Lab_1

David Gumpert Harryson

9/14/2020

Libraries

```
library(bnlearn)
library(gRain)
```

User defined functions

Prediction function for the BN given a junction tree (BN structure), training or test data, the observed variables and the ones to predict

```
BN_predict <- function(grain_obj, data, obs_var, pred_var) {
  predictions <- rep(NULL,nrow(data))
  # Compiling the BN and establishing a junction tree (through the
  # Laurentzen-Spiegelhalter algorithm). Sets the clique potentials and prepares
  # the data for a faster algorithm execution
  BN_compiled <- compile(grain_obj)
  for (i in 1:nrow(data)) {
    X <- NULL
    for (j in obs_var) {
      if(data[i,j] == "yes") {
        X[j] <- "yes"
      } else {
        X[j] <- "no"
      }
    }
    # Updating evidence in the compiled BN for observation i
    ev <- setEvidence(object = BN_compiled,
                      nodes = obs_var,
                      states = X)
    # Calculating the posterior distribution on the pred_var (the node to
    # predict). First calculating the probability distribution fit, then assign
    # "yes" if > 0.5
    dist_fit <- querygrain(object = ev,
                           nodes = pred_var)
    if(dist_fit$S["yes"] > 0.5) {
      predictions[i] <- "yes"
    } else {
      predictions[i] <- "no"
    }
```

```
  }
  return (predictions)
}
```

Comparison function of two BN:s, checking similarity

```
compareBN <- function(BN1, BN2) {
  plot(BN1, main="BN 1")
  plot(BN2, main="BN 2")

  print(vstructs(BN1))
  print(vstructs(BN2))

  cpdag1 <- cpdag(BN1)
  cpdag2 <- cpdag(BN2)
  plot(cpdag1, main="cpdag BN1")
  plot(cpdag2, main="cpdag BN2")

  print(all.equal(BN1,BN2))
  print(all.equal(cpdag1,cpdag2))
}
```

Missclassification identification function where the in parameters are the predictions and the true valuesand out parameter is the percentage of missclassified dataobjects

```
missclass_check <- function(pred, true_val ) {
  return (1-sum(diag(table(pred, true_val)))/length(pred))
}
```

Task 1

Show that multiple runs of the hill-climbing algorithm can return non-equivalent Bayesian network (BN) structures. Explain why this happens.

To show that multiple runs of the HC-algorithm can return non-equivalent Bayesian network structures I have decided to run the algorithm for two BN structures with different starting criterias such as: iss values, scoring methods, random restarts, and initial structures. The results are presented one by one below.

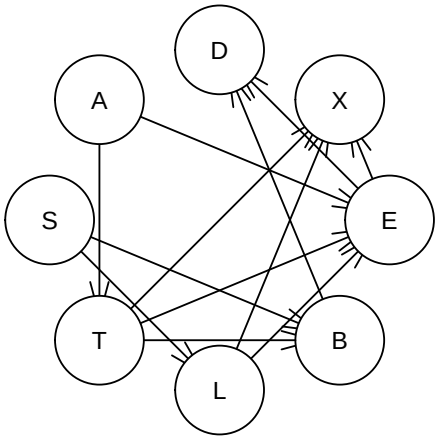
Different iss values

```
BN1 <- hc(data, iss = 5, score = "bde")
BN2 <- hc(data, iss = 2, score = "bde")
par(mfrow = c(2,2))
compareBN(BN1, BN2)
```

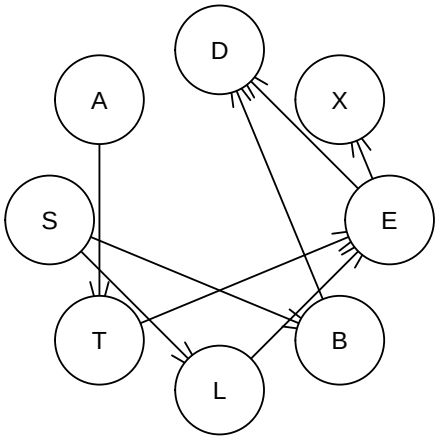
```
##      X   Z   Y
## [1,] "S" "B" "T"
## [2,] "A" "E" "T"
## [3,] "A" "E" "L"
## [4,] "T" "E" "L"
```

