Adv. ML. Lab 1

Simon Jorstedt

2024-09-14

```
# Packages and other setup
library(gRain)
```

```
## Loading required package: gRbase
```

```
library(BiocManager)
library(bnlearn)
```

```
##
## Attaching package: 'bnlearn'
```

```
## The following objects are masked from 'package:gRbase':
##
##     ancestors, children, parents
```

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
data("asia")
head(asia, 3)
```

```
##    A   S   T  L   B   E   X   D
## 1 no yes  no no yes  no  no yes
## 2 no yes  no no  no  no  no  no
## 3 no  no yes no  no yes yes yes
```

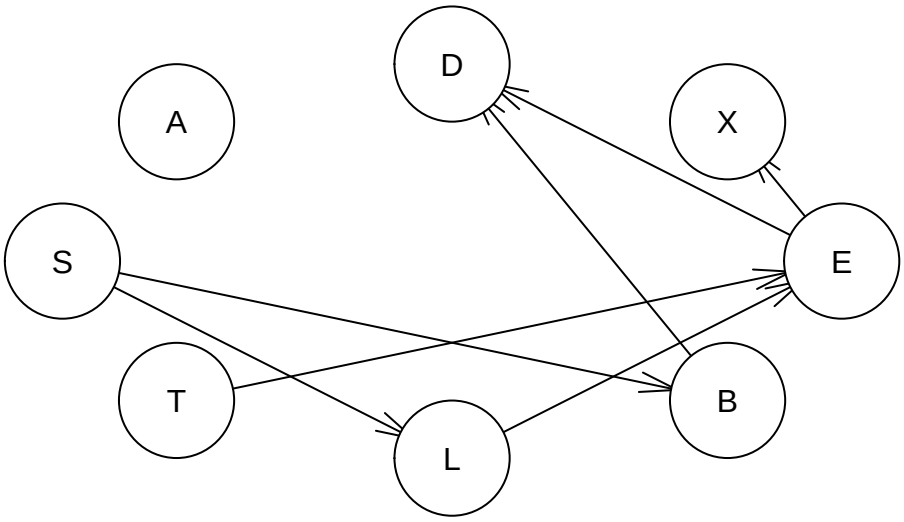
Problem 1

We are to show that multiple runs of HC can return different Bayesian Network structures. Two Bayesian Network structures are identical iff they have the same adjacencies, *and* they have the same unshielded colliders. We can check this by using the function `all.equal`. Below we see two very different Bayesian Network structures being generated. Here the two structures differ both in the adjacencies, and in the unshielded colliders.

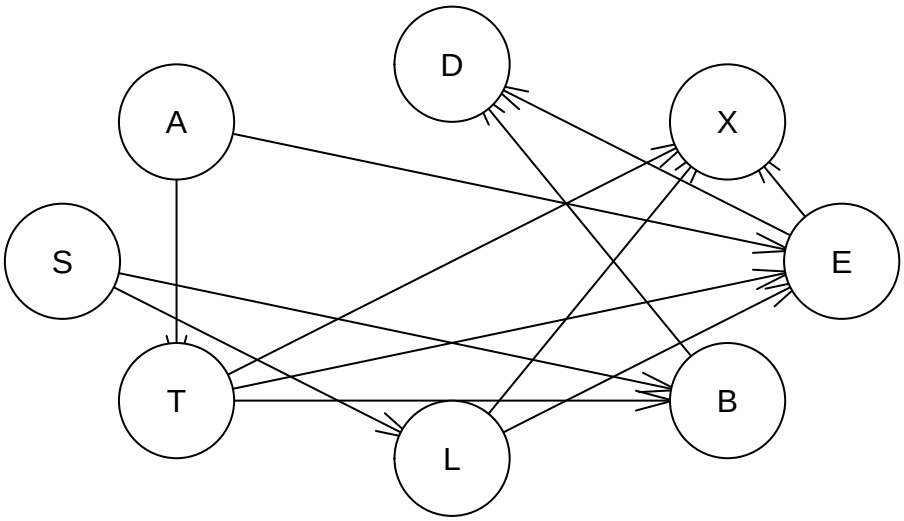
```r
# Run Hill Climbing Algorithm to find Bayesian Network strcucture

set.seed(83746)
hc_result_1 <- hc(x=asia)
hc_result_2 <- hc(x=asia, restart = 5, score="bde", iss=5)

