Laboration group report in Advanced Machine Learning

Laboration 1

732A96

Duc Tran
William Wiik
Mikael Montén
Johannes Hedström

Division of Statistics and Machine Learning
Department of Computer Science
Linköping University

12 september 2024

Contents

1 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.

Answer:

Two Bayesian network structures are considered non-equivalent if they represent different sets of conditional independencies.

The hill-climbing(hc) algorithm starts with an empty dag. Then it tries to add, remove or reverse any edge in G that improves the BDeu score the most. With this heuristic search, the hc algorithm can get stuck in a local optima and we can not guarantee we get the DAG with the highest BDeu score. There are 2 parameters; "restart" and "imaginary sample size"(iss) that we can change in the hc algorithm that can affect the DAG the algorithm returns.

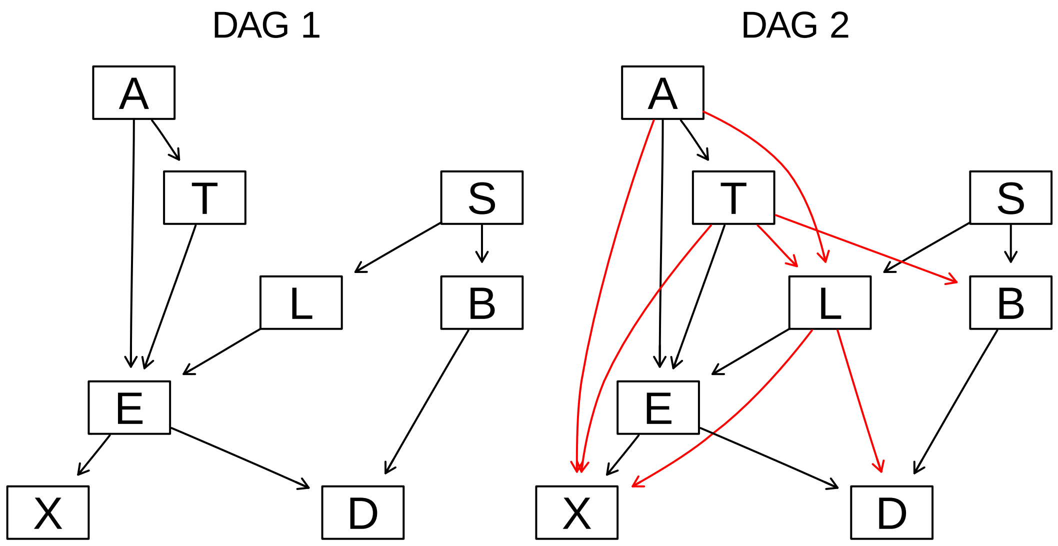
Restart specifies the number of random restarts the algorithm performs (to reduce risk of get stuck in one local optima). With iss, the number of edges will be affected, where higher values is less regularization.

A DAG with iss 4(DAG 1) and DAG with iss 30(DAG 2) is presented below

```r
set.seed(12345)
# hc algorithm on data with empty dag(default value)
skeleton = hc(asia,                # Data
              restart = 0,         # Number of random restarts
              iss = 4,             # Imaginary sample size (regularization)
              score="bde")         # BDeu score

# hc algorithm on data with empty dag(default value)
skeleton2 = hc(asia,               # Data
              restart = 0,         # Number of random restarts
              iss = 30,            # Imaginary sample size (regularization)
              score="bde")         # BDeu score

graphviz.compare(skeleton, skeleton2,
                 main = c("DAG 1", "DAG 2"), layout = "dot")
```



In the figure, we can see that DAG 1 has 9 edges. DAG 2 has the same edges (with same direction) as DAG 1 with an addition of 7 edges highlighted in red.

2 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]").

