Report Lab 1

Aman Kumar Nayak

9/13/2020

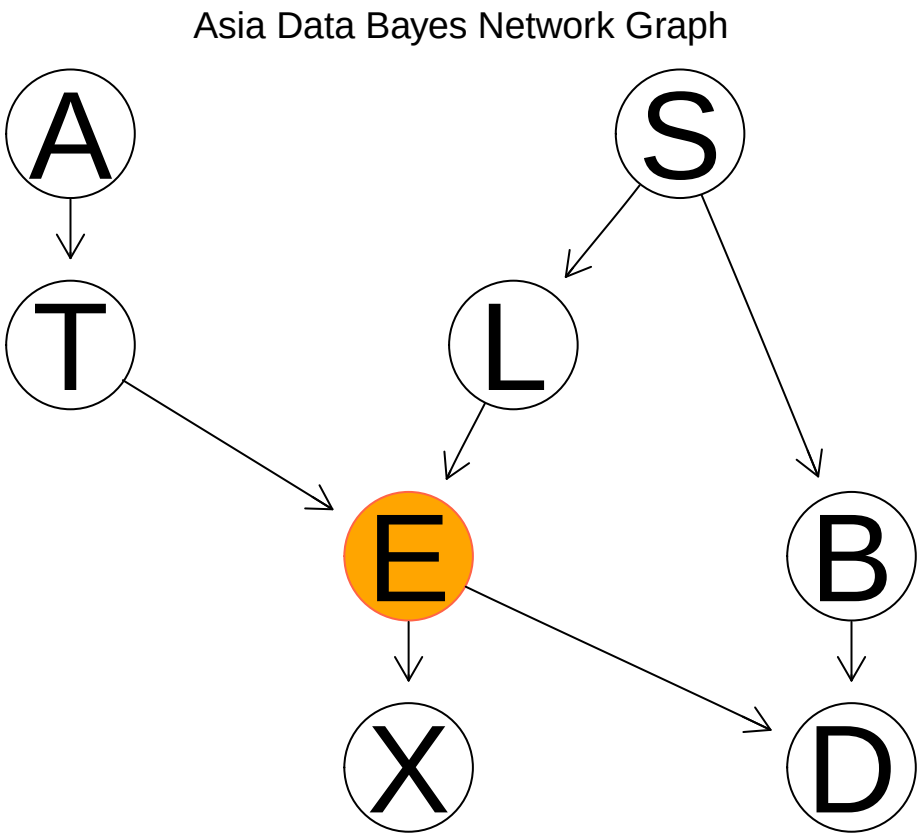
Data

The asia data set contains the following variables:

- D (dyspnoea), a two-level factor with levels yes and no.
- T (tuberculosis), a two-level factor with levels yes and no.
- L (lung cancer), a two-level factor with levels yes and no.
- B (bronchitis), a two-level factor with levels yes and no.
- A (visit to Asia), a two-level factor with levels yes and no.
- S (smoking), a two-level factor with levels yes and no.
- X (chest X-ray), a two-level factor with levels yes and no.
- E (tuberculosis versus lung cancer/bronchitis), a two-level factor with levels yes and no.

"Shortness-of-breath (dyspnoea) may be due to tuberculosis, lung cancer or bronchitis, or none of them, or more than one of them. A recent visit to Asia increases the chances of tuberculosis, while smoking is known to be a risk factor for both lung cancer and bronchitis. The results of a single chest X-ray do not discriminate between lung cancer and tuberculosis, as neither does the presence or absence of dyspnoea."

Standard learning algorithms are not able to recover the true structure of the network because of the presence of a node (E) with conditional probabilities equal to both 0 and 1. Monte Carlo tests seems to behave better than their parametric counterparts.

