732A96 - Advanced Machine Learning - Lab 1

Qinyuan Qi(qinqi464)

2024-09-14

Exercises 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 in- cluded 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 sam- ple 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

Answer:

According to the question, we will change the score, start, restart and iss number to see the difference between the two hill climbing algorithms. Also we will compare the output of cpdag and vstructs to see difference of the hill climbing algorithms with different parameters.

1.1 default parameters vs restart = 10

We call hc function with the default parameters(score=bic,restart=0), and then change restart to 10, after that we verify the equivalence between the two hill climbing algorithms 1 and 2.

```
############################  Code For Exercises 1.1 ##############################
# default parameters vs restart = 10

# Parameters
# 1. score:default:bic
#    1). bic:the Bayesian Information Criterion (BIC) score
#    2). bde:the logarithm of the Bayesian Dirichlet equivalent (uniform) score (BDeu)
# 2. restart:the number of random restarts(integer), default: 0
# 3. iss:the imaginary sample size used by the Bayesian Dirichlet scores
#   (bde, mbde, bds, bdj),default: 1
# 4. start:the initial structure of the network, default: empty graph(NULL)
data("asia")
hill_climbing_1 <- hc(asia)
hill_climbing_2 <- hc(asia, restart = 10)
# calc the score of the two hill climbing algorithms
cat("The score of hill climbing algorithm 1 and 2 are:", score(hill_climbing_1, asia),
    "and" ,score(hill_climbing_2, asia))
```

```
## The score of hill climbing algorithm 1 and 2 are: -11107.29 and -11107.29
```

```
# check equivalent using all.equal
cat("Equivalency between two hill climbing algorithms:",
    all.equal(hill_climbing_1, hill_climbing_2))
```

```
## Equivalency between two hill climbing algorithms: TRUE
```

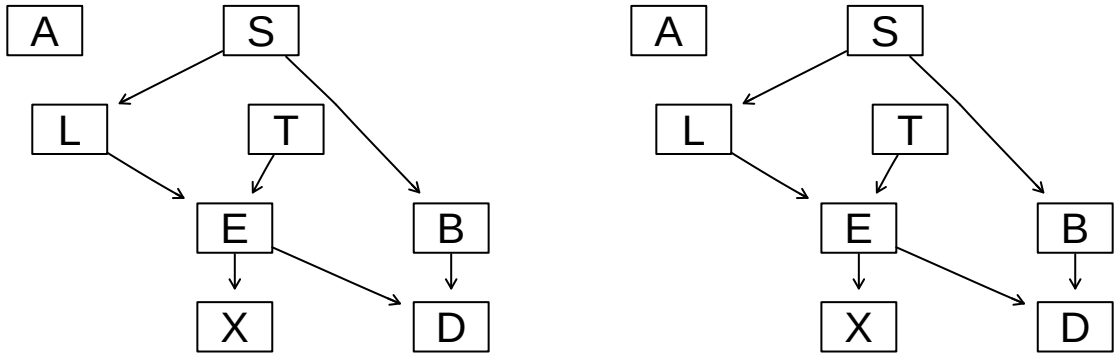


Figure 1: The difference between hill climbing algorithms 1 and 3

1.2 default parameters vs initial structure(random)

Then we specify the initial structure, and then check the equivalence between the two hill climbing algorithms 1 and 3.

```
############################  Code For Exercises 1.2 ##############################
# default parameters vs initial structure(random)

initial_structure <- random.graph(names(asia), num = 1, method = "ordered")
hill_climbing_3 <- hc(asia, start = initial_structure)
# calc the score of the two hill climbing algorithms
cat("The score of hill climbing algorithm 1 and 3 are:", score(hill_climbing_1, asia),
    "and" ,score(hill_climbing_3, asia))
```

```
## The score of hill climbing algorithm 1 and 3 are: -11107.29 and -11147.11
```

```
# check equivalent using all.equal
cat("Equivalency between two hill climbing algorithms:",
    all.equal(hill_climbing_1, hill_climbing_3))
```

```
## Equivalency between two hill climbing algorithms: Different number of directed/undirected arcs
```

