Advanced Machine Learning

Lab 1: Graphical Models

Jakob Berggren

2023-09-08

Question 1

Question 1 aimed to demonstrate that multiple runs of the hill-climbing algorithm can return non-equivalent Bayesian network structures. To do this, the asia dataset from bnlearn was used.

First, data was loaded according to the instructions. Two different Bayesian Networks were then created using the Hill-Climb (HC) algorithm included in the bnlearn package.

```
# Load data
data("asia")

# Two BNs using hill climb algorithm.
set.seed(12345) # Ensure we use the same seed
bn1 <- hc(x = asia, restart = 10)

set.seed(12345)
bn2 <- hc(x = asia, restart = 10, score = 'aic') # Using different score function.
```

Here I tested different methods to yield non-equivalent BNs and found that changing the score function worked.

Recall from the lecture that two DAGs are equivalent if and only if they have the same adjacencies and unshielded colliders. Thus checking the unshielded colliders of bn1 and bn2 respectively should yield different results.

```
# Check unshielded colliders. We will find they are different.
unshielded.colliders(bn1)
```

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

```
unshielded.colliders(bn2)
```

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

To ensure non-equivalence, convert BNs into CPDAGs to see marginal dependencies and direct causal relationships. Here I also used all.equal to check for equivalence.

```
# Convert to CPDAGs to standardize representation and check for equivalence.
bn1 <- cpdag(bn1)
bn2 <- cpdag(bn2)

all.equal(bn1, bn2)
```

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

Finally, we can also plot the CPDAGs to see that they are non-equivalent

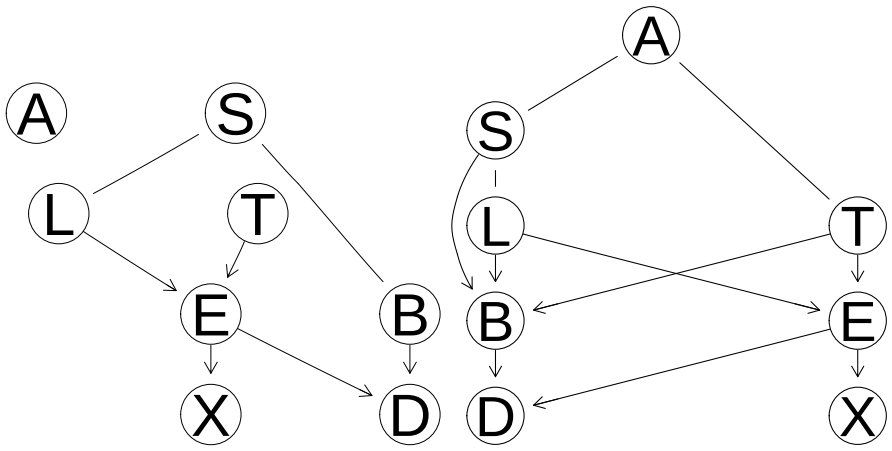

Figure 1: Non-equivalent Bayesian Structures

The reason why we get non-equivalent BN-structures, is that hill climb is not asymptotically correct under faithfulness, i.e. it may get trapped in local optima. This results in the algorithm yielding different outcomes.

Question 2

The objective of question 2 was to learn a BNs structure and parameters from 80% training data (obtained from the asia dataset), and then use the BN to compute the posterior distribution of S = yes and S = no using exact or approximate inference. The results should also be compared with those of the true Asia BN.

First step was to once again load the asia dataset, and then split it in to 80/20 train and test respectively.

```
# Again, Load the data.
data("asia")

# Split into 80/20 train test.
n <- dim(asia)[1]
set.seed(12345)
id <- sample(1:n, floor(n * 0.8))
train <- asia[id, ]
test  <- asia[-id, ]
```

Using the training data, the BN structure was learned with the Hill Climb Algorithm. Furthermore the true asia BN structure was obtained by running the given code. Using the structures, the parameters could then be learned. Again, using the training data.

```
# Learning BN structure using HC.
set.seed(12345)
bn_struct <- hc(x = train, restart = 10)
true_struct <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]") # True Asia BN

# Learn parameters
bn_param <- bn.fit(x = bn_struct, data = train)
true_param <- bn.fit(x = true_struct, data = train)
```

To make predictions and compute inference, the BN is transformed into a gRain object and compiled.

```
# Transform into gRain object and compile.
grain_obj <- as.grain(bn_param)
bn_compiled <- compile(grain_obj)
```

```
true_grain_obj <- as.grain(true_param)
true_bn_compiled <- compile(true_grain_obj)
```

Next, predictions are made on the test dataset.

```
# Custom predict function
predict_bn <- function(obj) {
  pred <- c()
  for (i in 1:nrow(test)) {
    evidence <- setEvidence(object = obj,
                            nodes = c("A", "T", "L", "B", "E", "X", "D"),
                            states = as.character(unlist(test[i,-2])))
    query <- querygrain(evidence, nodes = c("S"))$S
    pred[i] <- ifelse(query["yes"] > 0.5, "yes", "no")
  }
  return (pred)
}
```

For each row in the test dataset, evidence is observed for all nodes except "S" (since this is the one we want to compute the conditional probability for). Then querygrain is used to compute the probability of "S" given the evidence in the step before. The ifelse statement then predicts "yes" if the probability is >0.5 and "no" otherwise.