# Check if BN structures are equal
all.equal(hc_result_1, hc_result_2)
```

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

```r
# Plot BN 1 and 2
plot(hc_result_1)
```



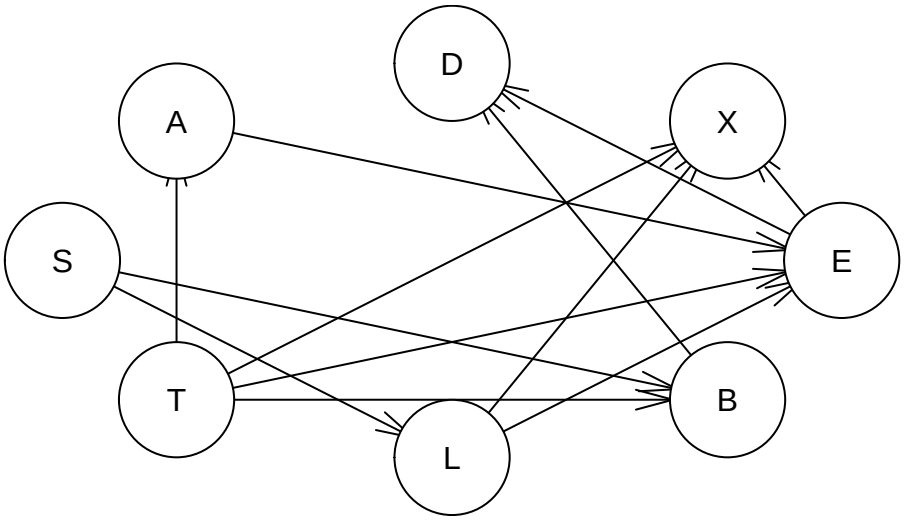

```r
plot(hc_result_2)
```



Problem 2 - Learn a BN

Now, we shall learn a Bayesian Network (structure *and* parameters) using 80% of the data. Then we will use the learned network to classify the remaining 20% of the data. Below we see that the structure does not contain any undirected edges, so we can go directly to fitting the parameters using maximum likelihood estimation.

```
# Split data
set.seed(81776)
train_indices <- sample(1:nrow(asia), 0.8*nrow(asia), replace = FALSE)
train_data <- asia[train_indices,]
test_data <- asia[-train_indices,]

# Fit the Bayesian network structure
bn_structure <- hc(x=train_data, restart = 100, score="bde", iss=5)
plot(bn_structure)
```




```
# Fit the parameters for a Bayesian Network structure.
parameters_bn1 = bn.fit(bn_structure, train_data, method = "bayes")
```

Now we can proceed to classify the remaining 20% of the observations. Below we plot the confusion matrix, which certainly performs better than random chance, but with a lot of false positives and negatives.

```
# Classify values of variable S for observations in test data

# Setup: create grain object, and create test data set without S column
grain <- compile(as.grain(parameters_bn1))
test_data_no_S <- test_data[,-2]
S_values <- c()

for (i in 1:nrow(test_data)){
  evidence <- setEvidence(grain,
                          nodes = colnames(test_data_no_S),
                          states = as.vector(t(test_data_no_S[i,])))
  estimate <- querygrain(evidence, nodes = c("S"), type = "marginal")
```

```
  # Encode an estimate as "yes" or "no" depending on the estimated probabilities
  if (estimate$S[[1]] >= 0.5){predicted_S <- "no"}
  else{predicted_S <- "yes"}
  S_values <- c(S_values, predicted_S)
}

# Construct confusion matrix for predictions vs true value
confusionMatrix(data=as.factor(S_values), reference=test_data$S)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  359 114
##        yes 145 382
##
##                Accuracy : 0.741
##                  95% CI : (0.7127, 0.7679)
##     No Information Rate : 0.504
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.4822
##
##  Mcnemar's Test P-Value : 0.06231
##
##             Sensitivity : 0.7123
##             Specificity : 0.7702
##          Pos Pred Value : 0.7590
##          Neg Pred Value : 0.7249
##              Prevalence : 0.5040
##          Detection Rate : 0.3590
##    Detection Prevalence : 0.4730
##       Balanced Accuracy : 0.7412
##
##        'Positive' Class : no
##
```

Now we will proceed to fit parameters to the "true" Bayesian Network which we specify manually as a string below. Again we will use the 80% set of the data to train the parameters, and then evaluate its performance when predicting the remaining 20%.