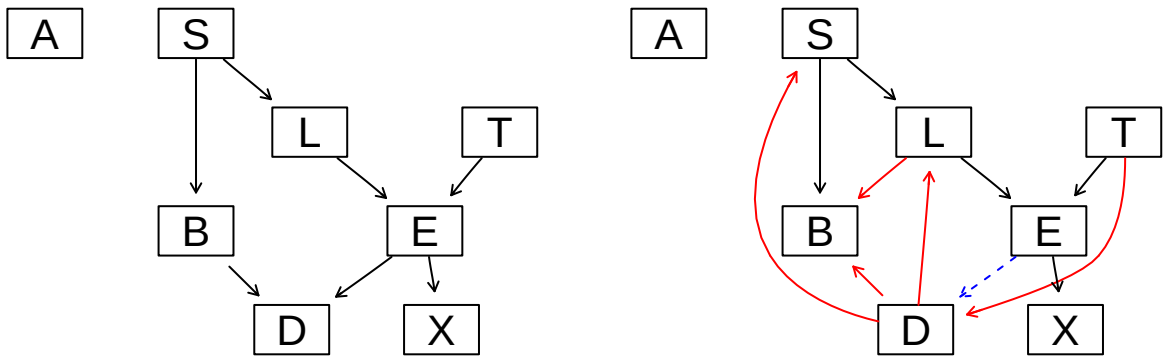


Figure 2: The difference between hill climbing algorithms 1 and 3

1.3 [restart = 100, BIC] vs [restart = 100, BDE]

Then we change the score to bde, and set restart to 100, verify the equivalence between the two hill climbing algorithms 2 and 4.

```
############################  Code For Exercises 1.3 ##############################
# [restart = 100, BIC] vs [restart = 100, BDE]

# change score value
hill_climbing_4 <- hc(asia,score = "bde", restart = 100)
# calc the score of the two hill climbing algorithms
cat("The score of hill climbing algorithm 2 and 4 are:", score(hill_climbing_2, asia),
    "and" ,score(hill_climbing_4, asia))
```

```
## The score of hill climbing algorithm 2 and 4 are: -11107.29 and -11107.29
```

```
# check equivalent using all.equal
cat("Equivalency between two hill climbing algorithms:",
    all.equal(hill_climbing_2, hill_climbing_4))
```

```
## Equivalency between two hill climbing algorithms: TRUE
```

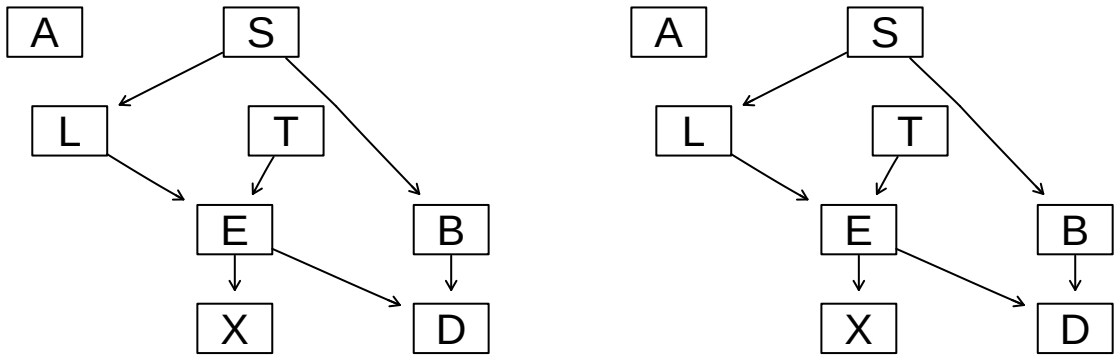


Figure 3: The difference between hill climbing algorithms 2 and 4

1.4 [BDE, restart = 100, ISS=1(default)] vs [BDE, restart = 100, iss = 100]

Then we copy all the parameter of hill_climbing_4 and change iss to 100.

```
############################  Code For Exercises 1.4 ##############################
# [BDE, restart = 100, ISS=1(default)] vs [BDE, restart = 100, iss = 100]

# change iss value
hill_climbing_5 <- hc(asia,score = "bde", restart = 100, iss = 100)
# calc the score of the two hill climbing algorithms
cat("The score of hill climbing algorithm 4 and 5 are:", score(hill_climbing_4, asia),
    "and" ,score(hill_climbing_5, asia))
```

```
## The score of hill climbing algorithm 4 and 5 are: -11107.29 and -11324.21
```

```
# check equivalent using all.equal
cat("Equivalency between two hill climbing algorithms:",
    all.equal(hill_climbing_4, hill_climbing_5))
```

```
## Equivalency between two hill climbing algorithms: Different number of directed/undirected arcs
```