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

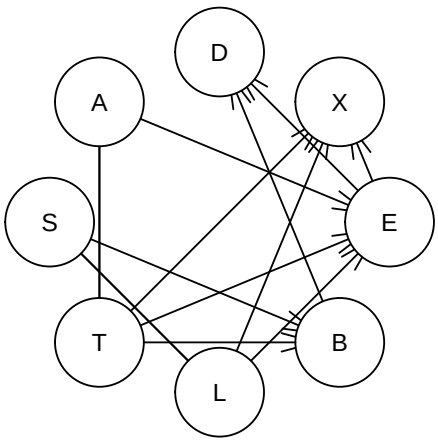
BN 1



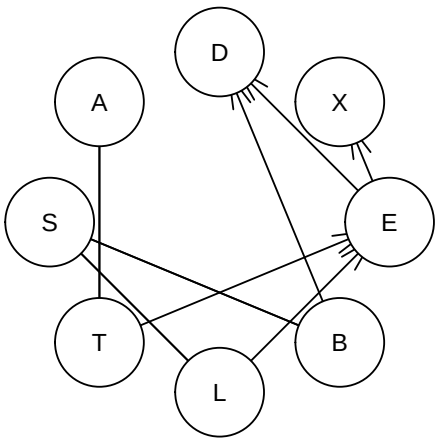
BN 2



cpdag BN1



cpdag BN2




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

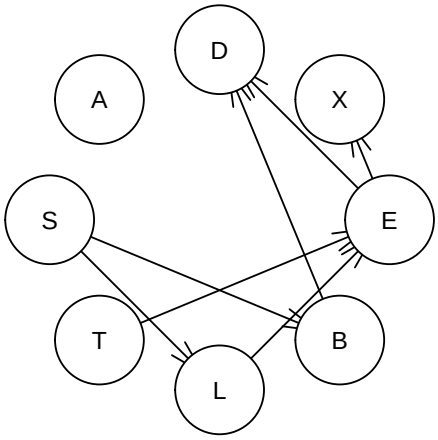
Different scoring methods

```
BN1 <- hc(data, score = "bic")
BN2 <- hc(data, score = "bde")
par(mfrow = c(2,2))
compareBN(BN1, BN2)
```

