
## 0.99025 0.00975
##
##   Parameters of node L (multinomial distribution)
##
## Conditional probability table:
##
##      S
## L             no        yes
##   no  0.98686869 0.88069307
##   yes 0.01313131 0.11930693
##
##   Parameters of node B (multinomial distribution)
##
## Conditional probability table:
##       no     yes
## 0.48775 0.51225
##
##   Parameters of node E (multinomial distribution)
##
## Conditional probability table:
##
## , , L = no
##
##      T
## E     no yes
##   no   1   0
##   yes  0   1
##
## , , L = yes
##
##      T
## E     no yes
##   no   0   0
##   yes  1   1
##
##
##   Parameters of node X (multinomial distribution)
##
## Conditional probability table:
##
##      E
## X             no        yes
##   no  0.95726265 0.00660066
##   yes 0.04273735 0.99339934
##
##   Parameters of node D (multinomial distribution)
##
## Conditional probability table:
##
## , , E = no
##
##      B
## D            no       yes
##   no  0.9004894 0.2158235
```

```
##   yes 0.0995106 0.7841765
##
## , , E = yes
##
##      B
## D            no       yes
##   no  0.2857143 0.1308901
##   yes 0.7142857 0.8691099
```

```
# Convert into grain format
ml_grain <- as.grain(ml_fit)
ml_grain <- compile(ml_grain)
```

Afterwards I use the gRain package on compiled grain format graph to do predictions on test data set:

```
S_probs2 <- c()
S_probs2<- apply(new_test,1,function(a){
    Evid2 <- setEvidence(ml_grain,evidence=a)
    probs <- querygrain(Evid2)$S
    return(probs)
})

Predict2 <- c()
Predict2[which(S_probs2[1,]>0.5)] <- "no"
Predict2[which(S_probs2[2,]>=0.5)] <- "yes"

# Confussion Matrix
conf2 <- table(test$S, Predict2)
misscl2 <- missclassf(conf2)
```

```
## The confusion matrix is:
```

```
##      Predict2
##        no yes
##   no  345 160
##   yes 131 364
```

```
## Missclassification rate is 0.291.
```

The same predictions are made using the true graph (provided in the lab questions).

```
# The real graph
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]") #true graph
dag_fit <- bn.fit(dag, train, method = "mle")
dag_grain <- as.grain(dag_fit)
dag_compile <- compile(dag_grain)


S_probs <- c()
S_probs<- apply(new_test,1,function(a){
    Evid2 <- setEvidence(dag_compile,evidence=a)
    probs <- querygrain(Evid2)$S
    return(probs)
})

Predict_S <- c()
Predict_S[which(S_probs[1,]>0.5)] <- "no"
Predict_S[which(S_probs[2,]>=0.5)] <- "yes"
```

```r
# Confussion Matrix

confT <- table(test$S, Predict_S)
missclT <- missclassf(confT)
```

```
## The confusion matrix is:
```

```
##      Predict_S
##        no yes
##   no  345 160
##   yes 131 364
```

```
## Missclassification rate is 0.291.
```

The results of a BN learnt on 80% of data are exactly the same as the results of the true Asian BN.

Question 3

In this question I am using only the Markov blanket of node S to do the predictions. That is, S parents plus its children plus the parents of its children minus S itself.

```r
# Select only parents, children, and parents of children of S
mb_done <- mb(ml_fit,"S")
# Make a new testing data set only with the revelant nodes
new_test3 <- new_test[,mb_done]
# Predict the data
S_probs3 <- c()
S_probs3<- apply(new_test3,1,function(a){
    Evid2 <- setEvidence(dag_compile,evidence=a)
    probs <- querygrain(Evid2)$S
    return(probs)
})

# Classify the test data
Predict3 <- c()
Predict3[which(S_probs3[1,]>0.5)] <- "no"
Predict3[which(S_probs3[2,]>=0.5)] <- "yes"

