Advanced Machine Learning
LAB 1: Graphical Models

Mohammed Bakheet (mohba508)

14/09/2020

Contents

Question 1: **2**
- Using the same restart value: . . . 2
- Using different restart values: . . . 3

Question 2: **3**
- Building the network: . . . 3
- Comparing the network with the actual network: . . . 4

Qeustion 3: **4**

Question 4: **5**

Question 5: **7**

Appendix **7**

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 in- cluded in the bnlearn package. To load the data, run data("asia").
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.

Using the same restart value:

```
library(bnlearn)
```

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

```
## The following object is masked from 'package:stats':
##
##     sigma
```

```
library(gRain)
```

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

```
##
## Attaching package: 'gRbase'
```

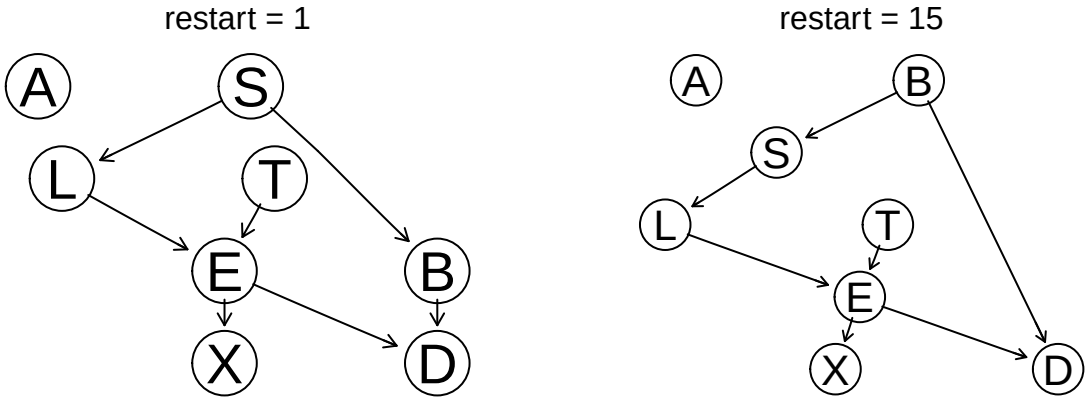
```
## The following objects are masked from 'package:bnlearn':
##
##     ancestors, children, parents
```

```
library(ggplot2)

data("asia")
#Fitting the parameters of a Bayesian network

hill_climbing = function(runs){

  for (i in 1:runs){
    dag = hc(asia, restart = 4)
    bn_arc = dag$arcs
    cat('The architecture for iteration ', i , ' is :\n', bn_arc , '\n')
  }
}

hill_climbing(4)
```

```
## The architecture for iteration  1  is :
##  B L E S E S T D E X B D L E
## The architecture for iteration  2  is :
##  B L E S T S E D E X B E L D
## The architecture for iteration  3  is :
##  B S T S L E E D B E L E X D
## The architecture for iteration  4  is :
##  L E S T S B E E X B E L D D
```

We can see from these values that the architecture of Bayesian Network changes for each run, and that is due to the fact that the Hill-Climbing algorithm chooses the next graph randomly, it gives the same probability for all graphs because it doesn't konw what the actual graph is.

Using different restart values:



When we compare the two graphs with different restart values, the graphs architecture is different, the restart graphs are chosen randomly, that's why the algorithm ends up in different architecture. the direction from node (S) to node (L) is different in the two graphs.

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]").

Hint: You already know the Lauritzen-Spiegelhalter algorithm for inference in BNs, which is an exact algorithm. 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, setFinding and querygrain from the package gRain. For approximate inference, you may need the functions prop.table, table and cpdist from the bnlearn package. When you try to load the package gRain, you will get an error as the pack- age RBGL cannot be found. You have to install this package by running the following two commands (answer no to any offer to update packages):
source("https://bioconductor.org/biocLite.R") biocLite("RBGL")

```
#Shuffling the data and dividing it into training and testing
n = dim(asia)[1]
id=sample(1:n, floor(n*0.8))
train_asia = asia[id,]
test_asia = asia[-id,]
actual_dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
```

Building the network:

```
## Fitting the BN network >>>>
```

```
## The misclassification rate is:
##  0.311
```

```
##          Actual
## Predicted    no   yes
##       no  0.309 0.143
##       yes 0.168 0.380
```

