# Advanced Machine Learning

Hugo M (hugmo418), Emanuel H (emahe385), Viktor B (vikbe291)

2024-09-14

## Lab 1

### Statement of Contribution

All codes and explanation were discussed by all the members in the group.

- Question 1: Code by Hugo, Emanuel and Viktor,
- Question 2: Code by Viktor,
- Question 3: Code by Hugo,
- Question 4: Code by Emanuel,
- Question 5: Interpretation of discussion together.

### Question 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.

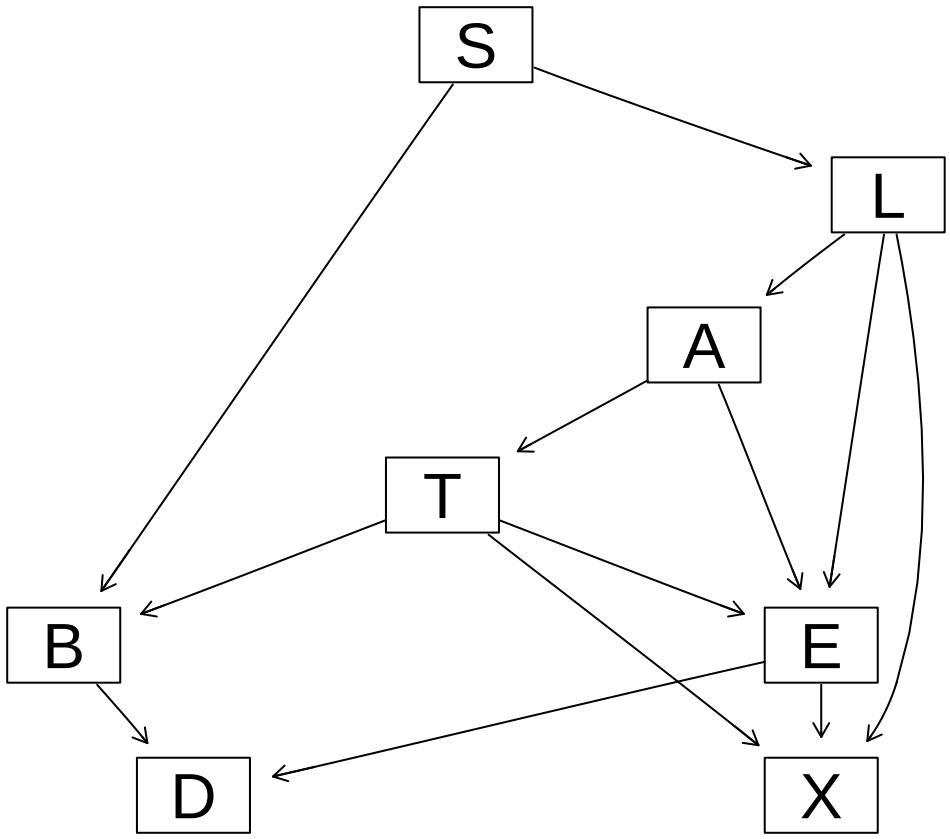
Hint: Check the function hc in the bnlearn package. Note that you can specify the initial structure, the number of random restarts, the score, and the equivalent sample size (a.k.a imaginary sample size) in the BDeu score. You may want to use these options to answer the question. You may also want to use the functions plot, arcs, vstructs, cpdag and all.equal.

```
data("asia")
```