# Confussion Matrix
conf3 <- table(test$S, Predict3)
missc13 <- missclassf(conf3)
```

```
## The confusion matrix is:
```

```
##      Predict3
##        no yes
##   no  345 160
##   yes 131 364
```

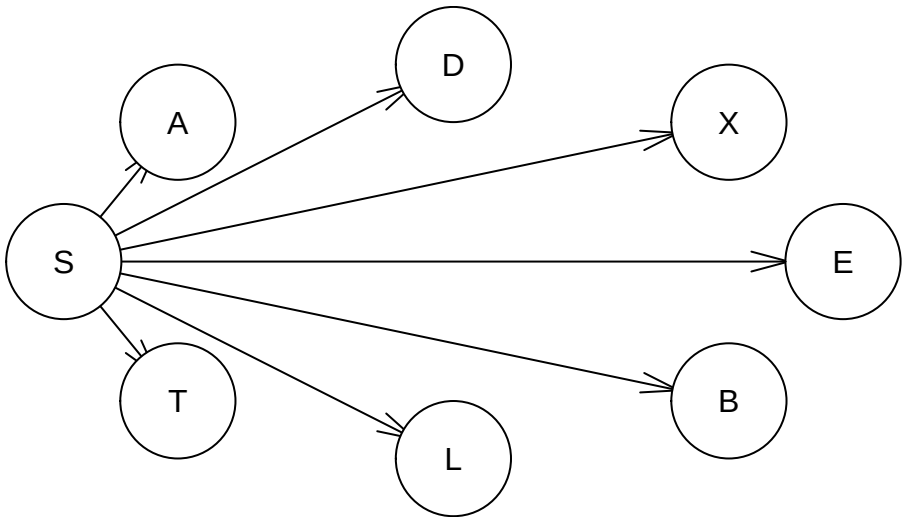
```
## Missclassification rate is 0.291.
```

BN that uses only Markov Blanket of S produced exactly the same accuracy of predictions as the previous two BNs.

Question 4

Naive Bayes classifier has a graph with all arrows pointing from S towards other nodes. This indicates that the other nodes are independent from each other given the class label. The corresponding graph is plotted below:

```
# Naive Bayes
cnames <- colnames(asia)
naive_graph = empty.graph(cnames)
arc.set = matrix(c("S", "A", "S","D", "S", "X", "S", "E", "S", "B", "S", "L", "S", "T"),
                            ncol = 2, byrow = TRUE,
                            dimnames = list(NULL, c("from", "to")))
arcs(naive_graph) = arc.set
plot(naive_graph)
```



The same procedure of learning the parameters, compiling the resulting structure and making predictions based on the posterior distribution of S is done below:

```
naive_fit <- bn.fit(naive_graph, train, method = "mle")
# Using grain package"
naive_grain <- as.grain(naive_fit)
naive_grain <- compile(naive_grain)


S_probs4 <- c()
S_probs4<- apply(new_test,1,function(a){
    Evid2 <- setEvidence(naive_grain,evidence=a)
```

```
    probs <- querygrain(Evid2)$S
    return(probs)
})

Predict4 <- c()
Predict4[which(S_probs4[1,]>0.5)] <- "no"
Predict4[which(S_probs4[2,]>=0.5)] <- "yes"