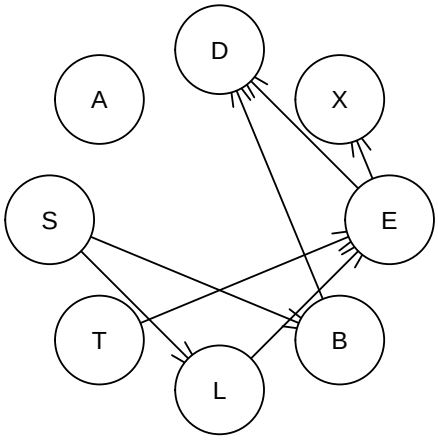
bic vs bde

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

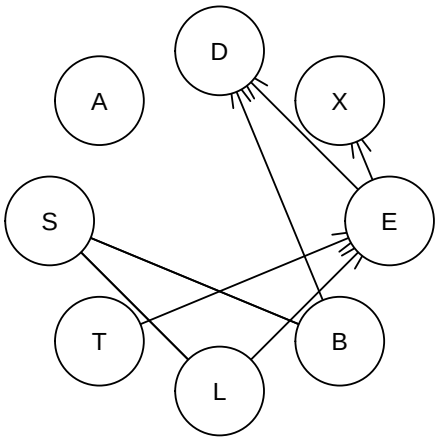
BN 1



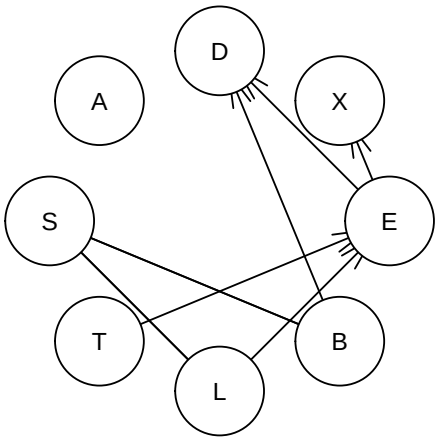
BN 2



cpdag BN1



cpdag BN2




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

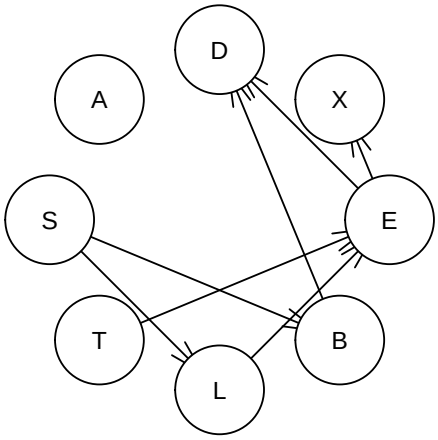
```
BN1 <- hc(data, score = "bde")
BN2 <- hc(data, score = "aic")
```

```r
par(mfrow = c(2,2))
compareBN(BN1, BN2)
```