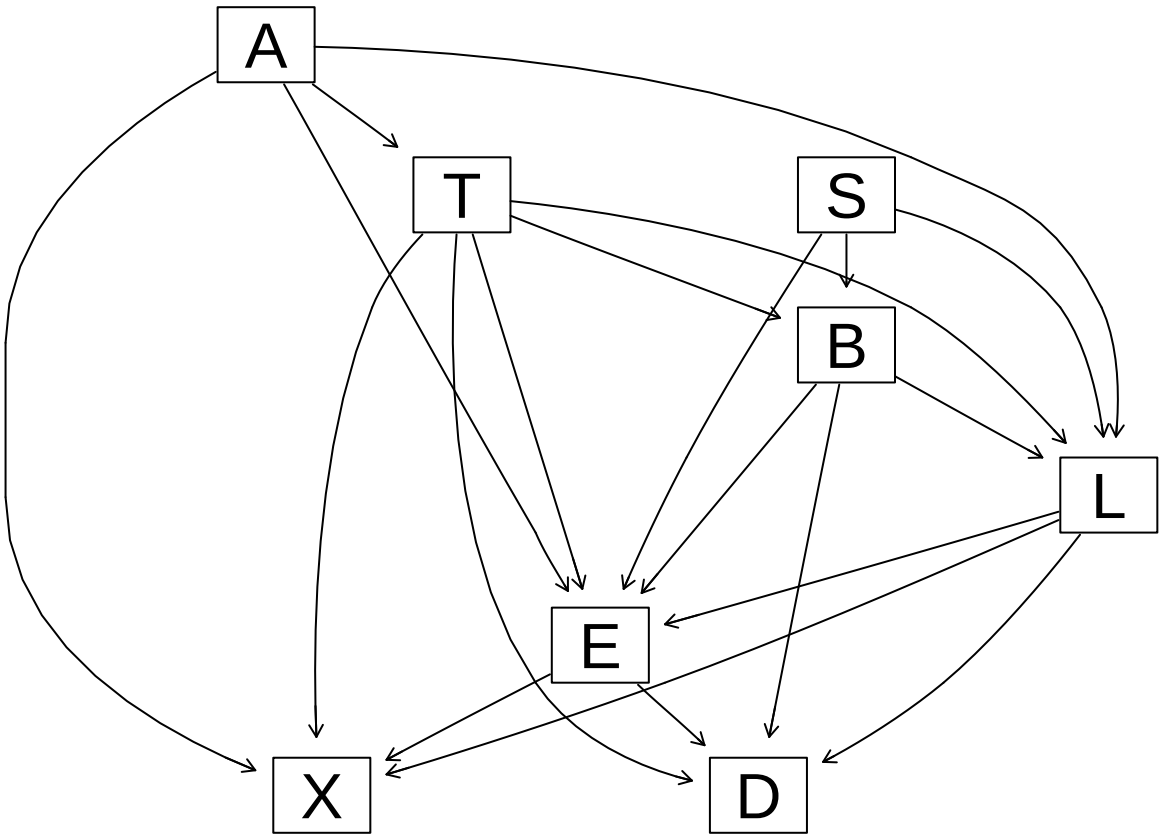
For this question, we found three different ways of obtaining different structures:

#### Method 1: changing the imaginary sample size

```
set.seed(123)

#Learning a first graph with Hill Climb with imaginary sample size = 10
graph_a = hc(asia, restart = 10, score = "bde", iss = 10)
#Learning a second graph with Hill Climb with imaginary sample size = 100
graph_b = hc(asia, restart = 10, score = "bde", iss = 100)

#Plotting the graphs
par(mar=c(0.5,0.5,0.5,0.5))
graphviz.plot(graph_a)
```