conf4 <- table(test$S, Predict4)
misscl4 <- missclassf(conf4)
```

```
## The confusion matrix is:
```

```
##      Predict4
##        no yes
##   no  370 135
##   yes 194 301
```

```
## Missclassification rate is 0.329.
```

Question 5 - Comparison of all BNs

```
## Missclassification rate by the BN based on the ture graph is 0.291.
```

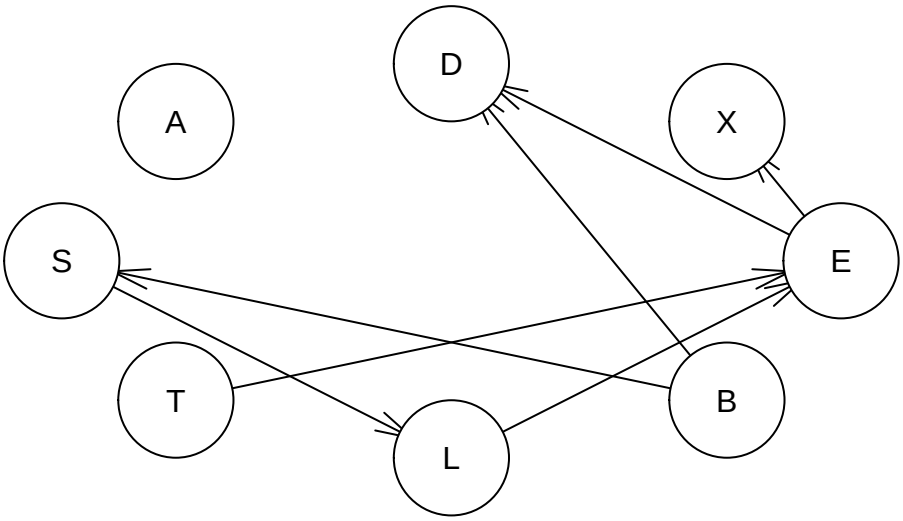
```
## Missclassification rate from question 2 is 0.291.
```