Answer: In this task the hc algorithm is used with iss set to four and restart set to 100. Data is split 80/20 by the code as follows:

```r
# Split data to training/test (80/20)
set.seed(12345)
n <- dim(asia)[1]
train_ind <- sample(1:n, 0.8*n)
train_df <- asia[train_ind, ]
test_df <- asia[-train_ind, ]
```

The skeleton of the DAG is found with the hc algoritm, the marginal probabilities are found with maximum likelihood estimates with the code as follows.

```r
# Learn the network structure (DAG)
dag_1 = hc(train_df,             # Data
              restart = 100,        # Number of random restarts
              iss = 1,              # Imaginary sample size
              score="bde")          # Score
# Learn the conditional probabilities from data
bn_dag = bn.fit(dag_1,
                train_df,
                method = "mle")
```

The test data were classified as "no" or "yes" depending on the largest value between:
$p(S = "yes"|A, T, L, E, X, B, D)$ and $p(S = "no"|A, T, L, E, X, B, D)$ with the code as follows:

```r
grain_dag <- as.grain(bn_dag)
pred_fun <- function(grain_dag, observation, pred_index){
  # Returns p(s|a,b,c)
  # Where s = observation[pred_index] and a,b,c are the remaining columns in observation
  # Get values as "no", "yes"
  values <- unlist(lapply(observation[-pred_index], as.character))
  # Extract variable names
  nodes <- names(values)

  # set a,b,c for p(s|a,b,c)
  evidence <- setEvidence(grain_dag,
                          nodes = nodes,
                          states = values)
  # for binary: gets p(s="no"|a,b,c) and p(s="yes"|a,b,c)
  prob <- querygrain(evidence,
                     nodes = names(observation[pred_index]),
                     type="marginal")[[1]]

  # Predict by largest value (pred=yes when tied)
  pred <- ifelse(prob["no"] > prob["yes"], "no", "yes")
  return(pred)
}

# Predict observations from DAG
for (index in 1:dim(test_df)[1]){
  test_df$pred[index] <- pred_fun(grain_dag, test_df[index, ], 2)
}
```

The confusion matrix is presented in table 1 with the accuracy in the output.

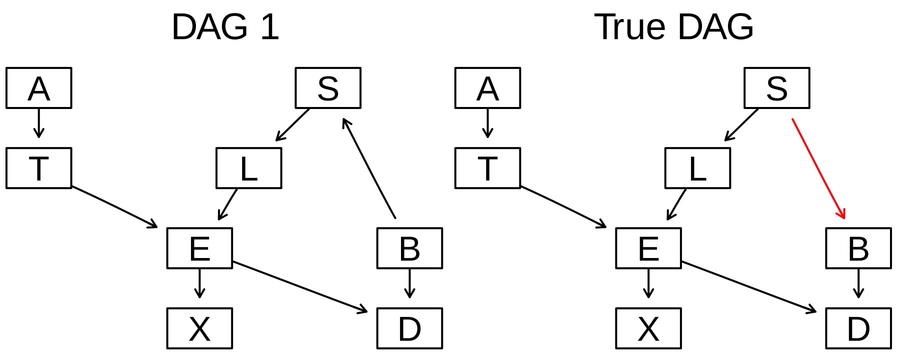
Table 1: Confusion matrix for test data on DAG 1.

	no	yes
no	337	176
yes	121	366

```
## Accuracy:  0.703
```

The true DAG is created and the conditional probabilities are estimated with mle. The true DAG is presented in figure 4. Test data is predicted with the true dag and the confusion matrix is presented in table 2.

```r
true_dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
graphviz.compare(dag_1, true_dag,
                 main = c("DAG 1", "True DAG"), layout = "dot")
```



Comparing DAG 1 with the True DAG we see that the edge between S and B is S<-B in DAG 1 and the reverse in the True DAG. From the lecture slides these graphs are equivalent in probability.

The confusion matrix and accuracy is presented in table 2.

Table 2: Confusion matrix for test data on the true DAG.

	no	yes
no	337	176
yes	121	366

```
## Accuracy:  0.703
```

Comparing the confusion matrix and accuracy from table 2 with table 1, we can conclude that the confusion matrices are identical.

3 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.

Answer:

In figure comparing DAG 1 and True DAG, the parent to S is B. The children to S is L. The parents to "L" is "S" and B has no parent. The Markov blanket of S is "L" and "B".

```r
mb(bn_dag, "S")
```

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

The confusion matrix and accuracy for the prediction when only "B" and "L" are given are presented in table 3.

```r
ind_b <- which(colnames(test_df) == "B")
ind_l <- which(colnames(test_df) == "L")

# Predict observations from DAG
for (index in 1:dim(test_df)[1]){
  test_df$pred_c[index] <- pred_fun(grain_dag, test_df[index, c(2, ind_b, ind_l)], 1)
}
```

Table 3: Confusion matrix for test data on the my DAG when only evidence for L and B are given.

	no	yes
no	337	176
yes	121	366

```
## Accuracy:  0.703
```

The confusion matrix in table 3 is identical to the confusion matrix in the true dag in table 2.