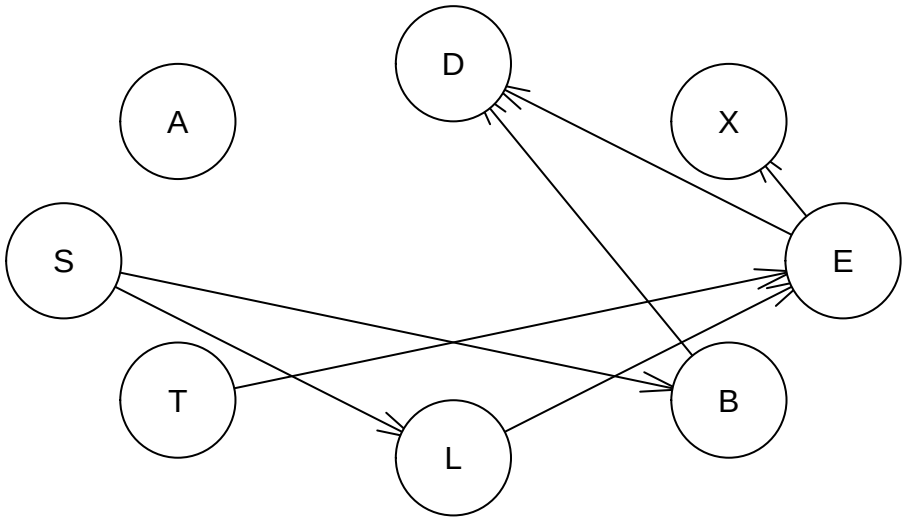



```
graphviz.plot(graph_b)
```

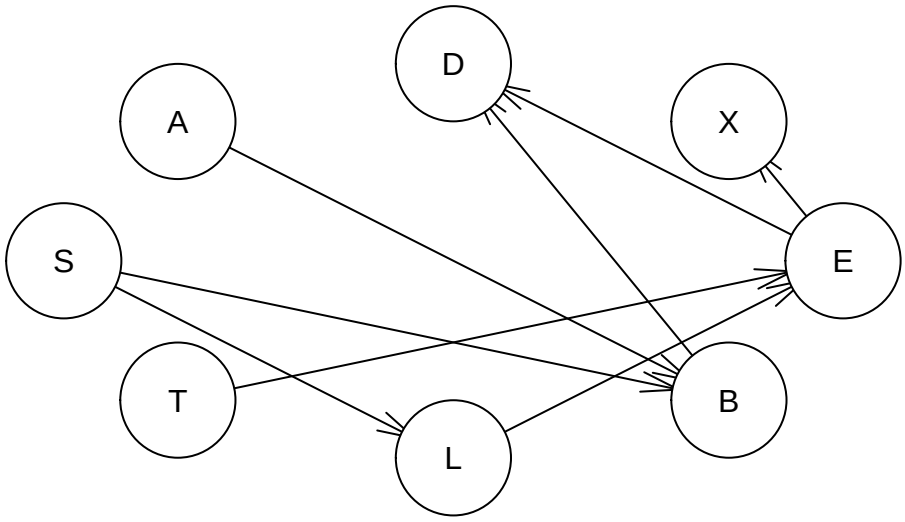



**Method 2: Specifying initial structure**

```
dag <- hc(asia)

plot(dag)
```




```
# Specify some initial structure
forced_arcs <- data.frame(from=c("A", "S"),
                          to=c("B", "B"))

dag_forced <- hc(asia, whitelist = forced_arcs)

plot(dag_forced)
```




```
all.equal(dag, dag_forced)
```