```
# Create network
true_dag <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

# Fit the parameters
params_true_dag = bn.fit(true_dag, train_data, method = "bayes")


# Classify values of variable S for observations in test data (in the "true" Bayesian Network)

# Setup: create grain object, and create test data set without S column
grain_true_dag <- compile(as.grain(params_true_dag))
S_values_true_dag <- c()
```

```r
for (i in 1:nrow(test_data)){
  evidence <- setEvidence(grain_true_dag,
                          nodes = colnames(test_data_no_S),
                          states = as.vector(t(test_data_no_S[i,])))
  estimate <- querygrain(evidence, nodes = c("S"), type = "marginal")

  # Encode an estimate as "yes" or "no" depending on the estimated probabilities
  if (estimate$S[[1]] >= 0.5){predicted_S <- "no"}
  else{predicted_S <- "yes"}
  S_values_true_dag <- c(S_values_true_dag, predicted_S)
}

# Construct confusion matrix for predictions vs true value for "true" BN
confusionMatrix(data=as.factor(S_values_true_dag), reference=test_data$S)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  357 111
##        yes 147 385
##
##                Accuracy : 0.742
##                  95% CI : (0.7137, 0.7689)
##     No Information Rate : 0.504
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.4843
##
##  Mcnemar's Test P-Value : 0.02933
##
##             Sensitivity : 0.7083
##             Specificity : 0.7762
##          Pos Pred Value : 0.7628
##          Neg Pred Value : 0.7237
##              Prevalence : 0.5040
##          Detection Rate : 0.3570
##    Detection Prevalence : 0.4680
##       Balanced Accuracy : 0.7423
##
##        'Positive' Class : no
##
```

We find that both the trained network parameters achieve an accuracy of about 74 %.

Problem 3

Now we shall see what happens when we train the parameters using only the markov blanket of S, which we obtain using the `mb` function in `bnlearn`. We will do so for the parameters trained on the "true" Bayesian Network. In this DAG, the blanket only contains the nodes "B" and "L" which are children of "S". The

result might shock you! We see that the accuracy is now lower than when predicting using the entire fitted network. This implies that the actual underlying independence structure is in fact even more complicated.

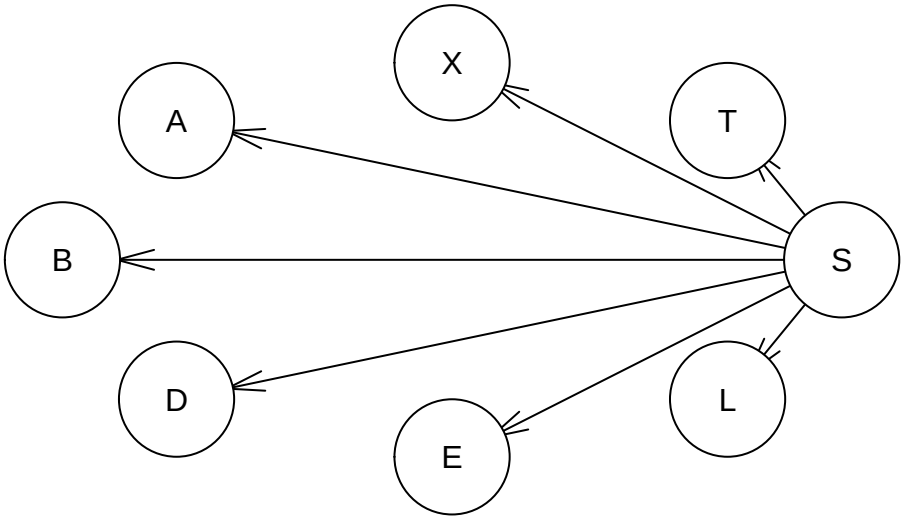
```
# Extract Markov blanket
markov_blanket <- mb(params_true_dag, "S")