Comparing the network with the actual network:

```
## The actual predicted vaues for S are :
```

```
## Fitting the BN network >>>>
## The misclassification rate is:
##  0.311
```

```
##          Actual
## Predicted    no   yes
##       no  0.309 0.143
##       yes 0.168 0.380
```

Qeustion 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.

```
learned_bn_markov = function(){
  #Creating a BN object using hill-climbing
  dag = hc(train_asia)
  fit = bn.fit(dag, train_asia, method = "bayes")
  #Fitting the model with the training data
  cat('Fitting the BN network >>>>', '\n')

  #Converting the bn.fit object to grain object
  grained_fit = as.grain(fit)
  compiled_grain = compile(object = grained_fit)

  mbs_predict = c()
  for (i in 1:dim(test_asia)[1]){
    ev = c()
    for(j in mb(fit,'S')){

      if(test_asia[i,j]=="no"){
        ev = c(ev,"no")
      }
      else{
        ev = c(ev,"yes")
      }
    }
    set_evidence = setEvidence(compiled_grain, nodes = mb(fit,'S'), states =ev)
    s_predicted = querygrain(set_evidence)
    s_numeric = lapply(s_predicted, as.numeric)
    class_prediction =  ifelse(s_numeric$S[2] > s_numeric$S[1],"yes","no")
```

```
    mbs_predict[i] = class_prediction
  }

  confusion = table("Predicted" = mbs_predict, "Actual" = test_asia$S)
  misclass = 1-sum(diag(confusion))/sum(confusion)
  cat('The misclassification rate is: \n', misclass)
  prop.table(x = table("Predicted" = mbs_predict, "Actual" = test_asia$S))

}
learned_bn_markov()
```

```
## Fitting the BN network >>>>
## The misclassification rate is:
##  0.311
```

```
##          Actual
## Predicted    no   yes
##       no  0.309 0.143
##       yes 0.168 0.380
```

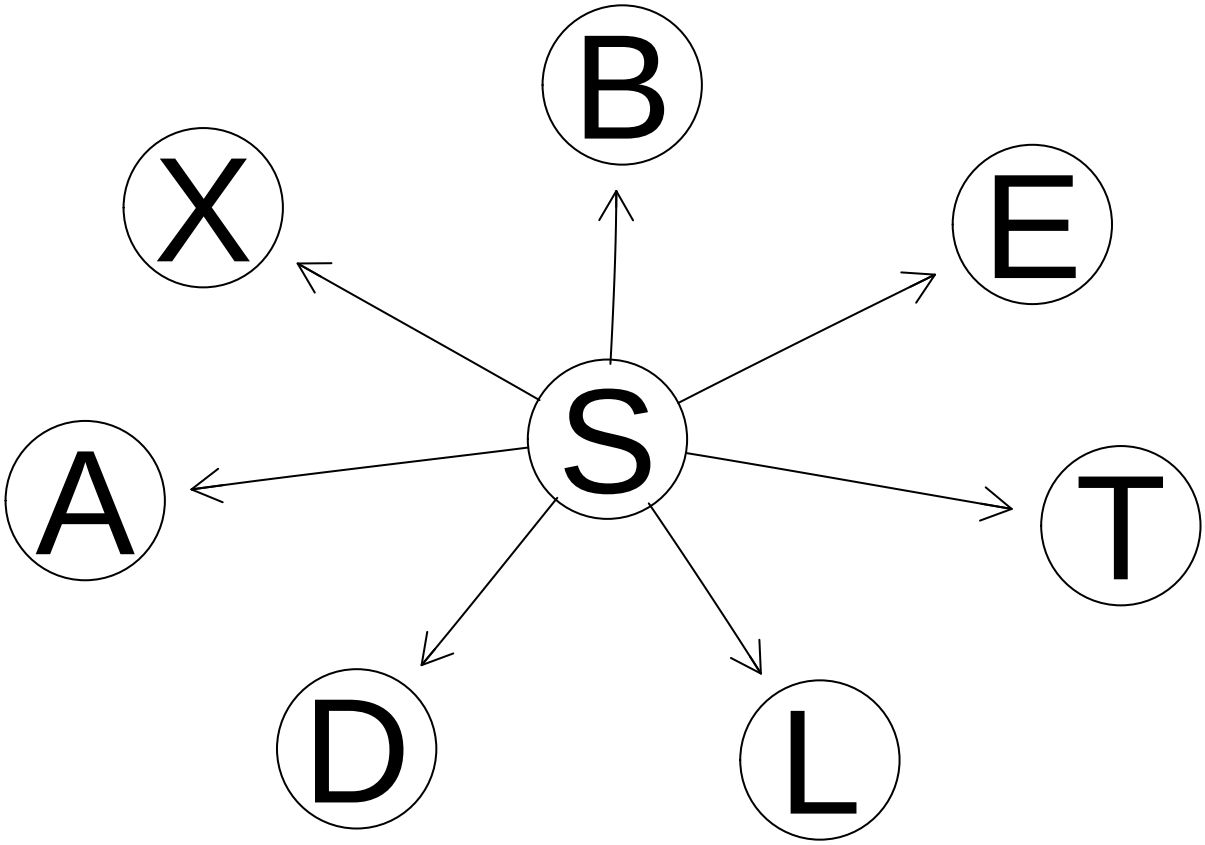
The result is the same because S is independent of all other nodes, except for its parents and S is a parent of L and B.