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

**Method 3: changing the score function**

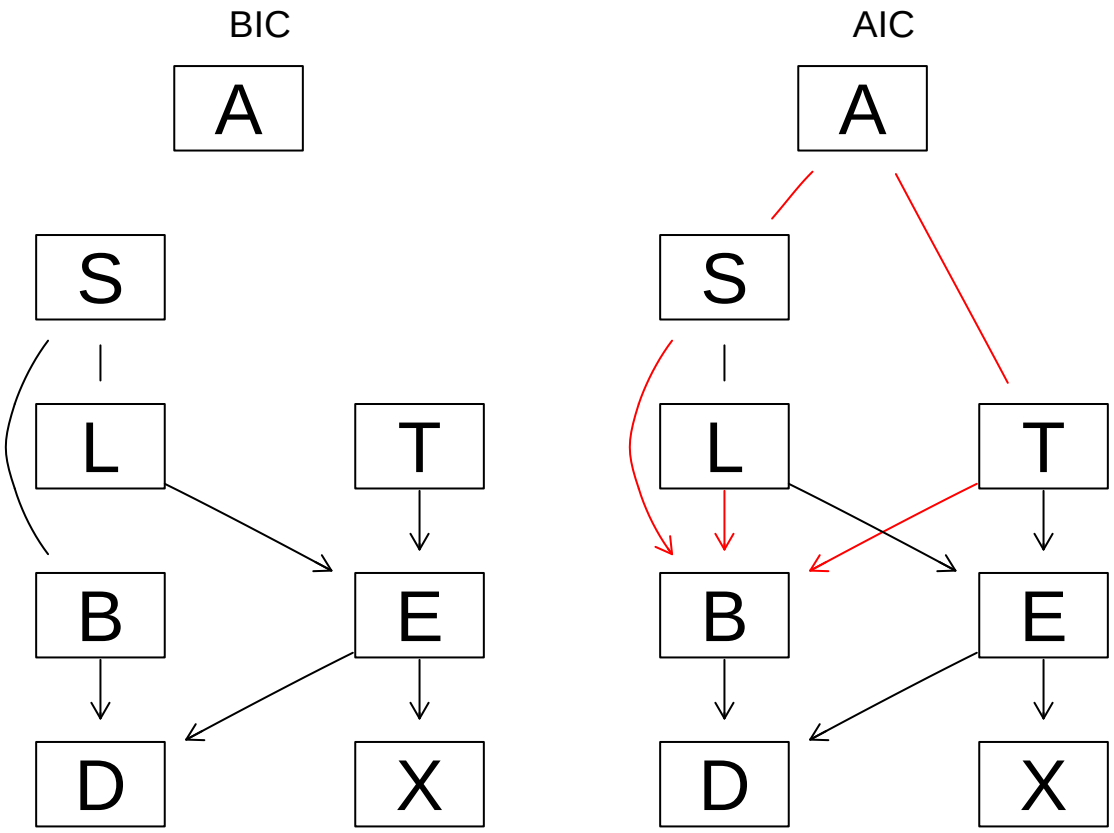
```
# Hill-climb
hc1 <- hc(asia, restart=0, score="bic")
vs.hc1 <- vstructs(hc1)

# And again but change score (this will generate another graph)

hc2 <- hc(asia, restart=0, score="aic")
vs.hc2 <- vstructs(hc2)

vs.hc1
```

```
##      X   Z   Y
## [1,] "T" "E" "L"
## [2,] "B" "D" "E"
```

```
vs.hc2
```

```
##      X   Z   Y
## [1,] "S" "B" "T"
## [2,] "T" "B" "L"
## [3,] "T" "E" "L"
## [4,] "B" "D" "E"
```

```
# Convert to CPDAGs to standardize representation
hc1 <- cpdag(hc1)
hc2 <- cpdag(hc2)

# Visualize
par(mfrow = c(1, 2))
graphviz.compare(hc1, hc2, main=c("BIC", "AIC"))
```




```
all.equal(hc1, hc2)
```

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

All three methods yields non-equivalent BN structures, as we can see with the function “all.equal”.

## Question 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]").

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
set.seed(123)  # For consistency
split <- sample(1:nrow(asia), size = 0.8 * nrow(asia))

train_data <- asia[split, ]
test_data <- asia[-split, ]

# Learn the structure of the BN from the training data
bn_structure <- hc(train_data, score="bic")
# plot(bn_structure)  # Visualize the learned structure

# Fit the parameters of the BN (Learn the parameters)
fitted_bn <- bn.fit(bn_structure, data = train_data)

# Convert the fitted bnlearn object to a gRain object
grain_bn <- as.grain(fitted_bn)

# Initialize vectors to store predictions and true values
predictions <- c()
true_values <- test_data$S  # Actual values of S

# Perform inference for each case in the test set
for (i in 1:nrow(test_data)) {
  # Set evidence for the test case (all variables except S) (S is 2:nd column)
  evidence <- setEvidence(grain_bn, nodes = colnames(test_data)[-2],
                          states = as.character(unlist(test_data[i, -2])))