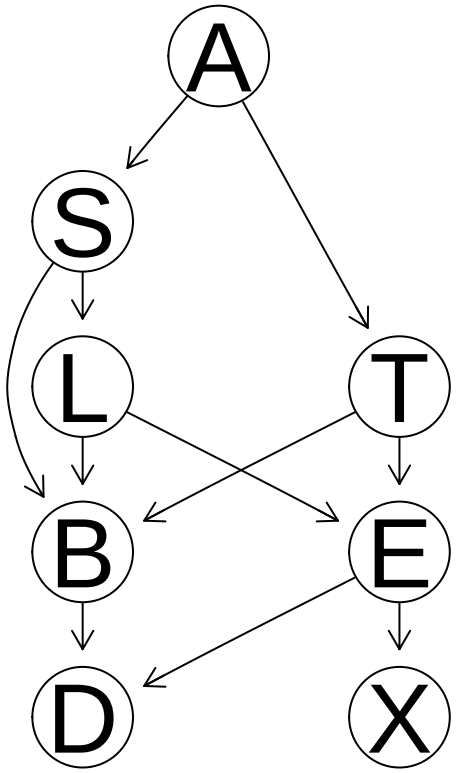


Question 1

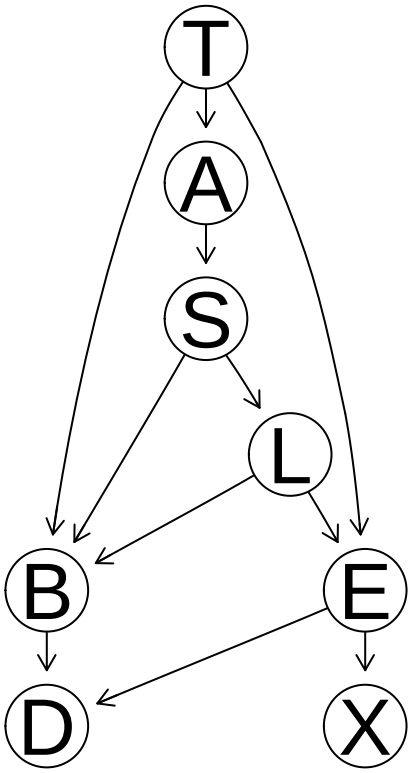
Task : Show Multiple run of the Hill-Climbing Algorithm can return non-equvalent Bayesian Network (BN) Structures.

Case 1 # Different Bayesian Network with same restart point, set as 10 and socre as AIC

BN with restart = 10



BN with restart = 10

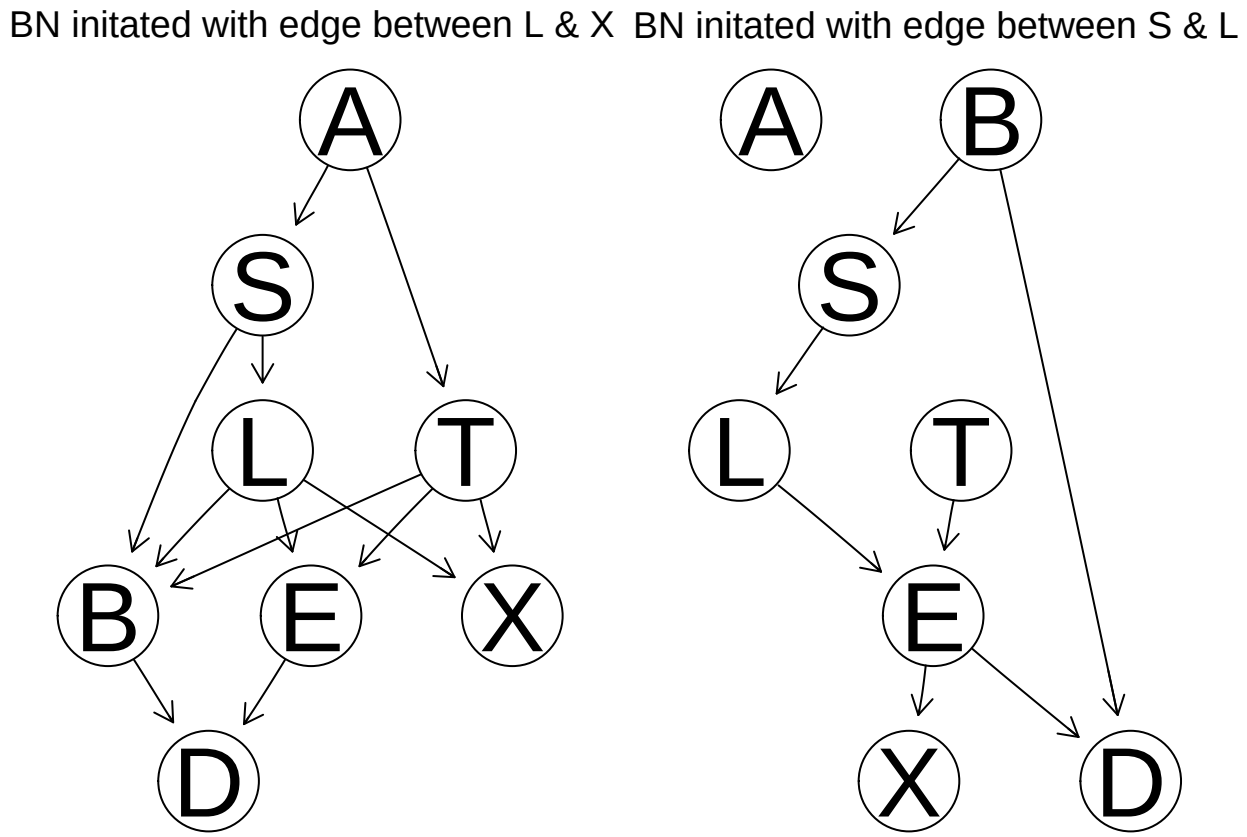



```
## Comparison of Networks

## Is both graph equal

## [1] "Different arc sets"

## true positive (tp) : Arcs which appear both in target and in current

##       from to
##  [1,] "B"  "D"
##  [2,] "L"  "E"
##  [3,] "S"  "B"
##  [4,] "S"  "L"
##  [5,] "E"  "D"
##  [6,] "A"  "S"
##  [7,] "L"  "B"
##  [8,] "T"  "E"
##  [9,] "T"  "B"
## [10,] "E"  "X"

## alse positive (fp) arcs, which appear in current but not in target

##      from to
## [1,] "T"  "A"
```

```
## false negative (fn) arcs, which appear in target but not in current

##      from to
## [1,] "A"  "T"
```