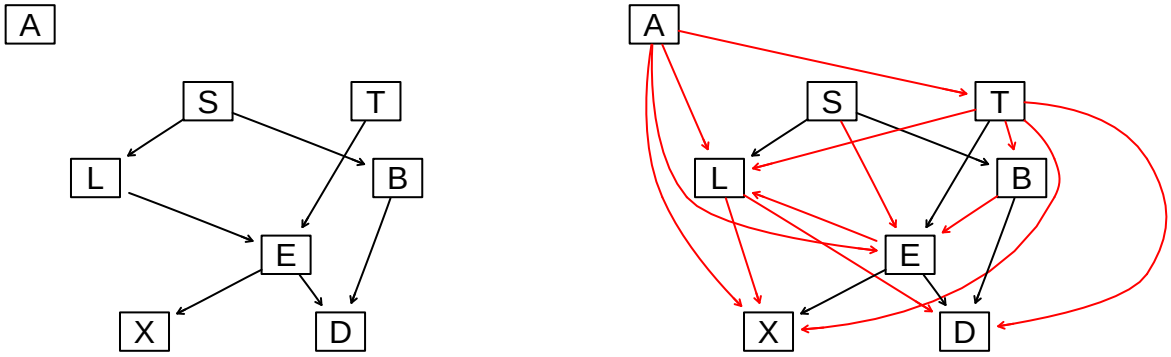


Figure 4: The difference between hill climbing algorithms 4 and 5

1.5 output of cpdag and vstructs

We also check the cpdag output of all hill climbing algorithms,listed below.

```
##
##   Bayesian network learned via Score-based methods
##
##   model:
##     [partially directed graph]
##   nodes:                                 8
##   arcs:                                  7
##     undirected arcs:                     2
##     directed arcs:                       5
##   average markov blanket size:           2.25
##   average neighbourhood size:            1.75
##   average branching factor:              0.62
##
##   learning algorithm:                    Hill-Climbing
##   score:                                 BIC (disc.)
##   penalization coefficient:              4.258597
##   tests used in the learning procedure:  77
##   optimized:                             TRUE

##
##   Bayesian network learned via Score-based methods
##
##   model:
##     [partially directed graph]
##   nodes:                                 8
##   arcs:                                  7
##     undirected arcs:                     2
##     directed arcs:                       5
##   average markov blanket size:           2.25
##   average neighbourhood size:            1.75
##   average branching factor:              0.62
##
##   learning algorithm:                    Hill-Climbing
##   score:                                 BIC (disc.)
##   penalization coefficient:              4.258597
##   tests used in the learning procedure:  255
##   optimized:                             TRUE
```

```
##
##   Bayesian network learned via Score-based methods
##
##   model:
##     [partially directed graph]
##   nodes:                                8
##   arcs:                                 10
##     undirected arcs:                    7
##     directed arcs:                      3
##   average markov blanket size:          2.75
##   average neighbourhood size:           2.50
##   average branching factor:             0.38
##
##   learning algorithm:                   Hill-Climbing
##   score:                                BIC (disc.)
##   penalization coefficient:             4.258597
##   tests used in the learning procedure: 130
##   optimized:                            TRUE
##
##   Bayesian network learned via Score-based methods
##
##   model:
##     [partially directed graph]
##   nodes:                                8
##   arcs:                                 7
##     undirected arcs:                    2
##     directed arcs:                      5
##   average markov blanket size:          2.25
##   average neighbourhood size:           1.75
##   average branching factor:             0.62
##
##   learning algorithm:                   Hill-Climbing
##   score:                                Bayesian Dirichlet (BDe)
##   graph prior:                          Uniform
##   imaginary sample size:                1
##   tests used in the learning procedure: 1848
##   optimized:                            TRUE
##
##   Bayesian network learned via Score-based methods
##
##   model:
##     [partially directed graph]
##   nodes:                                8
##   arcs:                                 19
##     undirected arcs:                    1
##     directed arcs:                      18
##   average markov blanket size:          5.75
##   average neighbourhood size:           4.75
##   average branching factor:             2.25
##
##   learning algorithm:                   Hill-Climbing
##   score:                                Bayesian Dirichlet (BDe)
##   graph prior:                          Uniform
```

```
##   imaginary sample size:                 100
##   tests used in the learning procedure:  1734
##   optimized:                             TRUE
```

And vstructs output of all hill climbing algorithms,listed below.

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

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

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

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

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

1.6 comment

We found that when we run the same code it will generate several different results even if we just recompile the Rmd file. (Different arc sets/number of directed/undirected arcs/ True). For those HL algorithms with the same score, same vstructs, all.equal=TRUE, their cpdag's [tests used in the learning procedure] still can be different.