  # Query the posterior distribution of S
  posterior <- querygrain(evidence, nodes = "S")

  # Predict the most probable class for S
  predicted_class <- names(which.max(posterior$S))

  # Store the predicted class
  predictions <- c(predictions, predicted_class)
}
```

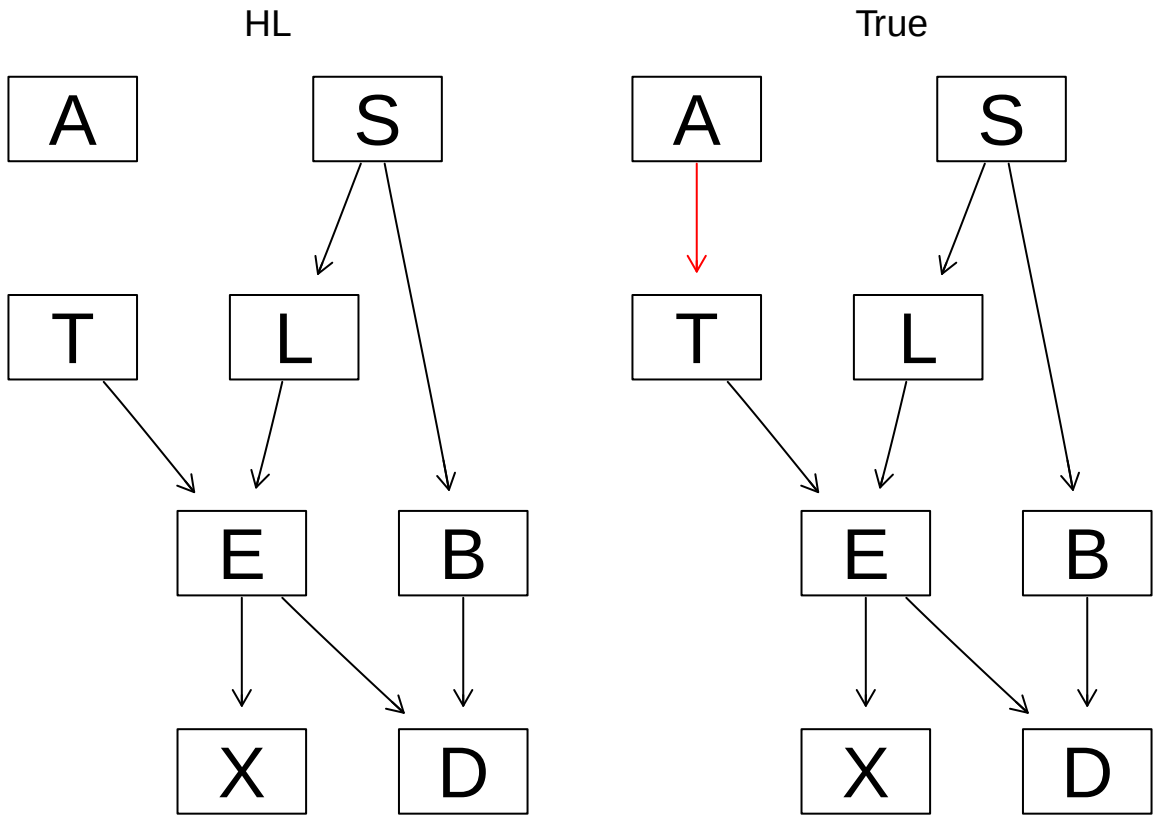
```
# Create the confusion matrix
confusion_matrix <- table(True = true_values, Predicted = predictions)
print("HL")
```

```
## [1] "HL"
```

```
print(confusion_matrix)
```

```
##      Predicted
## True   no yes
##   no  359 141
##   yes 116 384
```

```
# True Asia BN structure
dag <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

par(mfrow = c(1, 2))
graphviz.compare(bn_structure, dag, main=c("HL", "True") )
```




```
# Fit the true BN with the training data
true_fitted_bn <- bn.fit(dag, data = train_data)

# Convert to gRain object
```

```r
true_grain_bn <- as.grain(true_fitted_bn)

# Initialize vectors to store predictions for the true BN
true_bn_predictions <- c()

# Perform inference for the true BN
for (i in 1:nrow(test_data)) {
  evidence <- setEvidence(true_grain_bn, nodes = colnames(test_data)[-2], states = as.character(unlist(t
  posterior <- querygrain(evidence, nodes = "S")
  predicted_class <- names(which.max(posterior$S))
  true_bn_predictions <- c(true_bn_predictions, predicted_class)
}