It can be seen that with the same restart point we get a different network as Hill Climbing (HC) algorithm is an optimization technique that belongs to the family of local search/heuristic method. Since it is a Heuristic approach, it does not guarantee an optimal result but an approximate result. It can be seen from the above plot, the obtained solution is quite close to the true network.

The algorithm starts with an arbitrary point and attempts to find an optimal approximate solution but it can be seen from the below plot when we start it with the wrong start point it can end up at a poor solution.

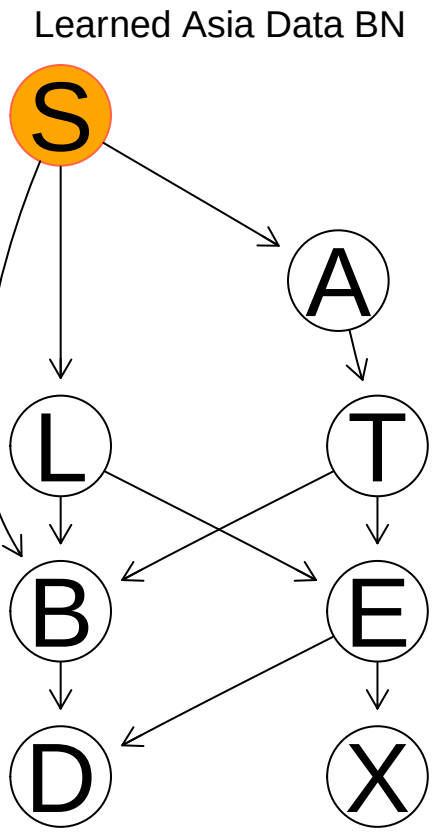


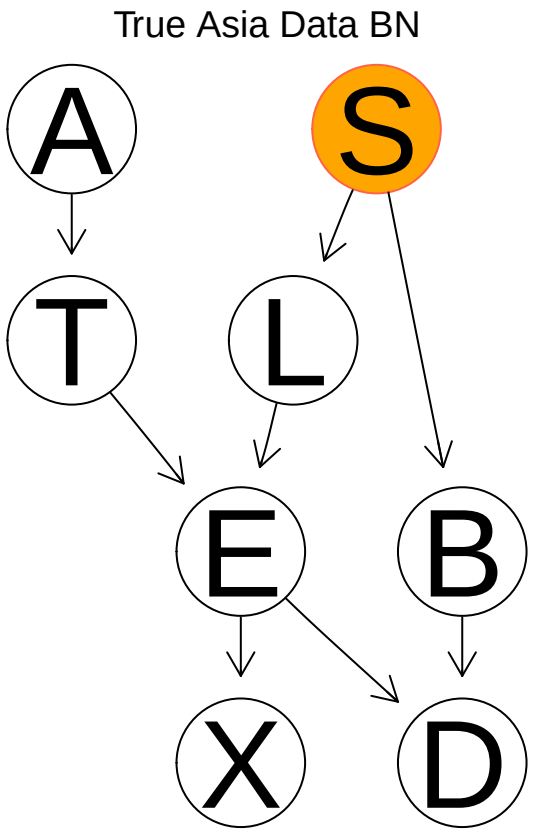

```
## Comparison of Networks

## Is both graph equal

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

## true positive (tp) : Arcs which appear both in target and in current

##      from to
## [1,] "L"  "E"
## [2,] "T"  "E"
## [3,] "E"  "D"
## [4,] "S"  "L"
## [5,] "B"  "D"
```

```
## alse positive (fp) arcs, which appear in current but not in target

##      from to
## [1,] "B"  "S"
## [2,] "E"  "X"

## false negative (fn) arcs, which appear in target but not in current

##      from to
## [1,] "T"  "B"
## [2,] "A"  "T"
## [3,] "L"  "B"
## [4,] "A"  "S"
## [5,] "S"  "B"
## [6,] "L"  "X"
## [7,] "T"  "X"
```