4 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.

Answer: From Bishop's book we have that conditioned on S, the distribution of the input variables "A", "T", "L", "B", "E", "X", and "D" are independent.

So for example $A \perp T|S$, $A \perp L|S$, $\dots$, $A \perp D|S$ can be read from the graph. Similarly changing "A" to any other variables they same type of independences can be read.

For $A \perp T|S$ we can have the constellation "chain" (A->S->T) or "fork" (A<-S->T). Therefore we can create a naive Bayes classifier as a BN by creating a DAG where S is a parent to all other nodes.

The naive Bayes DAG is presented in figure 5.

```
naive_dag = model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")
graphviz.plot(naive_dag, layout = "dot")
```

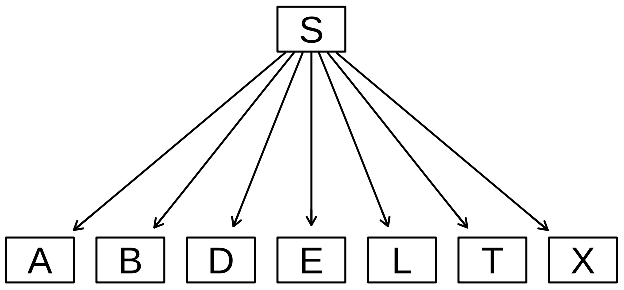


Figure 1: Naive Bayes DAG.

The parameters for the naive Bayes DAG are estimated with mle with training data. The confusion matrix for the predictions on test data is presented in table 4.

```
naive_dag = bn.fit(naive_dag,
                   train_df,
                   method = "mle")
naive_dag <- as.grain(naive_dag)
# Predict observations from DAG
for (index in 1:dim(test_df)[1]){
  test_df$naive_dag[index] <- pred_fun(naive_dag, test_df[index, -c(9,10,11)], 2)
}
```

Table 4: Confusion matrix for test data on the naive Bayes DAG.

	no	yes
no	359	154
yes	180	307

```
## Accuracy:  0.666
```

The confusion matrix in table 4 is not similar to the two previous DAGs, where accuracy is lower.

5 Question 5

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

Answer:

Accuracy for exercises (2-4)

- Question 2 (DAG1) : 0.703
- Question 2 (True DAG) : 0.703
- Question 3 (DAG1 with blanket) : 0.703
- Question 4 (DAG1) : 0.666

Two DAGs represent the same independencies according to the separation criterion if and only if the have the same adjacencies and unshielded colliders. We have that DAG1 and True DAG have same adjacencies and unshielded colliders, therefore they are equivalent.

The Markov blanket tells us that if we have evidence for L and B, then all other evidences for the other variables are not needed to infer S. In other words, $S \perp A, T, E, X, B, D | L, B$. This is why we get the same result in (2) and (3).

The naive Bayes does not have the same adjecencies and unshielded colliders as the two other DAGs, therefore the naive Bayes DAG is not equivalent to the two other DAGs. This can be seen in the confusion matrices, where the accuracy was lower for the naive Bayes DAG.

6 Statement of contribution

We compared everyone results and decided to go for the code structure from Duc Tran.
To ensure consistency, we compared each individual report by using the same seed and BN structure. This allowed us to verify that everyone had implemented the correct functions with the appropriate arguments. Once this was done, all results was the same, giving us confidence in proceeding with Duc Tran's code structure.

In the discussion section, we collectively summarized the input from all people. We settled on the code and interpretations that resonated best with everyone. Each person contributed to the discussions by sharing insights and perspectives on the topic.