```
## Missclassification rate from question 3 is 0.291.
```

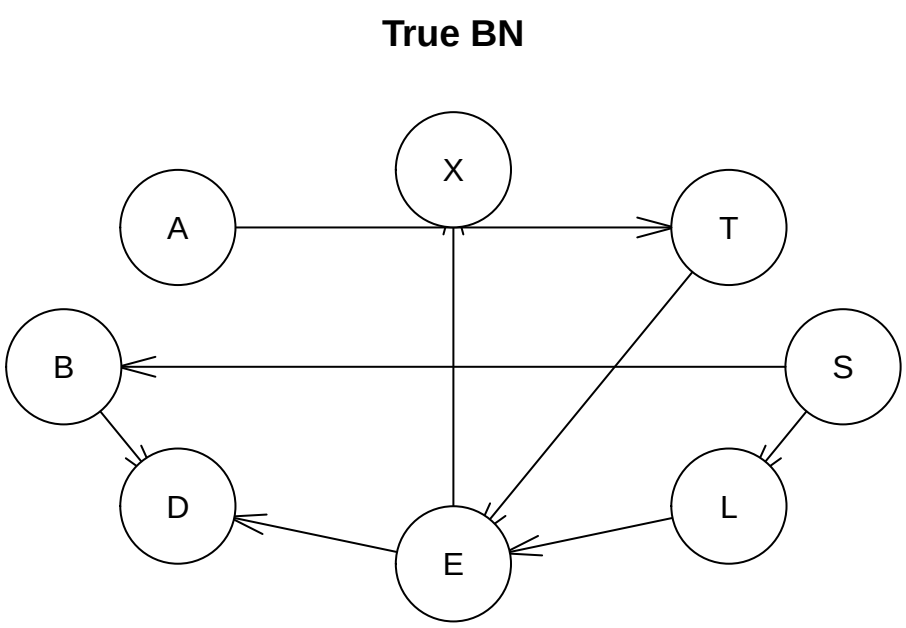
```
## Missclassification rate by Naive Bayes Classifier is 0.329.
```

As seen from above, the results of the first 3 BNs are exactly the same. To understand better why this is happening, we can look again at the corresponding BNs plots:

BN Question 2



[A][T][B][S|B][L|S][E|T:L][X|E][D|B:E]



[A][T][B][S|B][L|S][E|T:L][X|E][D|B:E]

We notice that both plots have the same structure around S: the node S is connected only to the nodes L and B (both BN have the same Markov Blanket of S). This indicates that the Markov Blanket of S is the same in the learnt BN from question 2 and the true BN. The predictions in the question 3 specifically uses only the Markov Blanket of S. Therefore, they all produce the same predictions.

The Naive Bayes Classifier produced slightly worse results. This might be due to the loss of some information as the relationshps between other nodes are ignored (the naive assumption of independence). A direct results of this is the fact that Markov Blanket of S in Naive Bayes is different from the Markov Blanket of the other BNs structures.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)

library(bnlearn)
library(gRain)
library(RBGL)
#source("https://bioconductor.org/biocLite.R")
#biocLite("RBGL")
library(gRbase)
data("asia")

missclassf <- function(table){
    result <- (table[1,2] + table[2,1])/(table[1,2] + table[2,1]+table[1,1] + table[2,2])
```

```
    return(result)
}
#par( mfrow = c(1,1) )
BN1<-hc(asia)
#summary(BN1)
plot(BN1)
title(BN1, main = "BN1: default settings (score=BIC)")
#cat("\n")
#arcs(BN1)
#vstructs(BN1)
#cpdag(BN1)

BN2 <- hc(asia, start = NULL, score = "aic")
plot(BN2)
title(BN2, main = "BN2: score = aic")
#cat("\n")
#cpdag(BN2)

BN3 <- hc(asia, start = NULL, score = "bde", restart = 2)
plot(BN3)
title(BN3, main = "BN3: score=bde, restart=2")
#cpdag(BN3)


#The imaginary or equivalent sample size of the prior distribution can be specified using the iss param
#it specifies the weight of the prior compared to the sample and thus controls the smoothness of the po

BN4 <- hc(asia, start = NULL, score = "bde", restart = 2, iss=10)
plot(BN4)
title(BN4, main = "BN4: score=bde, restart=2, iss=10")
#cpdag(BN3)

BN5 <- hc(asia, start = NULL, score = "bde", restart = 2, iss=100)
plot(BN5)
title(BN5, main = "BN5: score=bde, restart=2, iss=100")
#cpdag(BN4)

BN6 <- hc(asia, start = NULL, score = "bde", restart = 5, iss=50)
plot(BN6)
title(BN6, main = "BN6: score=bde, restart=5, iss=50")
#cpdag(BN5)

BN7 <- hc(asia, start = NULL, score = "bde", restart = 10, iss=50)
plot(BN7)
title(BN7, main = "BN7: score=bde, restart=10, iss=50")
#cpdag(BN6)

cnames <- colnames(asia)
startg <- random.graph(nodes=cnames, num=1, method="melancon")
plot(startg)
title(startg, main = "Generated Random Starting Graph")

BN8 <- hc(asia, start = startg)
```

```r
plot(BN8)
title(BN8, main = "BN8: with the starting graph")

BN9 <- hc(asia, start = startg, score = "bde", restart = 5, iss=10)
plot(BN9)
title(BN9, main = "BN9: BN8 + score = bde, restart = 5, iss=10 ")
# Split data into 80% training and 20% testing data sets
N <- nrow(asia)
ind <- sample(1:N, size = 0.8*N)
train <- asia[ind,]
test <- asia[-ind,]
# Remove column S for predictions from testing data set
cnames2 <- colnames(test[-2])
new_test <- test[,-2]

# Generate a BN for training
train_learn <- hc(train, start = NULL, restart = 10)
summary(train_learn)
plot(train_learn)
title(train_learn, main="BN: default with restart = 10")

# Learn parameters using ML
ml_fit <- bn.fit(train_learn, train, method = "mle")
ml_fit

# Convert into grain format
ml_grain <- as.grain(ml_fit)
ml_grain <- compile(ml_grain)
S_probs2 <- c()
S_probs2<- apply(new_test,1,function(a){
    Evid2 <- setEvidence(ml_grain,evidence=a)
    probs <- querygrain(Evid2)$S
    return(probs)
})

Predict2 <- c()
Predict2[which(S_probs2[1,]>0.5)] <- "no"
Predict2[which(S_probs2[2,]>=0.5)] <- "yes"

# Confussion Matrix
conf2 <- table(test$S, Predict2)
misscl2 <- missclassf(conf2)

cat("The confusion matrix is:")
cat("\n")
cat("\n")
conf2
cat("\n")
cat(paste0("Missclassification rate is ",misscl2,"."))
cat("\n")

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