Question 2

Task : compute the posterior probability distribution of S for each case and classify it in the most likely class.






```
## Confusion Matrix for Learned BN

##      prediction
```

```
##          no   yes
##   no  0.322 0.146
##   yes 0.120 0.412

## Misclassification rate for Learned Model is  0.266

## Confusion Matrix for True BN

##      prediction
##          no   yes
##   no  0.322 0.146
##   yes 0.120 0.412

## Misclassification rate for True Model is  0.266
```

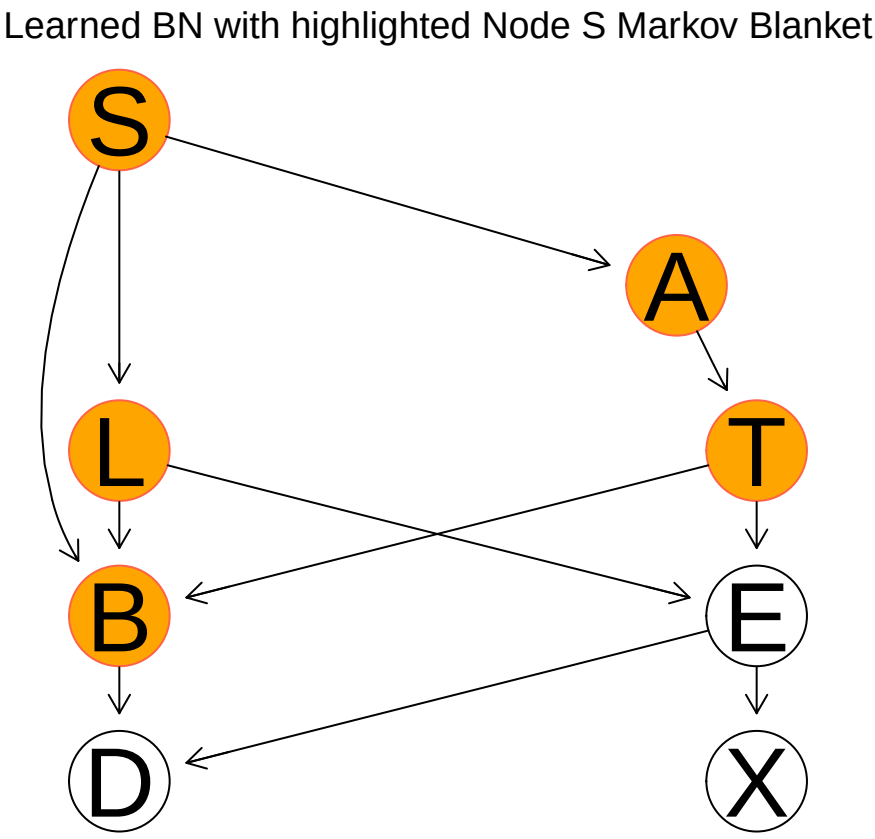
It can be seen that both Learned BN and True BN both are pretty close in Structure and, thus they have the same misclassification rate.

Question 3

Task : 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.

The Markov blanket of a node *S*, the set of nodes that completely separates S from the rest of the graph. Generally speaking, it is the set of nodes that includes all the knowledge needed to do inference on S, from estimation to hypothesis testing to prediction: the parents of S, the children of S, and those children's other parents.

```
## Markov Blanket node for node S is :  A T L B
```