# Compute the confusion matrix for the true BN
true_bn_confusion_matrix <- table(True = true_values, Predicted = true_bn_predictions)
print("True")
```

```
## [1] "True"
```

```r
print(true_bn_confusion_matrix)
```

```
##      Predicted
## True   no yes
##   no  359 141
##   yes 116 384
```

In the comparison of the two graphs, we can see that the only difference is an edge between A and T, which does not affect the conditional probabilities of S. Hence we observe that the two structures perform the same in the confusion matrices and in the predictions.

## 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. Hint: You may want to use the function mb from the bnlearn package.

```r
n = nrow(asia)
train = asia[1:round(0.8*n),]
test = asia[(round(0.8*n)+1):n,]

#Learn the Structure
g = hc(train)
#Learn the parameters
fitted = bn.fit(g, train, method = "mle")
#Extract the Markov blanket
Smb = mb(fitted, "S")
print(Smb)
```

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

```r
#convert bn.fit object to grain object:
mbbn = as.grain(fitted)
mbbn = compile(mbbn)
summary(mbbn)
```

```
## Independence network: Compiled: TRUE Propagated: FALSE
##  Nodes : Named chr [1:8] "A" "S" "T" "L" "B" "E" "X" "D"
##  - attr(*, "names")= chr [1:8] "A" "S" "T" "L" ...
##  Number of cliques:                 6
##  Maximal clique size:               3
##  Maximal state space in cliques:    8
```

```r
#querygrain : obtain the conditional distribution of a set of variables - possibly (and typically) give

#Classify the remaining 20% of the Asia dataset
preds = rep(NaN, dim(test)[1])

for(i in 1:dim(test)[1]){
  nodes = Smb #nodes of interest are the ones in my S-Markov blanket
  states = as.vector(unlist(c(test[i,Smb])))
  evidences = setEvidence(mbbn, nodes, states)
  prob = querygrain(evidences, nodes = "S")
  preds[i] = names(which.max(c(prob$S[1],prob$S[2])))
}
#Report the confusion matrix
preds = as.factor(preds)
confmat = confusionMatrix(preds, test$S)
print(confmat$table)
```

```
##           Reference
## Prediction  no yes
##        no  358 120
##        yes 147 375
```

```r
print(confmat$overall[1])
```

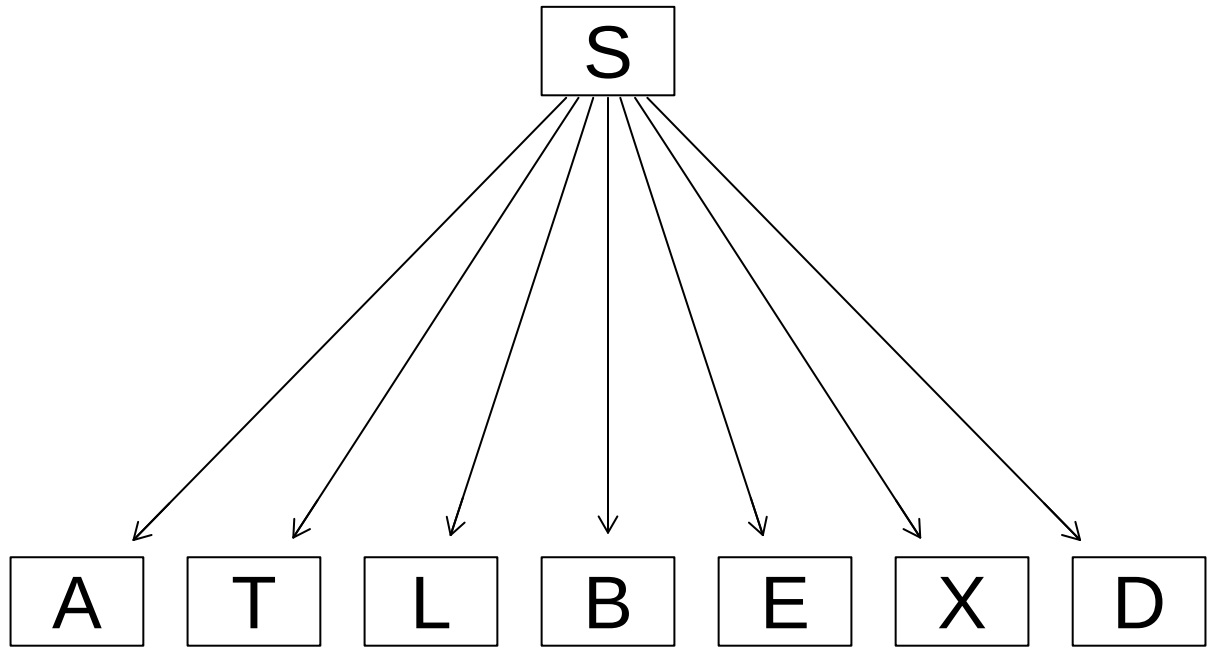
```
## Accuracy
##    0.733
```

## 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.