```
dag_fit <- bn.fit(dag, train, method = "mle")
dag_grain <- as.grain(dag_fit)
dag_compile <- compile(dag_grain)


S_probs <- c()
S_probs<- apply(new_test,1,function(a){
    Evid2 <- setEvidence(dag_compile,evidence=a)
    probs <- querygrain(Evid2)$S
    return(probs)
})

Predict_S <- c()
Predict_S[which(S_probs[1,]>0.5)] <- "no"
Predict_S[which(S_probs[2,]>=0.5)] <- "yes"

# Confussion Matrix

confT <- table(test$S, Predict_S)
missclT <- missclassf(confT)


cat("The confusion matrix is:")
cat("\n")
cat("\n")
confT
cat("\n")
cat(paste0("Missclassification rate is ",missclT,"."))
cat("\n")

# Select only parents, children, and parents of children of S
mb_done <- mb(ml_fit,"S")
# Make a new testing data set only with the revelant nodes
new_test3 <- new_test[,mb_done]
# Predict the data
S_probs3 <- c()
S_probs3<- apply(new_test3,1,function(a){
    Evid2 <- setEvidence(dag_compile,evidence=a)
    probs <- querygrain(Evid2)$S
    return(probs)
})

# Classify the test data
Predict3 <- c()
Predict3[which(S_probs3[1,]>0.5)] <- "no"
Predict3[which(S_probs3[2,]>=0.5)] <- "yes"

# Confussion Matrix
conf3 <- table(test$S, Predict3)
missc13 <- missclassf(conf3)

cat("The confusion matrix is:")
cat("\n")
```

```
cat("\n")
conf3
cat("\n")
cat(paste0("Missclassification rate is ",misscl3,"."))
cat("\n")

# Naive Bayes
cnames <- colnames(asia)
naive_graph = empty.graph(cnames)
arc.set = matrix(c("S", "A", "S","D", "S", "X", "S", "E", "S", "B", "S", "L", "S", "T"),
                            ncol = 2, byrow = TRUE,
                            dimnames = list(NULL, c("from", "to")))
arcs(naive_graph) = arc.set
plot(naive_graph)
naive_fit <- bn.fit(naive_graph, train, method = "mle")
# Using grain package"
naive_grain <- as.grain(naive_fit)
naive_grain <- compile(naive_grain)


S_probs4 <- c()
S_probs4<- apply(new_test,1,function(a){
    Evid2 <- setEvidence(naive_grain,evidence=a)
    probs <- querygrain(Evid2)$S
    return(probs)
})

Predict4 <- c()
Predict4[which(S_probs4[1,]>0.5)] <- "no"
Predict4[which(S_probs4[2,]>=0.5)] <- "yes"

conf4 <- table(test$S, Predict4)
misscl4 <- missclassf(conf4)

cat("The confusion matrix is:")
cat("\n")
cat("\n")
conf4
cat("\n")
cat(paste0("Missclassification rate is ",misscl4,"."))
cat("\n")


cat(paste0("Missclassification rate by the BN based on the ture graph is ",missclT,"."))
cat("\n")
cat(paste0("Missclassification rate from question 2 is ",misscl2,"."))
cat("\n")
cat(paste0("Missclassification rate from question 3 is ",misscl3,"."))
cat("\n")
cat(paste0("Missclassification rate by Naive Bayes Classifier is ",misscl4,"."))
cat("\n")
```

```
plot(train_learn)
title(train_learn, main="BN Question 2")
plot(dag)
title(train_learn, main="True BN")
```