We also found that when we increase the restart value from 1 to 100, the result does not change significantly, maybe just because the model is relatively simple. Since restart will help the model jump out of the local optimal, we will use a relatively big number to help the model find the global optimal.(we set 100 here).

And for iss, if we increase the value to a big number, say 100, we can find the model gets very complicated. And [tests used in the learning procedure] in cpdag's output get much bigger than others, it need more data or steps to train or to optimize the score and this makes sense.

Regarding the equivalent of two BN, they need to meet 2 conditions:

1) They have the same unshielded colliders
2) They have the same adjacencies

When we call the hill climbing algorithm, it will run from an empty DAG, and then add/remove/reverse edges to improve the score. But the hill climbing algorithm is a local search algorithm, it can not guarantee to find the global optimal solution. So if the algorithm is stuck in a local optimal, even when we set a restart value to help it jump out of the trap, it still can not guarantee to find the same optimal(or global optimal) result every time.

Exercises 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 elimi- nation and cluster trees. There are also approximate algorithms for when the exact ones are too demanding computationally. For exact inference, you may need the func- tions bn.fit and as.grain from the bnlearn package, and the functions compile, setEvidence and querygrain from the package gRain. For approximate infer- ence, you may need the functions cpquery or cpdist, prop.table and table from the bnlearn package.

Answer:

We will set score to bic and set restart to a relative big number ,say 100. We also need to define a function to calculate the confusion matrix for the later use.

```
############################  Code For Exercises 2 ############################
# define a hill_climbing_prediction and use it to predict the value of S
# and calculate the confusion matrix

# we split the data into 80% train and 20% test
set.seed(12345)
train_index <- sample(1:nrow(asia), 0.8 * nrow(asia))
train_data <- asia[train_index, ]
test_data <- asia[-train_index, ]

# learn the BN structure
bn_structure <- hc(train_data, score = "bic", restart = 100)

# define a function to calculate the confusion matrix
# input: test_data, training_data, hc_model, columns_index, column_nodes
hill_climbing_prediction <- function(test_data, training_data, hc_model, columns_index, column_nodes) {
  # learn the Bayesian network
  hc_fit <- bn.fit(x = hc_model, data = train_data)

  # convert hc_fit to a gRain object and then compile, the compiled object is used for inference
  bn_grain <- as.grain(hc_fit)

  # we convert the test data to a data frame, and convert all values to character
  test_data_char <- as.data.frame(lapply(test_data, as.character))

  bn_grain_compile <- compile(bn_grain)

  # define a vector to restore the prediction result
  predictions <- c()

  for (i in 1:nrow(test_data)) {
```

```
    # Create an evidence object
    # for nodes
    evidence <- setEvidence(object = bn_grain_compile,
                            nodes = colnames(test_data)[-columns_index],
                            states = test_data_char[i,-columns_index])

    # query the interest node with type="marginal"
    # and we need to predict S according to the question requirement
    potentials_list <- querygrain(object = evidence,
                                  nodes = column_nodes,
                                  type = "marginal")$S

    # based on the potentials_list, we get the most likely class
    predictions[i] <- ifelse(potentials_list[1] > potentials_list[2], "no", "yes")
  }

  # return confusion matrix of the prediction and the true value
  confusion_matrix <- table(predictions, test_data[,column_nodes])

}

# predict the value based on the defined hill climbing prediction function
q2_confusion_matrix <- hill_climbing_prediction(test_data = test_data,
                                                training_data = train_data,
                                                hc_model = bn_structure,
                                                columns_index = 2,
                                                column_nodes = "S")


# true Asia BN, provided in the exercise statement.
dag <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

# predict the value based on true Asia BN
q2_true_confusion_matrix <- hill_climbing_prediction(test_data = test_data,
                                                     training_data = train_data,
                                                     hc_model = dag,
                                                     columns_index = 2,
                                                     column_nodes = "S")

# print the confusion matrix
cat("The confusion matrix of the hill climbing algorithm is:\n")
```

```
## The confusion matrix of the hill climbing algorithm is:
```

```
print(q2_confusion_matrix)
```

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

```
cat("The confusion matrix of the true Asia BN is:\n")
```

```
## The confusion matrix of the true Asia BN is:
```

```r
print(q2_true_confusion_matrix)
```

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

From the output of the 2 confusion matrices, we see two identical confusion matrices. That means the hill climbing algorithm can predict the value of S pretty well.