7 Appendix

The code used in this laboration report are summarised in the code as follows:

```r
library("knitr")
library("bnlearn")
#library("RBGL")
library("Rgraphviz")
library("gRain")
knitr::opts_chunk$set(
  echo = TRUE,
  fig.width = 3.5,
  fig.height = 2)
set.seed(12345)
# hc algorithm on data with empty dag(default value)
skeleton = hc(asia,                 # Data
              restart = 0,          # Number of random restarts
              iss = 4,              # Imaginary sample size (regularization)
              score="bde")          # BDeu score

# hc algorithm on data with empty dag(default value)
skeleton2 = hc(asia,                # Data
              restart = 0,          # Number of random restarts
              iss = 30,             # Imaginary sample size (regularization)
              score="bde")          # BDeu score

graphviz.compare(skeleton, skeleton2,
                 main = c("DAG 1", "DAG 2"), layout = "dot")
# Split data to training/test (80/20)
set.seed(12345)
n <- dim(asia)[1]
train_ind <- sample(1:n, 0.8*n)
train_df <- asia[train_ind, ]
test_df <- asia[-train_ind, ]
# Learn the network structure (DAG)
dag_1 = hc(train_df,               # Data
              restart = 100,          # Number of random restarts
              iss = 1,                # Imaginary sample size
              score="bde")            # Score
# Learn the conditional probabilities from data
bn_dag = bn.fit(dag_1,
                train_df,
                method = "mle")
grain_dag <- as.grain(bn_dag)
pred_fun <- function(grain_dag, observation, pred_index){
  # Returns p(s|a,b,c)
  # Where s = observation[pred_index] and a,b,c are the remaining columns in observation
  # Get values as "no", "yes"
  values <- unlist(lapply(observation[-pred_index], as.character))
  # Extract variable names
  nodes <- names(values)

  # set a,b,c for p(s|a,b,c)
  evidence <- setEvidence(grain_dag,
                          nodes = nodes,
                          states = values)
  # for binary: gets p(s="no"|a,b,c) and p(s="yes"|a,b,c)
  prob <- querygrain(evidence,
                     nodes = names(observation[pred_index]),
                     type="marginal")[[1]]

  # Predict by largest value (pred=yes when tied)
  pred <- ifelse(prob["no"] > prob["yes"], "no", "yes")
  return(pred)
}

# Predict observations from DAG
for (index in 1:dim(test_df)[1]){
  test_df$pred[index] <- pred_fun(grain_dag, test_df[index, ], 2)
}
conf_matrix <- table(true = test_df$S, predict = test_df$pred)
kable(conf_matrix,
      caption = "Confusion matrix for test data on DAG 1.")
cat("Accuracy: ", round(sum(diag(conf_matrix))/sum(conf_matrix) ,3))
true_dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
graphviz.compare(dag_1, true_dag,
                 main = c("DAG 1", "True DAG"), layout = "dot")

true_dag = bn.fit(true_dag,
                  train_df,
                  method = "mle")
true_dag <- as.grain(true_dag)
# Predict observations from DAG
for (index in 1:dim(test_df)[1]){
  test_df$true_pred[index] <- pred_fun(true_dag, test_df[index, -9], 2)
}

conf_matrix <- table(true = test_df$S, predict = test_df$true_pred)
kable(conf_matrix,
      caption = "Confusion matrix for test data on the true DAG.")
cat("Accuracy: ", round(sum(diag(conf_matrix))/sum(conf_matrix) ,3))
mb(bn_dag, "S")
ind_b <- which(colnames(test_df) == "B")
ind_l <- which(colnames(test_df) == "L")

# Predict observations from DAG
for (index in 1:dim(test_df)[1]){
  test_df$pred_c[index] <- pred_fun(grain_dag, test_df[index, c(2, ind_b, ind_l)], 1)
}
```

```r
conf_matrix <- table(true = test_df$S, predict = test_df$pred_c)
kable(conf_matrix,
      caption = "Confusion matrix for test data on the my DAG when only evidence for L and B are given."
cat("Accuracy: ", round(sum(diag(conf_matrix))/sum(conf_matrix) ,3))
naive_dag = model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")
graphviz.plot(naive_dag, layout = "dot")
naive_dag = bn.fit(naive_dag,
                   train_df,
                   method = "mle")
naive_dag <- as.grain(naive_dag)
# Predict observations from DAG
for (index in 1:dim(test_df)[1]){
  test_df$naive_dag[index] <- pred_fun(naive_dag, test_df[index, -c(9,10,11)], 2)
}
conf_matrix <- table(true = test_df$S, predict = test_df$naive_dag)
kable(conf_matrix,
      caption = "Confusion matrix for test data on the naive Bayes DAG.")
cat("Accuracy: ", round(sum(diag(conf_matrix))/sum(conf_matrix) ,3))
```