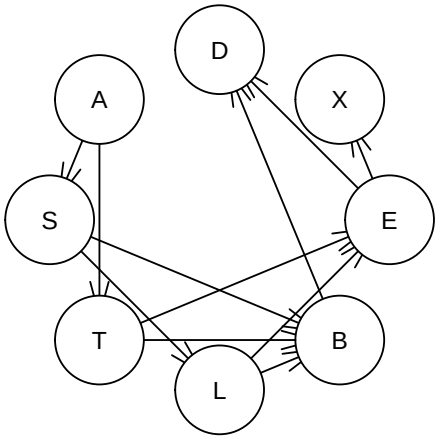
bde vs aic

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

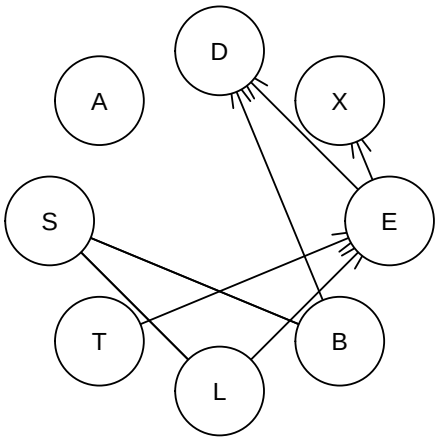
BN 1



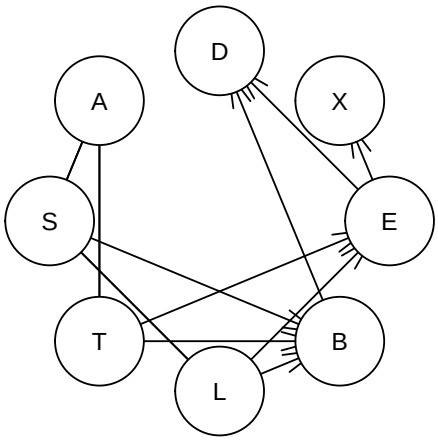
BN 2



cpdag BN1



cpdag BN2

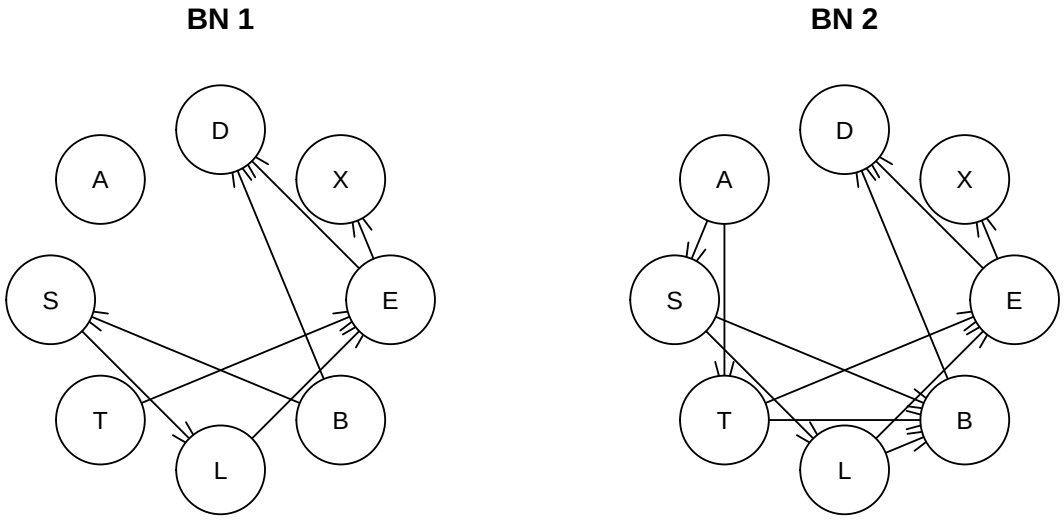


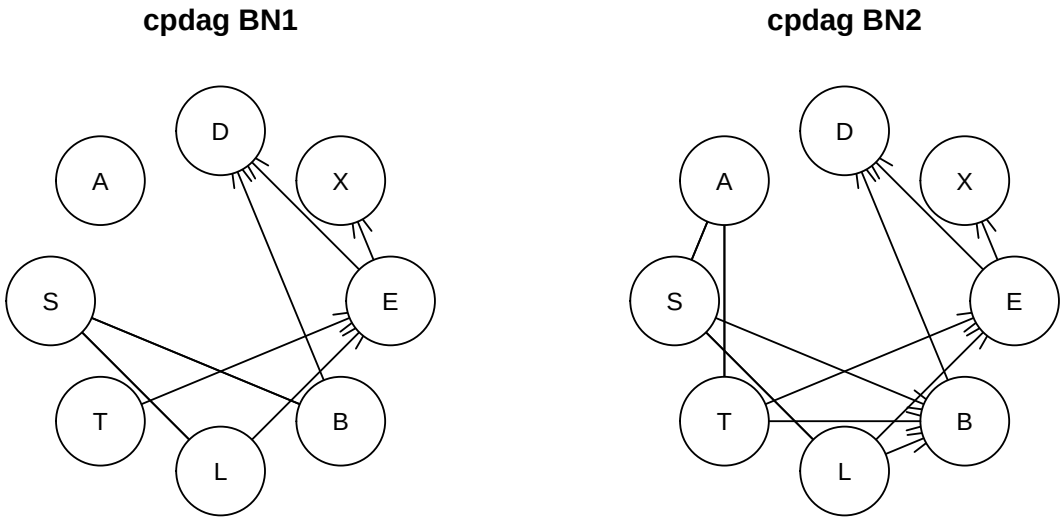

```
## [1] "Different number of directed/undirected arcs"
## [1] "Different number of directed/undirected arcs"
```

Different # of random restarts

```
BN1 <- hc(data, restart = 0)
BN1 <- hc(data, restart = 100)
par(mfrow = c(2,2))
compareBN(BN1, BN2)
```

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



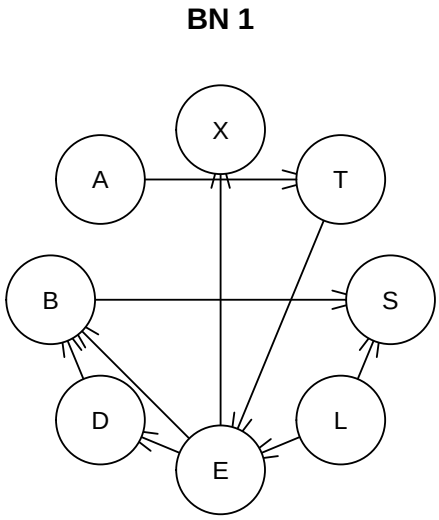


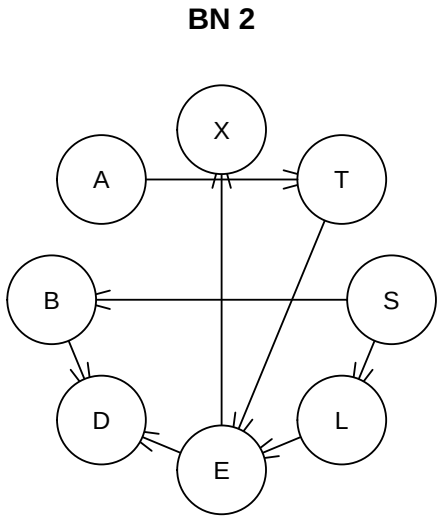

```
## [1] "Different number of directed/undirected arcs"
## [1] "Different number of directed/undirected arcs"
```

