# 732A96 Advanced Machine Learning: Lab 1 - Group Report

Shashi Nagarajan (shana299@student.liu.se)
Akshay Gurudath (aksgu350@student.liu.se) Nicolas Taba (nicta839@student.liu.se)
Salvador Marti Roman (salma742@student.liu.se)

13/09/2021

## 1. Multiple runs of the hill-climbing algorithm.

**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.**

**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 sample 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

```
library(bnlearn)
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.0.5
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.0.5
```

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.0.5
```

```
##
## Attaching package: 'e1071'
```

```
## The following object is masked from 'package:bnlearn':
##
##     impute
```

```
library(gRain)

data("asia")
set.seed(42)

head(asia)
```

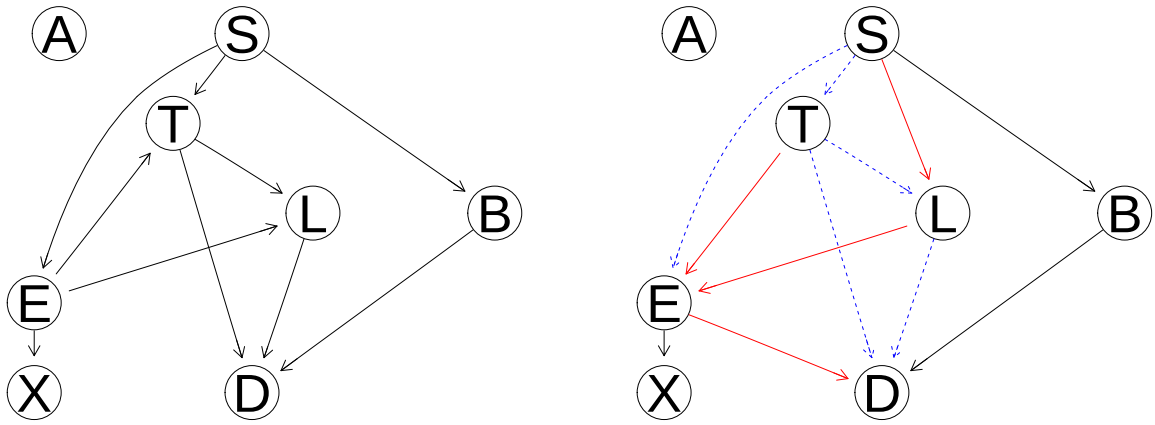
```
##    A   S   T  L   B   E   X   D
## 1 no yes  no no yes  no  no yes
## 2 no yes  no no  no  no  no  no
## 3 no  no yes no  no yes yes yes
```

```
## 4 no  no  no no yes  no  no yes
## 5 no  no  no no  no  no  no yes
## 6 no yes  no no  no  no  no yes
```

In order to show that multiple runs of the hill-climbing algorithm may return non-equivalent structures, we conduct two runs of the hill-climbing algorithm with each starting from a randomly initialised DAG.

The reason we observe two different outputs below is that without exploring all possible initialisations, the hill-climbing algorithm is only guaranteed to converge at a local minimum in finite time. What we observe below are results corresponding to two different local minima, seeded by two different initialisations.

```
r1 = random.graph(colnames(asia))
r2 = random.graph(colnames(asia))

while (all.equal(cpdag(r1), cpdag(r2)) == T) {
  r2 = random.graph(colnames(asia)) # ensure r2 is different from r1
}

hc1 = hc(asia, r1)
hc2 = hc(asia, r2)

graphviz.compare(hc1, hc2)
```




The following function checks if both adjacencies and unshielded colliders are the same in two given graphs. If two graphs share both characteristics the two are considered to be equivalent BN structures.

```
equivalent_graphs = function(hc1, hc2){
  flag = F
  if (all.equal(cpdag(hc1), cpdag(hc2)) != T) {
    print("The two runs produced graphs with different adjacencies")
    } else {
      hc1_uc <- vstructs(hc1, arcs = T)
      hc2_uc <- vstructs(hc2, arcs = T)
      hc1_uc_from <- hc1_uc[, 1]
      hc1_uc_to <- hc1_uc[, 2]
      hc2_uc_from <- hc2_uc[, 1]
      hc2_uc_to <- hc2_uc[, 2]
      if (nrow(hc1_uc) != nrow(hc2_uc)) {
        print("The two runs produced graph with different unshielded colliders")
        } else {
          for (i in 1:nrow(hc1_uc)) {
            match_list = which(hc2_uc_from %in% hc1_uc_from[i])
```