Exercises 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.

Answer:

We will use the mb function to get the Markov blanket of S, and then use the Markov blanket to predict the value of S using the predefined function.

After check the confusion matrix of the Markov blanket of S and the true Asia BN output below, we found they are same.

```r
############################  Code For Exercises 3 ############################
# use predefined function to predict the value of S based on the Markov blanket of S
# and calculate the confusion matrix

# get the Markov blanket of S ,using the hill climbing algorithm in step 2
markov_blanket <- mb(x = bn_structure,node = "S")

# remove the columns index of markov_blanket
markov_blanket_index <- which(!(colnames(test_data) %in% markov_blanket))
#print(markov_blanket_index)

# predict the value based on the defined hill climbing prediction function
q3_confusion_matrix <- hill_climbing_prediction(test_data = test_data,
                                                training_data = train_data,
                                                hc_model = bn_structure,
                                                columns_index = markov_blanket_index,
                                                column_nodes = "S")

# print the confusion matrix of the Markov blanket of S and true Asia BN
cat("The confusion matrix of the Markov blanket of S is:\n")
```

```
## The confusion matrix of the Markov blanket of S is:
```

```r
print(q3_confusion_matrix)
```

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

```r
cat("The confusion matrix of the true Asia BN is:\n")
```

```
## The confusion matrix of the true Asia BN is:
```

```r
print(q2_true_confusion_matrix)
```

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

Exercises 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.

Answer:

According to the definition of Naive Bayes Classifier, we found that

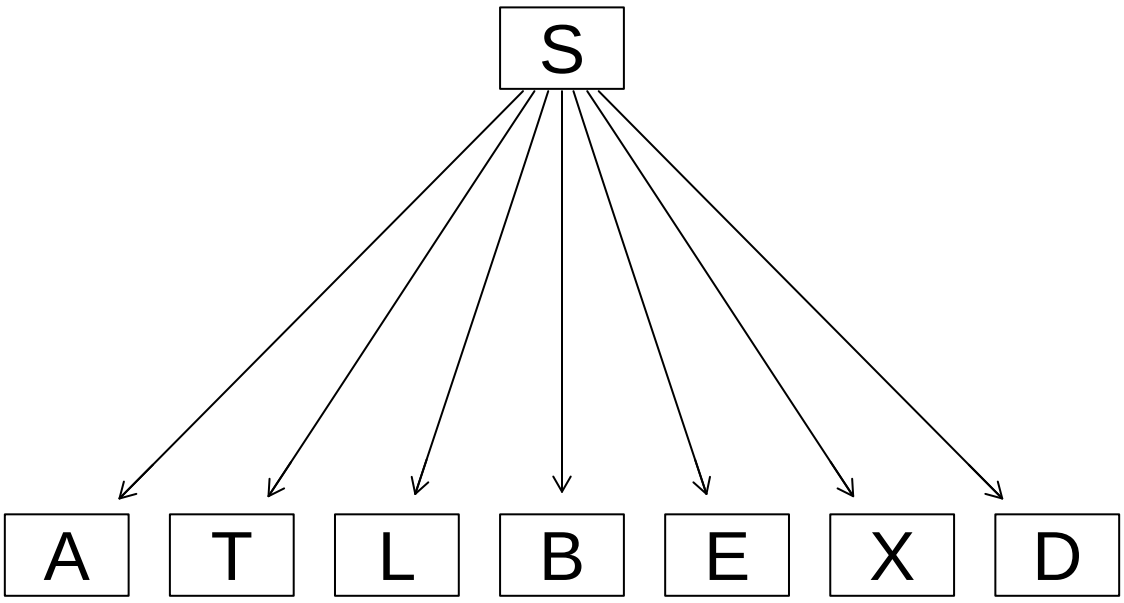
$$p(C_k|x_1, x_2, ..., x_n) = p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$$

That means we define our model as following and code as below.