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.
Hint: Check http://www.bnlearn.com/examples/dag/ to see how to create a BN by hand

```
learned_bn_naive = function(){
e = empty.graph(c("A","T","L","B","E","X","D","S"))
arc.set = matrix(c("S", "T",
                   "S", "L",
                   "S", "B",
                   "S", "D",
                   "S", "A",
                   "S", "E",
                   "S","X"),
                 ncol = 2, byrow = TRUE,
                 dimnames = list(NULL, c("from", "to")))
arcs(e) = arc.set
fit = bn.fit(e,data = train_asia)
graphviz.plot(fit, layout = "neato")
grained_fit = as.grain(fit)
compiled_grain = compile(object = grained_fit)

s_predict = c()
for (i in 1:dim(test_asia)[1]){
  ev = c()
  for(j in c("A","T","L","B","E","X","D")){

    if(test_asia[i,j]=="no"){
      ev = c(ev,"no")
```

```
    }
    else{
      ev = c(ev,"yes")
    }
  }
  set_evidence = setEvidence(compiled_grain, nodes = c("A","T","L","B","E","X","D"), states =ev)
  s_predicted = querygrain(set_evidence)
  s_numeric = lapply(s_predicted, as.numeric)
  class_prediction =  ifelse(s_numeric$S[2] > s_numeric$S[1],"yes","no")
  s_predict[i] = class_prediction
}

confusion = table("Predicted" = s_predict, "Actual" = test_asia$S)
misclass = 1-sum(diag(confusion))/sum(confusion)
cat('The misclassification rate is: \n', misclass)
prop.table(x = table("Predicted" = s_predict, "Actual" = test_asia$S))
}
learned_bn_naive()
```




```
## The misclassification rate is:
##  0.367
```

```
##          Actual
## Predicted    no   yes
##       no  0.333 0.223
##       yes 0.144 0.300
```

Question 5:

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

When using naive Bayes classifier, we assume that predictive variables are independent given the class variable, which is not correct in our actual network architecture, that's why the misclassification rate has increased when we assumed independence between predictive variables given the class variable (S).