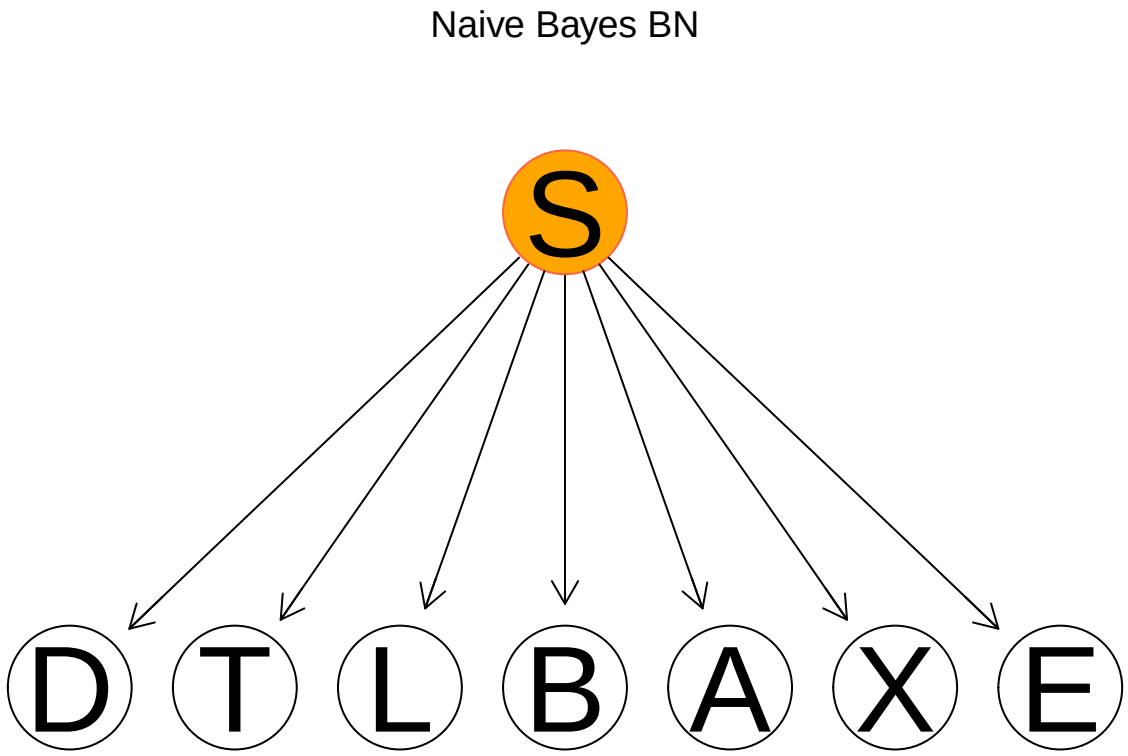
```
## Confusion Matrix for Markov Blanket of node S
```

```
##      prediction
##          no   yes
##   no  0.322 0.146
##   yes 0.120 0.412
```

```
## Misclassification rate for node S with Markov Blanket 0.266
```

Question 4

Task : Repeat the question 2 task using a naive Bayes classifier, i.e. the predictive variables are independent given the class variable.




```
## Confusion Matrix for Naive Bayes BN for S
```

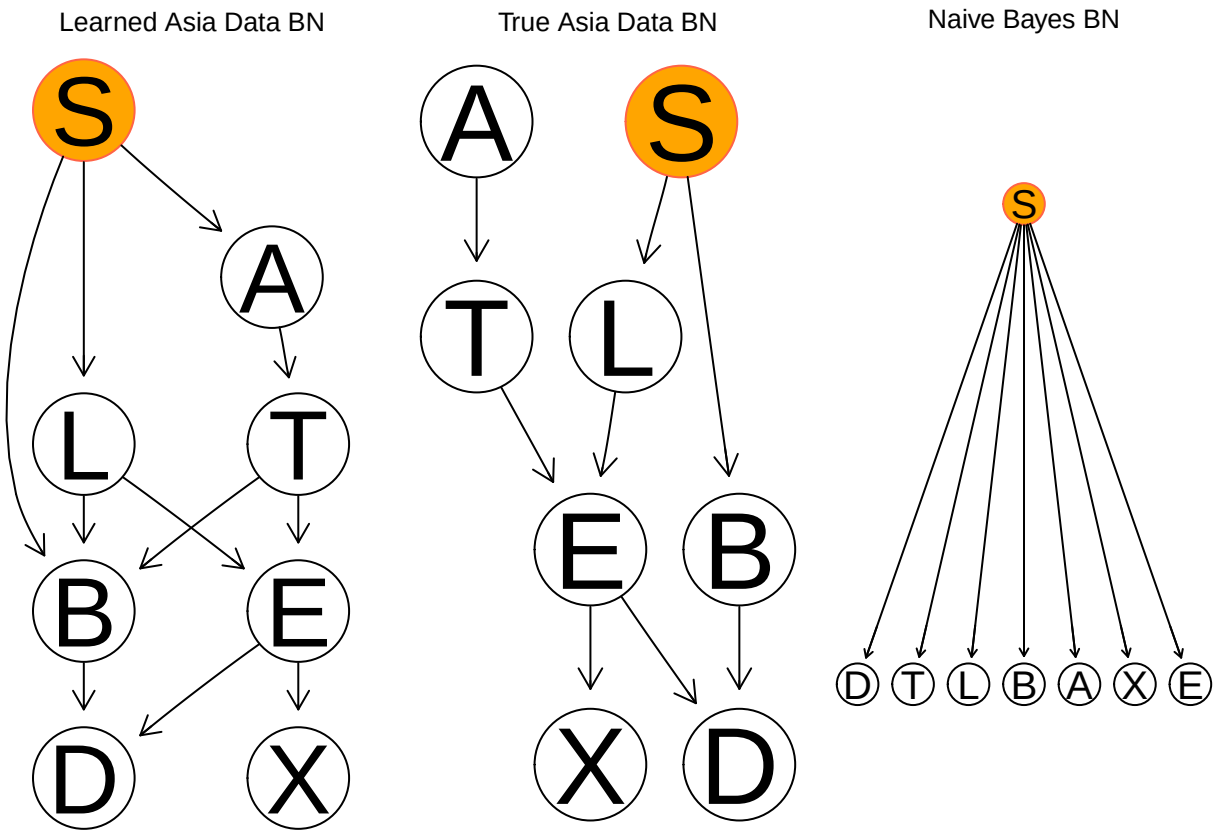
```
##      prediction
##          no   yes
##   no  0.349 0.119
##   yes 0.188 0.344
```

```
## Misclassification rate for node S Naive Bayes BN 0.307
```