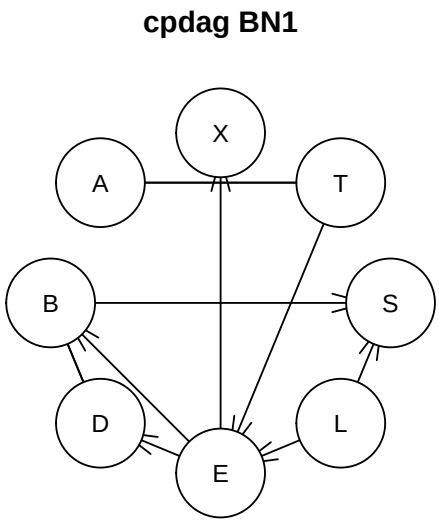
Different initial structures (DAGs)

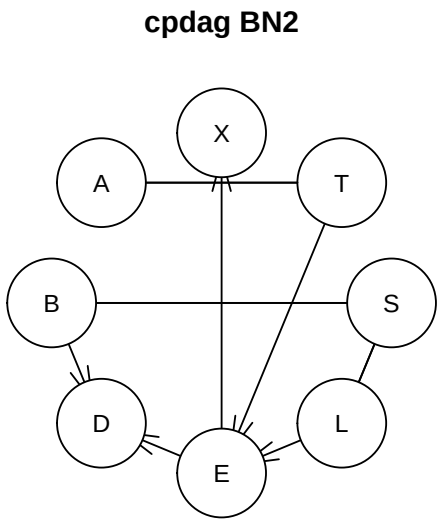
```
init_struct_1 <- model2network("[A|B:D][S|T:L:B][T][L][B|D][E][X][D]")
init_struct_2 <- model2network("[A][S][T][L][B][E][X][D]")
BN1 <- hc(data, start = init_struct_1, score = "bde", iss = 2)
BN2 <- hc(data, start = init_struct_2, score = "bde", iss = 2)
par(mfrow = c(2,2))
compareBN(BN1, BN2)
```

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










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

Reflection of task one

As shown above multiple runs of the HC algorithm can produce different, non-equivilant, BN structures. This is because the HC algorithm only reassures to find a global optima for convex problems. When the problem is not convex the algorithm can result in local optima where the solution cannot be improved by