S_values_true_dag_blanket <- c()
test_data_blanket <- test_data[,c(-2, -4, -5)]

for (i in 1:nrow(test_data)){
  evidence <- setEvidence(grain_true_dag,
                          nodes = colnames(test_data_blanket),
                          states = as.vector(t(test_data_blanket[i,])))
  estimate <- querygrain(evidence, nodes = c("S"), type = "marginal")

  # Encode an estimate as "yes" or "no" depending on the estimated probabilities
  if (estimate$S[[1]] >= 0.5){predicted_S <- "no"}
  else{predicted_S <- "yes"}
  S_values_true_dag_blanket <- c(S_values_true_dag_blanket, predicted_S)
}

# Construct confusion matrix for predictions vs true value for "true" BN
confusionMatrix(data=as.factor(S_values_true_dag_blanket), reference=test_data$S)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  360 181
##        yes 144 315
##
##                Accuracy : 0.675
##                  95% CI : (0.645, 0.704)
##     No Information Rate : 0.504
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.3496
##
##  Mcnemar's Test P-Value : 0.04583
##
##             Sensitivity : 0.7143
##             Specificity : 0.6351
##          Pos Pred Value : 0.6654
##          Neg Pred Value : 0.6863
##              Prevalence : 0.5040
##          Detection Rate : 0.3600
##    Detection Prevalence : 0.5410
##       Balanced Accuracy : 0.6747
##
##        'Positive' Class : no
##
```

Problem 4

Now we shall first encode a Naive Bayes network structure, train the parameters on it using the training data, and then classify test observations using this set of Network and parameters. The results posted below indicate that this network structure performs slightly worse (overlapping accuracy confidence intervals) than our structures from problem 2, and slightly better (also overlapping confidence intervals) than our structure from problem 3.

```
dag_bayes <- model2network("[A|S][T|S][L|S][B|S][E|S][X|S][D|S][S]")
plot(dag_bayes)
```




```
# Fit Bayesian Network on the Naive Bayes independence structure
parameters_naive_bayes <- bn.fit(dag_bayes, train_data, method="bayes")

# Setup: create grain object, and create test data set without S column
grain_naive_bayes <- compile(as.grain(parameters_naive_bayes))
S_values_naive_bayes <- c()

for (i in 1:nrow(test_data)){
  evidence <- setEvidence(grain_naive_bayes,
                          nodes = colnames(test_data_no_S),
                          states = as.vector(t(test_data_no_S[i,])))
  estimate <- querygrain(evidence, nodes = c("S"), type = "marginal")

  # Encode an estimate as "yes" or "no" depending on the estimated probabilities
  if (estimate$S[[1]] >= 0.5){predicted_S <- "no"}
  else{predicted_S <- "yes"}
  S_values_naive_bayes <- c(S_values_naive_bayes, predicted_S)
}

# Construct confusion matrix for predictions vs true value for "true" BN
confusionMatrix(data=as.factor(S_values_naive_bayes), reference=test_data$S)
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction  no yes
##        no  393 185
##        yes 111 311
##
##                Accuracy : 0.704
##                  95% CI : (0.6746, 0.7322)
##     No Information Rate : 0.504
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4073
##
##  Mcnemar's Test P-Value : 2.205e-05
##
##             Sensitivity : 0.7798
##             Specificity : 0.6270
##          Pos Pred Value : 0.6799
##          Neg Pred Value : 0.7370
##              Prevalence : 0.5040
##          Detection Rate : 0.3930
##    Detection Prevalence : 0.5780
##       Balanced Accuracy : 0.7034
##
##        'Positive' Class : no
##
```

Problem 5

In summary, different results are obtained in problems 2-4, because our explicit independence assumptions about the network structure in each problem do not agree with the underlying "true" independence structure.