```
            if (length(match_list) == 0) {
              flag = T
              break
              } else {
                for (j in match_list) {
                  if (hc2_uc_to[j] != hc1_uc_to[i]) {
                    flag = T
                    break
                  }
                }
              }
              if (flag) {break}
            }
        }
  }
  if (flag) {print("The two runs produced graph with different unshielded colliders")}
}

equivalent_graphs(hc1,hc2)
```

```
## [1] "The two runs produced graphs with different adjacencies"
```

## 2. Predicting smokers from a learnt a Bayesian Network using the Asia Dataset.

**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.**

**Hint:** You already know two algorithms for exact inference in BNs: Variable elimination and cluster trees. There are also approximate algorithms for when the exact ones are too demanding computationally. For exact inference, you may need the functions bn.fit and as.grain from the bnlearn package, and the functions compile, setEvidence and querygrain from the package gRain. For approximate inference, you may need the functions cpquery or cpdist, prop.table and table from the bnlearn package.

```
# split the data
set.seed(42)
n <- dim(asia)[1]
id <- sample(1:n, floor(n*0.8))
train <- asia[id, ]
test <- asia[-id, ]
```

### Approximate inference.

This approach uses cpquery which samples data instead of using every single entry.

```
hc_train = hc(train)
# in case result has undirected arcs, force directions; choice of ordering arbitrary
hc_train_dag = pdag2dag(hc_train, ordering = colnames(asia))
params_hc_train = bn.fit(hc_train_dag, train)
```

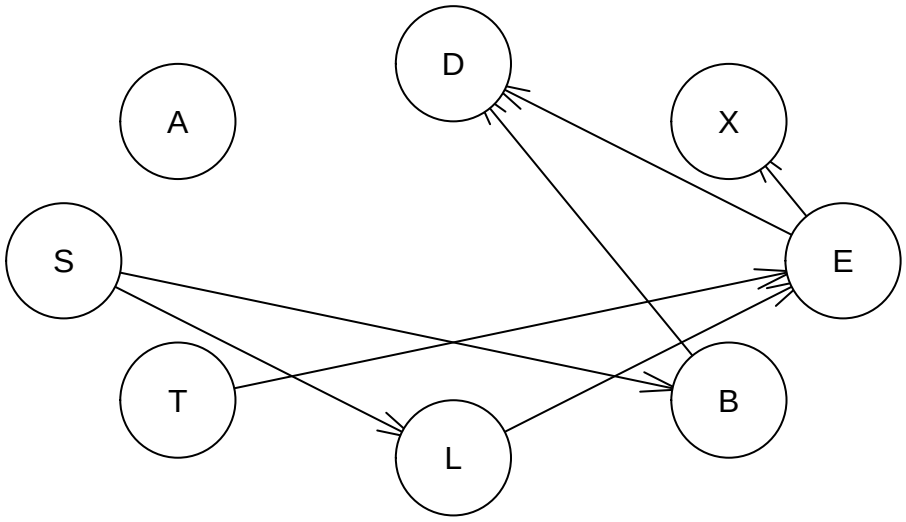
```
pred_S <- function(x, params) { # assume x is named
  evidence_text = paste(
      "(A == \"", x$A, "\") & ",
      "(T == \"", x$T, "\") & ",
      "(L == \"", x$L, "\") & ",
      "(B == \"", x$B, "\") & ",
      "(E == \"", x$E, "\")",
    sep = "")
  query_text = paste("cpquery(params, (S == \"yes\"), ", evidence_text, ")", sep = "")
  return(ifelse(eval(parse(text = query_text)) > 0.5, "yes", "no"))
}
preds = sapply(1:nrow(test), function(i) pred_S(test[i, ], params_hc_train))
cat("Confusion Matrix:")
```