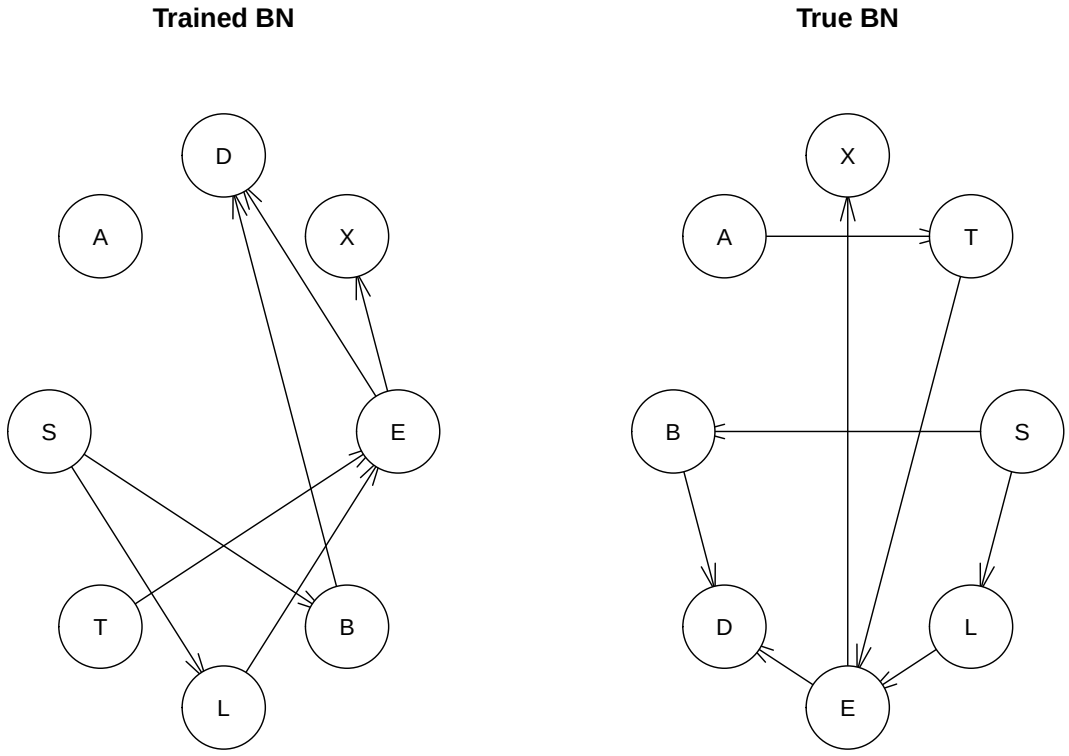
neighboring configurations, hence coming to a halt. Depending on the starting criteria this halt can occur in different "locations" in the search space. In this case we altered criteria such as the initial structure, restarts, scoring methods producing different BN structures with different number of directed/undirected arcs and different arc sets.

Task 2

Learn a BN from 80 % of the Asia dataset.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. Report the confusion matrix, i.e. true/false positives/negatives. Compare your results with those of the true Asia BN.

```
# Split the data set into training data (80%) and test data (20%)
sampling_size <- floor(0.8*nrow(data))
train_indices <- sample(seq_len(nrow(data)), sampling_size)
training_data <- data[train_indices,]
test_data <- data[-train_indices,]

# Learn the BN structure by training with the HC algorithm
# Then setting the true BN structure
# Finally plotting both networks to compare
BN_trained <- hc(x = training_data)
BN_true <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
par(mfrow = c(1,2))
plot(BN_trained, main = "Trained BN")
plot(BN_true, main = "True BN")
```




```
# Fitting the parameters to the training data for both the trained and true
# structure
BN_fit <- bn.fit(x = BN_trained, data = training_data)
BN_fit_true <- bn.fit(x = BN_true, data = training_data)

# Make the fitted BNs to gRain objects
BN_grain_fit <- as.grain(BN_fit)
BN_grain_fit_true <- as.grain(BN_fit_true)

# Setting the observed variables and the variable to be predicted
# Will be reused as in-parameters to the user-defined prediction function
obs_vars <- c("A","D", "X", "E", "B", "L", "T")
pred_vars <- c("S")

# Now its time to predict the S variable from the test data
# Uses own function BN_predict
# In parameters are the BN, test data, observed variables, variables to predict
BN_trained_res <- BN_predict(grain_obj = BN_grain_fit,
                         data = test_data,
                         obs_var = obs_vars,
                         pred_var = pred_vars)
BN_true_res <- BN_predict(grain_obj = BN_grain_fit_true,
```

```
                          data = test_data,
                          obs_var = obs_vars,
                          pred_var = pred_vars)

# Creating and printing the confusion matrix (CM) for both the trained BN and the true BN
CM <- table(BN_trained_res, test_data$S)
CM_true <- table(BN_true_res, test_data$S)
print(CM)
```

```
##
## BN_trained_res  no yes
##            no  332 126
##            yes 163 379
```

```
print(CM_true)
```

```
##
## BN_true_res  no yes
##         no  332 126
##         yes 163 379
```

```
print(missclass_check(BN_trained_res, test_data$S))
```

```
## [1] 0.289
```