Question 5

Task Explain result of Task 2 - 4

Comparison



The set of nodes that graphically isolates a target node from the rest of the DAG is called its Markov blanket, these nodes are the parents of S, the children of S, and those children's other parents. In Naive Bayes, we assume conditional independence of features(remaining nodes) for S thus conditional probability obtained in the case of Naive Bayes is different from other networks thus increased the misclassification rate.

Considering Learned Bayesian Network, in this S is Independent given L, B, A, T and this is the same for the Markov blanket case thus they both return the same result. Considering True Bayesian Network S is independent considering L and B and since this is pretty close to Learned Network we have also obtained, same result (but it needed to be informed that even though we have the same confusion matrix, in this case, their distribution was not same and same had been validated post analyzing predictions, result for this is not shared and code for same is available as commented line in function: fnPredict(returnDataFrame = True))

References

- Journal of Statistical Software
- Bayesian networks in R with the gRain package
- 732A96 Course Lecture Notes
- Inference in Bayesian Networks
- Understanding Bayesian Networks with Examples in R
- Wiki - Hill climbing
- Wiki Naive Bayes

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
library(gRain)
library(bnlearn)
library(Rgraphviz)

dag.True  = bnlearn::model2network(string = "[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

bnlearn::graphviz.plot(dag.True, main = "Asia Data Bayes Network Graph"
                       #,  layout = "neato",
                       ,highlight = list(nodes = "E" ,  col = "tomato",
                                         fill = "orange") )



#1st Structure
#graph without any inital structure is used here but different restart values

bn.hc1 = hc(asia , score = "aic" , restart = 10)
bn.hc2 = hc(asia , score = "aic" , restart = 10)

#intiate network
initiate = bnlearn::empty.graph(nodes = c("D" , "T", "L", "B", "A", "S", "X", "E"))
initiate = bnlearn::set.arc(initiate, from = "L" , to = "X" )
bn.hc3 = bnlearn::hc(asia , score = "aic" , restart = 10 , start = initiate)

initiate2 = bnlearn::empty.graph(nodes = c("D" , "T", "L", "B", "A", "S", "X", "E"))
initiate2 = bnlearn::set.arc(initiate2, from = "S" , to = "B" )
bn.hc4 = bnlearn::hc(asia , score = "bde" , restart = 10
                     , start = initiate2
                     )

par(mfrow = c(1, 2))
bnlearn::graphviz.plot(bn.hc1 , main = "BN with restart = 10")
bnlearn::graphviz.plot(bn.hc2, main = "BN with restart = 10")
cat("\n")
cat("Comparison of Networks")
cat("\n")
cat("Is both graph equal ")
cat("\n")
all.equal(bn.hc1 , bn.hc2)
cat("\n")

comp = bnlearn::compare(bn.hc1 , bn.hc2 , arcs = TRUE)

cat("true positive (tp) : Arcs which appear both in target and in current")
cat("\n")
comp$tp
cat("\n")

cat("alse positive (fp) arcs, which appear in current but not in target")
cat("\n")
comp$fp
```

```r
cat("\n")

cat("false negative (fn) arcs, which appear in target but not in current")
cat("\n")
comp$fn
cat("\n")




par(mfrow = c(1, 2))
bnlearn::graphviz.plot(bn.hc3 , main = "BN initated with edge between L & X")
bnlearn::graphviz.plot(bn.hc4 , main = "BN initated with edge between S & L")



cat("\n")
cat("Comparison of Networks")
cat("\n")


cat("Is both graph equal ")
cat("\n")
all.equal(bn.hc2 , bn.hc3)
cat("\n")

comp = bnlearn::compare(bn.hc3 , bn.hc4 , arcs = TRUE)

cat("true positive (tp) : Arcs which appear both in target and in current")
cat("\n")
comp$tp
cat("\n")

cat("alse positive (fp) arcs, which appear in current but not in target")
cat("\n")
comp$fp
cat("\n")

cat("false negative (fn) arcs, which appear in target but not in current")
cat("\n")
comp$fn
cat("\n")