Appendix

```
RNGversion('3.5.1')
knitr::opts_chunk$set(echo = TRUE)
library(bnlearn)
library(gRain)
library(ggplot2)

data("asia")
#Fitting the parameters of a Bayesian network

hill_climbing = function(runs){

  for (i in 1:runs){
    dag = hc(asia, restart = 4)
    bn_arc = dag$arcs
    cat('The architecture for iteration ', i , ' is :\n', bn_arc , '\n')
  }
}

hill_climbing(4)
par(mfrow=c(2,2))
hill_1 = hc(asia, score = 'bic', restart = 1)
graphviz.plot(hill_1, main = "restart = 1")

hill_2 = hc(asia, score = 'bic', restart = 15)
graphviz.plot(hill_2, main = "restart = 15")
#Shuffling the data and dividing it into training and testing
n = dim(asia)[1]
id=sample(1:n, floor(n*0.8))
train_asia = asia[id,]
test_asia = asia[-id,]
actual_dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
learedBN = function(actual = FALSE){

  #Creating a BN object using hill-climbing
  dag = hc(train_asia)
  fit = bn.fit(dag, train_asia, method = "bayes")
  #Fitting the model with the training data
  if (actual == TRUE){
    fit = bn.fit(actual_dag, train_asia, method = "bayes")
  }
  cat('Fitting the BN network >>>>', '\n')
  fit
```

```
  #Converting the bn.fit object to grain object
  grained_fit = as.grain(fit)
  grained_fit
  compiled_grain = compile(object = grained_fit)
  compiled_grain

  s_predict = c()
  for (i in 1:dim(test_asia)[1]){
    ev = c()
    for(j in c("A","T","L","B","E","X","D")){

      if(test_asia[i,j]=="no"){
        ev = c(ev,"no")
      }
      else{
        ev = c(ev,"yes")
      }
    }
    set_evidence = setEvidence(compiled_grain, nodes = c("A","T","L","B","E","X","D"), states =ev)
    s_predicted = querygrain(set_evidence)
    s_numeric = lapply(s_predicted, as.numeric)
    class_prediction =  ifelse(s_numeric$S[2] > s_numeric$S[1],"yes","no")
    s_predict[i] = class_prediction
  }


  confusion = table("Predicted" = s_predict, "Actual" = test_asia$S)
    misclass = 1-sum(diag(confusion))/sum(confusion)
  cat('The misclassification rate is: \n', misclass )
  prop.table(x = table("Predicted" = s_predict, "Actual" = test_asia$S))

}
learedBN()
cat('The actual predicted vaues for S are :\n')
learedBN(TRUE)
learned_bn_markov = function(){
  #Creating a BN object using hill-climbing
  dag = hc(train_asia)
  fit = bn.fit(dag, train_asia, method = "bayes")
  #Fitting the model with the training data
  cat('Fitting the BN network >>>>', '\n')

  #Converting the bn.fit object to grain object
  grained_fit = as.grain(fit)
  compiled_grain = compile(object = grained_fit)

  mbs_predict = c()
  for (i in 1:dim(test_asia)[1]){
    ev = c()
    for(j in mb(fit,'S')){

      if(test_asia[i,j]=="no"){
        ev = c(ev,"no")
```

```
      }
      else{
        ev = c(ev,"yes")
      }
    }
    set_evidence = setEvidence(compiled_grain, nodes = mb(fit,'S'), states =ev)
    s_predicted = querygrain(set_evidence)
    s_numeric = lapply(s_predicted, as.numeric)
    class_prediction =  ifelse(s_numeric$S[2] > s_numeric$S[1],"yes","no")
    mbs_predict[i] = class_prediction
  }

  confusion = table("Predicted" = mbs_predict, "Actual" = test_asia$S)
  misclass = 1-sum(diag(confusion))/sum(confusion)
  cat('The misclassification rate is: \n', misclass)
  prop.table(x = table("Predicted" = mbs_predict, "Actual" = test_asia$S))

}
learned_bn_markov()

learned_bn_naive = function(){
e = empty.graph(c("A","T","L","B","E","X","D","S"))
arc.set = matrix(c("S", "T",
                   "S", "L",
                   "S", "B",
                   "S", "D",
                   "S", "A",
                   "S", "E",
                   "S","X"),
                 ncol = 2, byrow = TRUE,
                 dimnames = list(NULL, c("from", "to")))
arcs(e) = arc.set
fit = bn.fit(e,data = train_asia)
graphviz.plot(fit, layout = "neato")
grained_fit = as.grain(fit)
compiled_grain = compile(object = grained_fit)

s_predict = c()
for (i in 1:dim(test_asia)[1]){
  ev = c()
  for(j in c("A","T","L","B","E","X","D")){

    if(test_asia[i,j]=="no"){
      ev = c(ev,"no")
    }
    else{
      ev = c(ev,"yes")
    }
  }
  set_evidence = setEvidence(compiled_grain, nodes = c("A","T","L","B","E","X","D"), states =ev)
  s_predicted = querygrain(set_evidence)
  s_numeric = lapply(s_predicted, as.numeric)
  class_prediction =  ifelse(s_numeric$S[2] > s_numeric$S[1],"yes","no")
```

```
  s_predict[i] = class_prediction
}

confusion = table("Predicted" = s_predict, "Actual" = test_asia$S)
misclass = 1-sum(diag(confusion))/sum(confusion)
cat('The misclassification rate is: \n', misclass)
prop.table(x = table("Predicted" = s_predict, "Actual" = test_asia$S))
}
learned_bn_naive()
```