```
## Confusion Matrix:
```

```
print(confusionMatrix(as.factor(preds), test$S))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  347 125
##        yes 160 368
##
##                Accuracy : 0.715
##                  95% CI : (0.6859, 0.7428)
##     No Information Rate : 0.507
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.4304
##
##  Mcnemar's Test P-Value : 0.04401
##
##             Sensitivity : 0.6844
##             Specificity : 0.7465
##          Pos Pred Value : 0.7352
##          Neg Pred Value : 0.6970
##              Prevalence : 0.5070
##          Detection Rate : 0.3470
##    Detection Prevalence : 0.4720
##       Balanced Accuracy : 0.7154
##
##        'Positive' Class : no
##
```

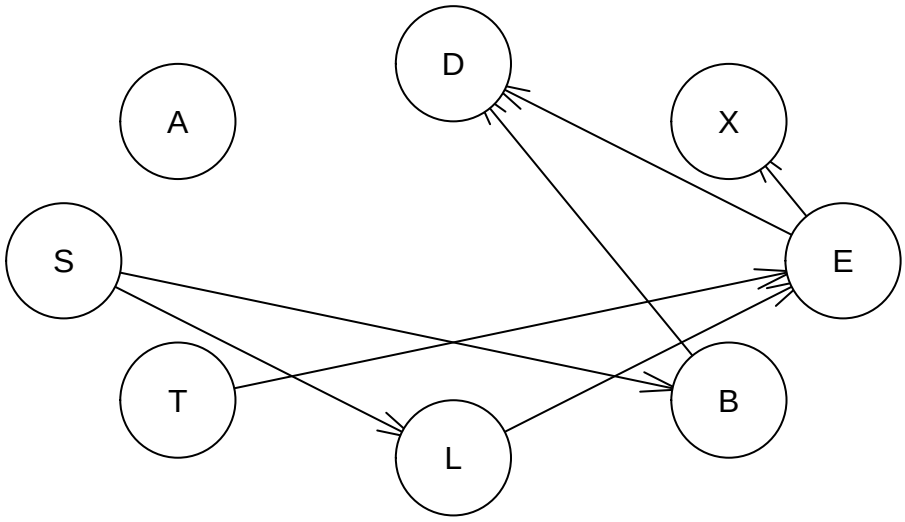
```
plot(hc_train,main="DAG obtained from Hill Climbing Algorithm")
```




### Exact inference.

```
# This function predicts S given the evidence data and the DAG
predict_S<-function(evidence_data,graph){
  mat<-as.matrix(evidence_data)
  x=1:nrow(mat)
  for (i in 1:nrow(mat)){
    sE=setEvidence(graph,nodes=names(evidence_data),states=mat[i,])
    x[i]=querygrain(sE,nodes=c("S"))$S[2] #The index 2 returs yes
  }
  S_predict <- ifelse(x >=0.5, "yes", "no")
  confusionMatrix(as.factor(test$S),as.factor(S_predict))
}

test_modified<-test[ , !(names(test) %in% c("S"))] #Evidence data

bn2<-bn.fit(hc(train),train) #Training the graph on Train data
bn_grain<-as.grain(bn2) #Graining and compiling it

c1<-compile(bn_grain)

plot(hc(train),main="DAG obtained from Hill Climbing Algorithm")
```

## DAG obtained from Hill Climbing Algorithm




```
print("Following is the confusion matrix for the DAG obtained from the Hill Climbing Algorithm")
```

```
## [1] "Following is the confusion matrix for the DAG obtained from the Hill Climbing Algorithm"
```

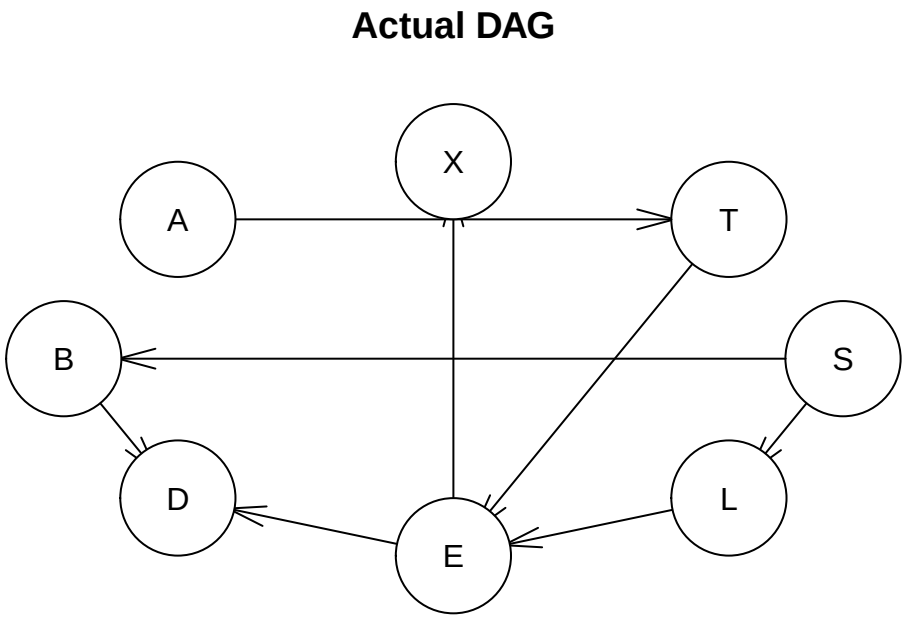
```
predict_S(test_modified,c1)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  347 160
##        yes 125 368
##
##                Accuracy : 0.715
##                  95% CI : (0.6859, 0.7428)
##     No Information Rate : 0.528
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.4304
##
##  Mcnemar's Test P-Value : 0.04401
##
##             Sensitivity : 0.7352
##             Specificity : 0.6970
##          Pos Pred Value : 0.6844
##          Neg Pred Value : 0.7465
##              Prevalence : 0.4720
```