#Divide Data in Train (80%) and Test(20%) of original data.

asiaData = bnlearn::asia

#Spliting Data in Train and Test
n = dim(asiaData)[1]
suppressWarnings(RNGversion("3.5.9"))
set.seed(12345)
id = sample(1:n , floor(n*0.8))
train = asiaData[id,]
```

```
test = asiaData[-id , ]

fnPredict = function(trainData , testData , DAGraph.bnObject , predNode = "S" ,
                     Method = "bayes" , markovB = FALSE , markovObj = NULL ,
                     returnDataFrame = FALSE){

  #fit bn model to training Data
  modelFit = bnlearn::bn.fit(x = DAGraph.bnObject , data = trainData ,
                             method = Method)

  # The main data structure in gRain is the grain class, which stores a fitted Bayesian network as a li
  # conditional probability tables (much like bnlearn's bn.fit objects) and makes it possible for setEv
  # and querygrain() to perform posterior inference via belief propagation.  #via docs

  # #converting to grain object and compiling
  compiled_Grain = gRbase::compile(object = bnlearn::as.grain(modelFit))
  #compiled_Grain = bnlearn::as.grain(modelFit)


  #fetch index of predNode in order to remove that while making predictions


  if(markovB == FALSE)
    {
      indx = which(colnames(testData) == predNode)
      nodesName = colnames(testData[-indx])
    }else{
      nodesName = as.vector(markovObj)
    }

  prediction = numeric(nrow(testData))

  #Added to see changing probabilities such that we can validate confusion matrix
  if(returnDataFrame == TRUE)
    dfPredTest = data.frame(matrix(data = NA , ncol = 2 , nrow = nrow(testData)))


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

    #In setEvidence function
    #nodes : A vector of nodes; those nodes for which the (conditional) distribution is requested.
    #states : A vector of states (of the nodes given by 'nodes')

    #fetching ith row of data set to predicted without target
    #extending logic for markov blanket case model

    if(markovB == FALSE){
      testDataRow.State = as.vector(unname(unlist(testData[i, -indx])))

    }else{
      testDataRow.State = as.vector(unname(unlist(testData[i, nodesName])))
    }
```

```
    # if(i == 1)
    #   print(testDataRow.State)

    Evidence_Grain = gRain::setEvidence(object = compiled_Grain ,
                                        nodes = nodesName ,
                                        states = testDataRow.State
                                        )

    queryGrain = querygrain(object = Evidence_Grain,
                            #object = compiled_Grain , evidence = Evidence_Grain ,
                             nodes = predNode ,
                             type = "marginal"
                            )

    qG = as.vector(unname(unlist(queryGrain)))
    #qG[1] = No and qG[2] = Yes
    #Added for visual validation of result @removed from final result
    if(returnDataFrame == TRUE)
    {
      dfPredTest[i,] = qG
    }

    prediction[i] = ifelse(qG[2] > qG[1]   ,  "yes" , "no")

  }#for

  #confusion matrix
  confusionMatrix = prop.table(table(test$S , prediction))

  if(returnDataFrame == TRUE)
  {
    return(list("Cmat" = confusionMatrix , "df" = dfPredTest))
  }else {
          return(confusionMatrix)
        }

}#fnPredict


#plot both Learned and True Graph
#creating DAG
set.seed(12345)
initiate = bnlearn::empty.graph(nodes = c("D" , "T", "L", "B", "A",  "X", "E" , "S"))
initiate = bnlearn::set.arc(initiate, from = "S" , to = "L" )
bn.hc = bnlearn::hc(asia , score = "aic" , restart = 10 , start = initiate )

#True
dag.True  = bnlearn::model2network(string = "[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")


par(mfrow = c(1,2))
bnlearn::graphviz.plot(bn.hc, main = "Learned Asia Data BN"
                       #,  layout = "neato",
```

```
                        ,highlight = list(nodes = "S" ,  col = "tomato",
                                          fill = "orange") )
bnlearn::graphviz.plot(dag.True, main = "True Asia Data BN"
                        #,  layout = "neato",
                        ,highlight = list(nodes = "S" ,  col = "tomato",
                                          fill = "orange") )

customModel = fnPredict(trainData = train , testData = test , DAGraph.bnObject = bn.hc ,
                         predNode = "S" , Method = "bayes" , returnDataFrame = FALSE)
cat("\n")
cat("Confusion Matrix for Learned BN")
cat("\n")
customModel
cat("\n")