[S][A|S][T|S][L|S][E|S][X|S][B|S][D|S]

```r
############################  Code For Exercises 4 ##############################
# create the Naive Bayes Classifier by hand, then use predefined function to
# predict the value of S based on the Markov blanket of S and calculate the
# confusion matrix

#dag_naive_bayes <- model2network("[S][A|S][T|S][L|S][E|S][X|S][B|S][D|S]")

# we create the BN by hand
dag_naive_bayes = empty.graph(c("A", "S", "T", "L", "B", "E", "X", "D"))

# set the arc(relations between nodes)
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")))

# assign the newly created arc.set to a bn object
arcs(dag_naive_bayes) = arc.set

# we plot the graph
graphviz.plot(dag_naive_bayes, main = "Naive Bayes Classifier")
```

Naive Bayes Classifier




```
# predict the value based on the defined hill climbing prediction function
naive_bayes_confusion_matrix <- hill_climbing_prediction(test_data = test_data,
                                                         training_data = train_data,
                                                         hc_model = dag_naive_bayes,
                                                         columns_index = 2,
                                                         column_nodes = "S")

# print the confusion matrix of naive bayes classifier and true Asia BN
cat("The confusion matrix of the Naive Bayes Classifier is:\n")
```

```
## The confusion matrix of the Naive Bayes Classifier is:
```

```
print(naive_bayes_confusion_matrix)
```

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

```
cat("The confusion matrix of the true Asia BN is:\n")
```

```
## The confusion matrix of the true Asia BN is:
```

```
print(q2_true_confusion_matrix)
```

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

```
# calc the accuracy of the Naive Bayes Classifier and the true Asia BN
accuracy_naive_bayes <- sum(diag(naive_bayes_confusion_matrix)) / sum(naive_bayes_confusion_matrix)
accuracy_true_asia_bn <- sum(diag(q2_true_confusion_matrix)) / sum(q2_true_confusion_matrix)
```

```
cat ("The accuracy of the Naive Bayes Classifier is:", accuracy_naive_bayes, "\n")
```

```
## The accuracy of the Naive Bayes Classifier is: 0.666
```

```
cat ("The accuracy of the true Asia BN is:", accuracy_true_asia_bn, "\n")
```

```
## The accuracy of the true Asia BN is: 0.703
```

The result below shows that the confusion matrix of the Naive Bayes Classifier is the not same as the true Asia BN. The accuracy of the Naive Bayes Classifier is also lower than the true Asia BN.

Exercises 5

Explain why you obtain the same or different results in the exercises 4.

Answer:

Since the our model get the same confusion matrix as the true asia BN, so we will use the true asia BN as the reference to explain the difference between the Naive Bayes Classifier and the true asia BN.

In the case of true BN, the data nodes are dependent on each other according to the "true" relationship between them, it will perform better by capturing these correlations between features. But in the case of Naive Bayes Classifier,we will assume that the data nodes are independent of each other,and it means that Naive Bayes Classifier will contain less information than the true BN model. This is why the Naive Bayes Classifier will have a lower accuracy than the true BN model.

Let's check the value of markov blanket of S in Our BN model, true Asia BN and Naive Bayes Classifier BN model.

```
## The Markov blanket of S in our model is: L B
```

```
## The Markov blanket of S in true Asia BN is: B L
```

```
## The Markov blanket of S in Naive Bayes Classifier is: A T L B E X D
```

The Markov blanket of a node in a Bayesian network consists of its parents, children, and the parents of its children.

According to the output above, we can find that Markov blanket of S in our model and the true model is same, which is B,L. But for Naive Bayes Classifier these are: A T L B E X D, which means A T E X D are extra nodes which is not reflect the true relationship between the nodes.And this is why the confusion matrix differs between the Naive Bayes Classifier and the true Asia BN/Our model.