The predict function is used to compute the confusion matrices for both the learned and the true BN. We will find that they yield the same result.

```
# Confusion matrices
table(predict_bn(bn_compiled), test$S)
```

```
##
##        no yes
##   no  337 121
##   yes 176 366
```

```
table(predict_bn(true_bn_compiled), test$S) # CM from True Asia BN
```

```
##
##        no yes
##   no  337 121
##   yes 176 366
```

Question 3

Question 3 once again asked to classify the variable S, but this time only given the observations for the Markov Blanket of S.

The Markov Blanket consists of the parents of S, plus the children of S, plus the parents of the children minus S itself. In the case of S, this leaves the nodes L and B. The markov blanket could also be obtained by using the function mb from bnlearn. Once again, this was done both for the true asia BN and the learned BN from the question 2.

```
mb <- mb(bn_param, c("S"))
mb_true <- mb(true_param, c("S"))
```

Next, a slight variation of the previous predict function was used to make predictions only using the markov blanket. Here "nodes" consists of the markov blanket.

```r
# Custom predict function
predict_mb <- function(obj, nodes) {
  pred <- c()
  for (i in 1:nrow(test)) {
    evidence <- setEvidence(object = obj,
                            nodes = nodes,
                            states = as.character(unlist(
                              subset(test, select = nodes)[i,])))
    query <- querygrain(evidence, nodes = c("S"))$S
    pred[i] <- ifelse(query["yes"] > 0.5, "yes", "no")
  }
  return (pred)
}
```

Again, the confusion matrices was computed. As seen, the same result as in question 2 was obtained.

```r
# Predict and create confusion matrices.
table(predict_mb(bn_compiled, mb), test$S)
```

```
##
##        no yes
##   no  337 121
##   yes 176 366
```

```r
table(predict_mb(true_bn_compiled, mb_true), test$S)
```

```
##
##        no yes
##   no  337 121
##   yes 176 366
```

Question 4

Queston 4 asked to repeat excercise 2, but instead using a naive bayes classifier. In a naive bayes network, the predictive variables are independent given the class variable. First, I created a naive bayes network as follows.

```r
# Naive Bayes Structure
nb <- model2network("[S][A|S][B|S][D|S][E|S][L|S][T|S][X|S]")
```

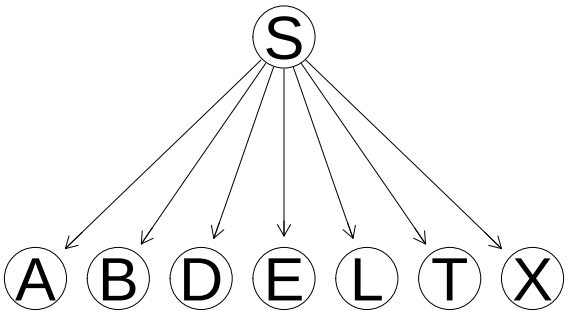

Figure 2: Naive Bayes Structure

The rest was very similar to Question 2. First the parameters was learned using the training data. The nb was then transformed into an gRain object and compiled.

```r
nb <- bn.fit(x = nb, data = train) # Parameters from training data

# Transform into gRain object and compile.
nb <- as.grain(nb)
nb <- compile(nb)
```

Using the same predict function as in Question 2, the confusion matrix was computed. We will find a slight variation in the results here.

```r
# Confusion matrix
table(predict_bn(nb), test$S)
```

```
##
##        no yes
##   no  359 180
##   yes 154 307
```

Question 5

Question 5 asks to explain why the same or different results are obtained through Question 2-4.

The reason for obtaining the same results in Q2 and Q3 (and Q4) lies in the structure of the network. In this case, "S" is conditionally independent other nodes given its markov blanket. This means that the markov blanket alone will capture the relationships of S with other nodes in the network, and thus the markov blanket contains all information needed to make the same predictions as using the entire network. This is why we get the same result.

In Q4, a naive Bayes structure is used instead. This can be seen as a simplification of the original structure, where the structure of the BN is essentially ignored for a simpler modeling process. In this representation, a different markov blanket is obtained (all nodes but S is the markov blanket of S here), and we can expect a different result when doing predictions.