Task 3

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.

```
# Now predict S only given the parents and children of the node
MB_trained <- mb(x = BN_fit, node = pred_vars)
MB_true <- mb(x = BN_fit_true, node = pred_vars)

MB_trained_res <- BN_predict(grain_obj =  BN_grain_fit,
           data = test_data,
           obs_var = MB_trained,
           pred_var = pred_vars)

MB_true_res <- BN_predict(grain_obj =  BN_grain_fit_true,
                                  data = test_data,
                                  obs_var = MB_true,
                                  pred_var = pred_vars)

# Create new confusion matrices
CM_MB_trained <- table(MB_trained_res, test_data$S)
CM_MB_true <- table(MB_true_res, test_data$S)
print(CM_MB_trained)
```

```
##
```

```
## MB_trained_res  no yes
##            no  332 126
##            yes 163 379
```

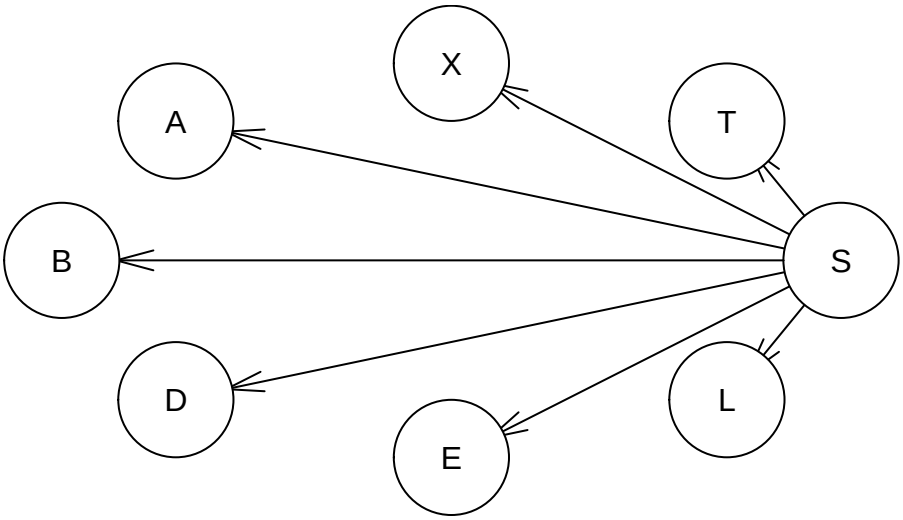
```
print(CM_MB_true)
```

```
##
## MB_true_res  no yes
##         no  332 126
##         yes 163 379
```

Task 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.

```
# Creating the Native Bayes network
BN_naive <- model2network("[S][A|S][T|S][L|S][B|S][D|S][E|S][X|S]")
plot(BN_naive)
```




```
# Fitting the parameters of the BN to the training data
BN_naive_fit <- bn.fit(x = BN_naive, data = training_data)
```

```r
# Transforming to a gRain object
BN_naive_grain <- as.grain(BN_naive_fit)

BN_naive_res <- BN_predict(grain_obj = BN_naive_grain,
                           data = test_data,
                           obs_var = obs_vars,
                           pred_var = pred_vars)

# Report the naive bayes CM
BN_naive_CM <- table(BN_naive_res, test_data$S)
print(BN_naive_CM)
```

```
##
## BN_naive_res  no yes
##          no  367 183
##          yes 128 322
```

```r
print(CM_true)
```

```
##
## BN_true_res  no yes
##         no  332 126
##         yes 163 379
```

```r
print(missclass_check(BN_naive_res, test_data$S))
```

```
## [1] 0.311
```

Task 5

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

Answer: The reason for the confusion matrices being the same in task 2 and task 3 is because the dependence of "S" is the same in both cases, which is "L" and "B" (which is the Markow blanket for S). Also, with knowledge about the state of L and B we have full information. Therefore the result is the same when doing prediction for the whole network and for the Markow blanket.

In task 4 on the other hand the network have completely different dependency where all the nodes are dependent on "S". Considering Bayes Theorem this affects the construction of the posterior distribution of S and it is therefor reasonable that the confusion matrices from the trained and true BNs are different. We know the true structure of the BN and that the posterior should only consist of the distributions of L, B and the prior of S.