MiscalLearned = 1 - sum(diag(customModel))/sum(customModel)
cat("Misclassification rate for Learned Model is " , MiscalLearned)
cat("\n")

trueModel = fnPredict(trainData = train , testData = test , DAGraph.bnObject = dag.True ,
                      predNode = "S" , Method = "bayes" , returnDataFrame = FALSE)
cat("\n")
cat("Confusion Matrix for True BN")
cat("\n")
trueModel
cat("\n")
MiscalTrueModel = 1 - sum(diag(trueModel))/sum(trueModel)
cat("Misclassification rate for True Model is " , MiscalTrueModel)
cat("\n")

bn.mb = mb(bn.hc , node = "S")

cat("Markov Blanket node for node S is : " , bn.mb)
cat("\n")
bnlearn::graphviz.plot(bn.hc, main = "Learned BN with highlighted Node S Markov Blanket"
                        #,  layout = "neato",
                        ,highlight = list(nodes = c("S" , bn.mb) ,  col = "tomato",
                                          fill = "orange") )

cat("\n")
MarkovBNPred = fnPredict(trainData = train , testData = test , DAGraph.bnObject = bn.hc
                  , predNode = "S" , Method = "mle" , markovB = TRUE ,
                 markovObj = bn.mb , returnDataFrame = FALSE)

cat("Confusion Matrix for Markov Blanket of node S")
cat("\n")
MarkovBNPred
cat("\n")
MiscalMB = 1 - sum(diag(MarkovBNPred))/sum(MarkovBNPred)
cat("Misclassification rate for node S with Markov Blanket" , MiscalMB)
cat("\n")
```

```r
NaiveBayesBN = bnlearn::empty.graph(nodes = c("D" , "T", "L", "B", "A",  "X", "E" , "S"))
NaiveBayesBN = bnlearn::set.arc(NaiveBayesBN, from = "S" , to = "D" )
NaiveBayesBN = bnlearn::set.arc(NaiveBayesBN, from = "S" , to = "T" )
NaiveBayesBN = bnlearn::set.arc(NaiveBayesBN, from = "S" , to = "L" )
NaiveBayesBN = bnlearn::set.arc(NaiveBayesBN, from = "S" , to = "B" )
NaiveBayesBN = bnlearn::set.arc(NaiveBayesBN, from = "S" , to = "A" )
NaiveBayesBN = bnlearn::set.arc(NaiveBayesBN, from = "S" , to = "X" )
NaiveBayesBN = bnlearn::set.arc(NaiveBayesBN, from = "S" , to = "E" )

# NaiveBayesBN = bnlearn::set.arc(NaiveBayesBN, from = "D" , to = "S" )
# NaiveBayesBN = bnlearn::set.arc(NaiveBayesBN, from = "T" , to = "S" )
# NaiveBayesBN = bnlearn::set.arc(NaiveBayesBN, from = "L" , to = "S" )
# NaiveBayesBN = bnlearn::set.arc(NaiveBayesBN, from = "B" , to = "S" )
# NaiveBayesBN = bnlearn::set.arc(NaiveBayesBN, from = "A" , to = "S" )
# NaiveBayesBN = bnlearn::set.arc(NaiveBayesBN, from = "X" , to = "S" )
# NaiveBayesBN = bnlearn::set.arc(NaiveBayesBN, from = "E" , to = "S" )

bnlearn::graphviz.plot(NaiveBayesBN, main = "Naive Bayes BN"
                       ,highlight = list(nodes = "S" ,  col = "tomato",
                                         fill = "orange") )


NaiveBayesBNPred = fnPredict(trainData = train , testData = test , DAGraph.bnObject = NaiveBayesBN
                 , predNode = "S" , Method = "mle" , markovB = FALSE , returnDataFrame = FALSE )

cat("Confusion Matrix for Naive Bayes BN for S ")
cat("\n")
NaiveBayesBNPred
cat("\n")
MiscalNaiveB = 1 - sum(diag(NaiveBayesBNPred))/sum(NaiveBayesBNPred)
cat("Misclassification rate for node S Naive Bayes BN" , MiscalNaiveB)
cat("\n")

par(mfrow = c(1,3))
bnlearn::graphviz.plot(bn.hc, main = "Learned Asia Data BN"
                       #,  layout = "neato",
                       ,highlight = list(nodes = "S" ,  col = "tomato",
                                         fill = "orange") )
bnlearn::graphviz.plot(dag.True, main = "True Asia Data BN"
                       #,  layout = "neato",
                       ,highlight = list(nodes = "S" ,  col = "tomato",
                                         fill = "orange") )

bnlearn::graphviz.plot(NaiveBayesBN, main = "Naive Bayes BN"
                       ,highlight = list(nodes = "S" ,  col = "tomato",
                                         fill = "orange") )
```