```r
# Calculate cutoff point
cutoff <- round(nrow(asia) * 0.80)

# Split
```

```r
training_data <- asia[1:cutoff, ]
test_data <- asia[(cutoff + 1):nrow(asia), ]

# Create Bayesian Network
specific_letters <- c("A", "S", "T", "L", "B", "E", "X", "D")
empty_graph <- empty.graph(specific_letters)

# Model as from every node to S
arc.set = 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(empty_graph) = arc.set

#Plot

graphviz.plot(empty_graph)
```




```r
# Rename for clarity
naive_bayes_graph <- empty_graph # Since graph is no longer empty

# Learn parameters
naive_bayes_model <- bn.fit(naive_bayes_graph, training_data)

# Used variables in confusion matrix
true_pos <- 0
```

```r
false_pos <- 0
true_neg <- 0
false_neg <- 0

# Calculate conditional probabilities
for (row in 1:nrow(test_data)){

  # Get the evidence
  A_val <- as.vector(test_data[row, "A"])
  T_val <- as.vector(test_data[row, "T"])
  L_val <- as.vector(test_data[row, "L"])
  B_val <- as.vector(test_data[row, "B"])
  E_val <- as.vector(test_data[row, "E"])
  X_val <- as.vector(test_data[row, "X"])
  D_val <- as.vector(test_data[row, "D"])

  # Calculate conditional probability for learned structure
  temp_cond_calc <- cpquery(naive_bayes_model, event = (S == "yes"),
                            evidence = (A == A_val & T == T_val & L == L_val &
                                          B == B_val & E == E_val & X == X_val & D == D_val))

  # Determine if it is true positive, false positive,
  #                    true negative or false negative
  if (temp_cond_calc >= 0.5){
    if (test_data[row, "S"] == "yes"){
      # Is a true positive
      true_pos <- true_pos + 1
    }
    else{
      false_pos <- false_pos + 1
    }
  }
  else{
    if (test_data[row, "S"] == "yes"){
      # False negative
      false_neg <- false_neg + 1
    }
    else{
      true_neg <- true_neg + 1
    }
  }
}

cat("TP: ", true_pos, "\n")
```

```
## TP:  305
```

```r
cat("FP: ", false_pos, "\n")
```

```
## FP:  118
```

```r
cat("FN: ", false_neg, "\n")
```

```
## FN:  190
```

```r
cat("TN: ", true_neg, "\n")
```

```
## TN:  387
```

## Question 5

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

Between question 2 and question 3, we change the evidence nodes from all the nodes to only the nodes in the Markov blanket of S. Since the Markov blanket contains all the useful features, it means that some features in question 2 were already not providing information, hence removing them does not change the results in the confusion matrix and the predictions.

Between question 2 and question 4, we change the structure from a learned Bayesian Network to a naive Bayesian network, which assumes conditional independence between the features, given the target class (S). Since the naive Bayesian assumes a certain network structure, it looses some information which explains the difference in the results.