Appendix: All code for this report

```
############################  Init code For Assignment ############################
rm(list = ls())
knitr::opts_chunk$set(echo = TRUE)
library(bnlearn)
library(gRain)
############################  Code For Exercises 1.1 ############################
# default parameters vs restart = 10

# Parameters
# 1. score:default:bic
#    1). bic:the Bayesian Information Criterion (BIC) score
#    2). bde:the logarithm of the Bayesian Dirichlet equivalent (uniform) score (BDeu)
# 2. restart:the number of random restarts(integer), default: 0
# 3. iss:the imaginary sample size used by the Bayesian Dirichlet scores
#   (bde, mbde, bds, bdj),default: 1
# 4. start:the initial structure of the network, default: empty graph(NULL)
data("asia")
hill_climbing_1 <- hc(asia)
hill_climbing_2 <- hc(asia, restart = 10)
# calc the score of the two hill climbing algorithms
cat("The score of hill climbing algorithm 1 and 2 are:", score(hill_climbing_1, asia),
    "and" ,score(hill_climbing_2, asia))
# check equivalent using all.equal
cat("Equivalency between two hill climbing algorithms:",
    all.equal(hill_climbing_1, hill_climbing_2))
# plot the graph
par(mfrow = c(1, 2))
graphviz.compare(hill_climbing_1, hill_climbing_2)
############################  Code For Exercises 1.2 ############################
# default parameters vs initial structure(random)

initial_structure <- random.graph(names(asia), num = 1, method = "ordered")
hill_climbing_3 <- hc(asia, start = initial_structure)
# calc the score of the two hill climbing algorithms
cat("The score of hill climbing algorithm 1 and 3 are:", score(hill_climbing_1, asia),
    "and" ,score(hill_climbing_3, asia))
# check equivalent using all.equal
cat("Equivalency between two hill climbing algorithms:",
    all.equal(hill_climbing_1, hill_climbing_3))
# plot the graph
par(mfrow = c(1, 2))
graphviz.compare(hill_climbing_1, hill_climbing_3)
############################  Code For Exercises 1.3 ############################
# [restart = 100, BIC] vs [restart = 100, BDE]

# change score value
hill_climbing_4 <- hc(asia,score = "bde", restart = 100)
# calc the score of the two hill climbing algorithms
cat("The score of hill climbing algorithm 2 and 4 are:", score(hill_climbing_2, asia),
    "and" ,score(hill_climbing_4, asia))
# check equivalent using all.equal
cat("Equivalency between two hill climbing algorithms:",
```

```
    all.equal(hill_climbing_2, hill_climbing_4))
# plot the graph
par(mfrow = c(1, 2))
graphviz.compare(hill_climbing_2, hill_climbing_4)
############################  Code For Exercises 1.4 ############################
# [BDE, restart = 100, ISS=1(default)] vs [BDE, restart = 100, iss = 100]

# change iss value
hill_climbing_5 <- hc(asia,score = "bde", restart = 100, iss = 100)
# calc the score of the two hill climbing algorithms
cat("The score of hill climbing algorithm 4 and 5 are:", score(hill_climbing_4, asia),
    "and" ,score(hill_climbing_5, asia))
# check equivalent using all.equal
cat("Equivalency between two hill climbing algorithms:",
    all.equal(hill_climbing_4, hill_climbing_5))
# plot the graph
par(mfrow = c(1, 2))
graphviz.compare(hill_climbing_4, hill_climbing_5)
############################  Code For Exercises 1.5 ############################
# output of cpdag

cpdag(hill_climbing_1)
cpdag(hill_climbing_2)
cpdag(hill_climbing_3)
cpdag(hill_climbing_4)
cpdag(hill_climbing_5)
# output of vstructs
vstructs(hill_climbing_1)
vstructs(hill_climbing_2)
vstructs(hill_climbing_3)
vstructs(hill_climbing_4)
vstructs(hill_climbing_5)
############################  Code For Exercises 2 ############################
# define a hill_climbing_prediction and use it to predict the value of S
# and calculate the confusion matrix

# we split the data into 80% train and 20% test
set.seed(12345)
train_index <- sample(1:nrow(asia), 0.8 * nrow(asia))
train_data <- asia[train_index, ]
test_data <- asia[-train_index, ]

# learn the BN structure
bn_structure <- hc(train_data, score = "bic", restart = 100)

# define a function to calculate the confusion matrix
# input: test_data, training_data, hc_model, columns_index, column_nodes
hill_climbing_prediction <- function(test_data, training_data, hc_model, columns_index, column_nodes) {
  # learn the Bayesian network
  hc_fit <- bn.fit(x = hc_model, data = train_data)

  # convert hc_fit to a gRain object and then compile, the compiled object is used for inference
  bn_grain <- as.grain(hc_fit)
```

```
  # we convert the test data to a data frame, and convert all values to character
  test_data_char <- as.data.frame(lapply(test_data, as.character))

  bn_grain_compile <- compile(bn_grain)

  # define a vector to restore the prediction result
  predictions <- c()

  for (i in 1:nrow(test_data)) {

    # Create an evidence object
    # for nodes
    evidence <- setEvidence(object = bn_grain_compile,
                            nodes = colnames(test_data)[-columns_index],
                            states = test_data_char[i,-columns_index])

    # query the interest node with type="marginal"
    # and we need to predict S according to the question requirement
    potentials_list <- querygrain(object = evidence,
                                  nodes = column_nodes,
                                  type = "marginal")$S

    # based on the potentials_list, we get the most likely class
    predictions[i] <- ifelse(potentials_list[1] > potentials_list[2], "no", "yes")
  }

  # return confusion matrix of the prediction and the true value
  confusion_matrix <- table(predictions, test_data[,column_nodes])

}

# predict the value based on the defined hill climbing prediction function
q2_confusion_matrix <- hill_climbing_prediction(test_data = test_data,
                                                training_data = train_data,
                                                hc_model = bn_structure,
                                                columns_index = 2,
                                                column_nodes = "S")


# true Asia BN, provided in the exercise statement.
dag <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

# predict the value based on true Asia BN
q2_true_confusion_matrix <- hill_climbing_prediction(test_data = test_data,
                                                     training_data = train_data,
                                                     hc_model = dag,
                                                     columns_index = 2,
                                                     column_nodes = "S")

# print the confusion matrix
cat("The confusion matrix of the hill climbing algorithm is:\n")
print(q2_confusion_matrix)
cat("The confusion matrix of the true Asia BN is:\n")
```

```r
print(q2_true_confusion_matrix)
############################  Code For Exercises 3 ############################
# use predefined function to predict the value of S based on the Markov blanket of S
# and calculate the confusion matrix

# get the Markov blanket of S ,using the hill climbing algorithm in step 2
markov_blanket <- mb(x = bn_structure,node = "S")

# remove the columns index of markov_blanket
markov_blanket_index <- which(!(colnames(test_data) %in% markov_blanket))
#print(markov_blanket_index)

# predict the value based on the defined hill climbing prediction function
q3_confusion_matrix <- hill_climbing_prediction(test_data = test_data,
                                                training_data = train_data,
                                                hc_model = bn_structure,
                                                columns_index = markov_blanket_index,
                                                column_nodes = "S")

# print the confusion matrix of the Markov blanket of S and true Asia BN
cat("The confusion matrix of the Markov blanket of S is:\n")
print(q3_confusion_matrix)
cat("The confusion matrix of the true Asia BN is:\n")
print(q2_true_confusion_matrix)
############################  Code For Exercises 4 ############################
# create the Naive Bayes Classifier by hand, then use predefined function to
# predict the value of S based on the Markov blanket of S and calculate the
# confusion matrix

#dag_naive_bayes <- model2network("[S][A|S][T|S][L|S][E|S][X|S][B|S][D|S]")

# we create the BN by hand
dag_naive_bayes = empty.graph(c("A", "S", "T", "L", "B", "E", "X", "D"))

# set the arc(relations between nodes)
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")))

# assign the newly created arc.set to a bn object
arcs(dag_naive_bayes) = arc.set

# we plot the graph
graphviz.plot(dag_naive_bayes, main = "Naive Bayes Classifier")

# predict the value based on the defined hill climbing prediction function
naive_bayes_confusion_matrix <- hill_climbing_prediction(test_data = test_data,
                                                training_data = train_data,
                                                hc_model = dag_naive_bayes,
                                                columns_index = 2,
                                                column_nodes = "S")

# print the confusion matrix of naive bayes classifier and true Asia BN
```

```r
cat("The confusion matrix of the Naive Bayes Classifier is:\n")
print(naive_bayes_confusion_matrix)
cat("The confusion matrix of the true Asia BN is:\n")
print(q2_true_confusion_matrix)

# calc the accuracy of the Naive Bayes Classifier and the true Asia BN
accuracy_naive_bayes <- sum(diag(naive_bayes_confusion_matrix)) / sum(naive_bayes_confusion_matrix)
accuracy_true_asia_bn <- sum(diag(q2_true_confusion_matrix)) / sum(q2_true_confusion_matrix)

cat ("The accuracy of the Naive Bayes Classifier is:", accuracy_naive_bayes, "\n")
cat ("The accuracy of the true Asia BN is:", accuracy_true_asia_bn, "\n")
############################  Code For Exercises 5 ############################
# Calculate the markov blanket of S in our model, true Asia BN and Naive Bayes Classifier

# get the Markov blanket of S in our model
markov_blanket_our_model <- mb(x = bn_structure,node = "S")
# get the Markov blanket of S in true Asia BN
markov_blanket_true_asia_bn <- mb(x = dag,node = "S")
# get the Markov blanket of S in Naive Bayes Classifier
markov_blanket_naive_bayes <- mb(x = dag_naive_bayes,node = "S")

cat("The Markov blanket of S in our model is:", markov_blanket_our_model, "\n")
cat("The Markov blanket of S in true Asia BN is:", markov_blanket_true_asia_bn, "\n")
cat("The Markov blanket of S in Naive Bayes Classifier is:", markov_blanket_naive_bayes, "\n")
```