```
##          Detection Rate : 0.3470
##    Detection Prevalence : 0.5070
##       Balanced Accuracy : 0.7161
##
##        'Positive' Class : no
##
```

**Comparison with the true graph.**

```
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
plot(dag,main="Actual DAG")
```




```
print("Following is the confusion matrix for the actual DAG")
```

```
## [1] "Following is the confusion matrix for the actual DAG"
```

```
predict_S(test_modified,compile(as.grain(bn.fit(dag,train))))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  347 160
##        yes 125 368
##
##                Accuracy : 0.715
##                  95% CI : (0.6859, 0.7428)
```

```
##     No Information Rate : 0.528
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.4304
##
##  Mcnemar's Test P-Value : 0.04401
##
##             Sensitivity : 0.7352
##             Specificity : 0.6970
##          Pos Pred Value : 0.6844
##          Neg Pred Value : 0.7465
##              Prevalence : 0.4720
##          Detection Rate : 0.3470
##    Detection Prevalence : 0.5070
##       Balanced Accuracy : 0.7161
##
##        'Positive' Class : no
##
```

## 3. Learning a Bayesian Network from the Markov Blanket of S

**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.

From now on we will continue using approximate inference for our results.

```
graph<-hc(train)
mb<-mb(graph,"S")
test_mb<-test[ ,mb]
print("The confusion matrix given only the Markov Blanket of S")
```

```
## [1] "The confusion matrix given only the Markov Blanket of S"
```

```
predict_S(test_mb,compile(as.grain(bn.fit(graph,train))))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  347 160
##        yes 125 368
##
##                Accuracy : 0.715
##                  95% CI : (0.6859, 0.7428)
##     No Information Rate : 0.528
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.4304
##
##  Mcnemar's Test P-Value : 0.04401
##
##             Sensitivity : 0.7352
##             Specificity : 0.6970
```

```
##          Pos Pred Value : 0.6844
##          Neg Pred Value : 0.7465
##              Prevalence : 0.4720
##          Detection Rate : 0.3470
##    Detection Prevalence : 0.5070
##       Balanced Accuracy : 0.7161
##
##        'Positive' Class : no
##
```

## 4. Using a naive Bayes classifier.

**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.**

In Naive Bayes (Wikipedia Naive Bayes, n.d.), the assumption is the following:

$$p(C, x1, x2, .., xn) \propto p(C)p(x1|C)p(x2|C)...p(xn|C)$$

This is the joint density of the DAG and We model all the other variables as being dependent only on S and no other variable.

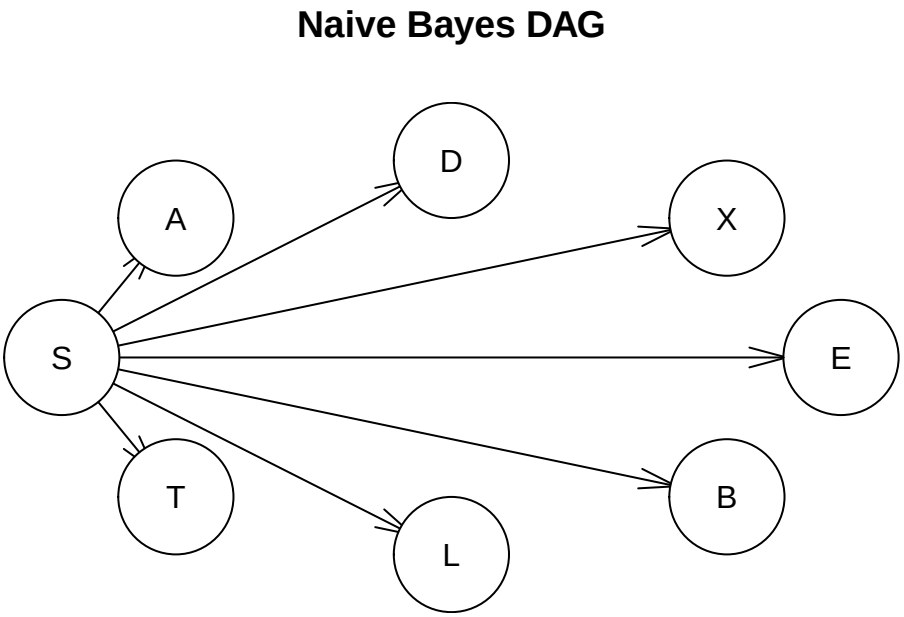
Therefore the DAG is as follows:

```
nb = empty.graph(colnames(asia))
arc.set = cbind(rep("S", ncol(asia) - 1), colnames(asia)[colnames(asia) != "S"])
colnames(arc.set) = c("from", "to")
arcs(nb) = arc.set
params_nb_train = bn.fit(nb, train)
preds_nb = sapply(1:nrow(test), function(i) pred_S(test[i, ], params_nb_train))
confusionMatrix(as.factor(preds_nb), test$S)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  346 126
##        yes 161 367
##
##                Accuracy : 0.713
##                  95% CI : (0.6839, 0.7409)
##     No Information Rate : 0.507
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.4264
##
##  Mcnemar's Test P-Value : 0.04475
##
##             Sensitivity : 0.6824
##             Specificity : 0.7444
##          Pos Pred Value : 0.7331
##          Neg Pred Value : 0.6951
```

```
##              Prevalence : 0.5070
##          Detection Rate : 0.3460
##    Detection Prevalence : 0.4720
##       Balanced Accuracy : 0.7134
##
##        'Positive' Class : no
##
```

```
plot(nb, main="Naive Bayes DAG")
```




## 5. Analysis and comparison of Naives Bayes graph, Hill Climbing graph and the Hill Climbing graph with Markov Blanket

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

**Task 2 vs Task 3** In the true causal BN corresponding to the Asia dataset, we see that

$$S \perp_G A, D, E, T, X | \phi$$

. This must mean that

$$S \perp_p A, D, E, T, X | \phi$$

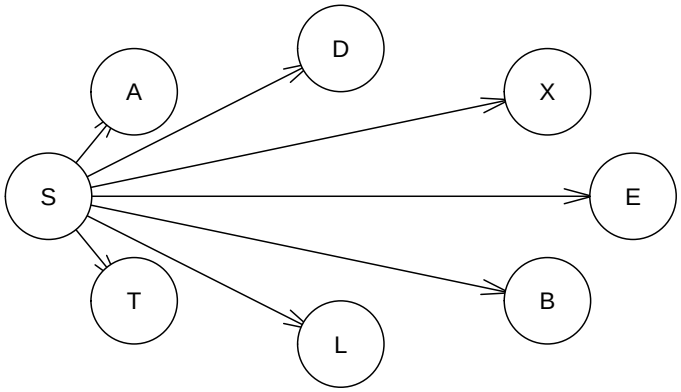
Furthermore, the Markov blanket of $S$ is $L, B$ in the DAG obtained from the 80%-random-sample of the Asia dataset. Thus, given the said blanket, other variables serve no purpose, causing result in Tasks 2 and 3 to be nearly identical (NB: approximate inference used for classification)

**Task 2 vs Task 4** Based on the graphs obtained, it is hard to reason why we obtained the same results in Tasks 2 and 4; it could be that errors on account of approximate inference and/or the specifications of data

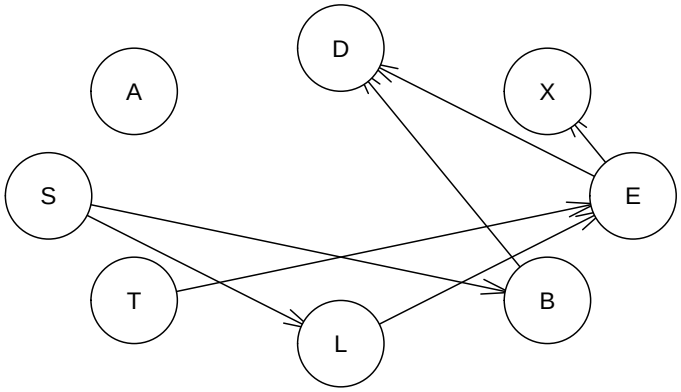
that we happened to observe are making it appear as though the results are identical though one would not exactly expect it.

In fact, we would not, in general, expect the Naive Bayes algorithm to return similar output vis-a-vis the generic probabilistic-graphical method; we would expect the latter to perform at least as well as the Naive Bayes algorithm, however, since Naive Bayes imposes some independence constraints that may not be warranted prior to observing data.

**Naive Bayes DAG**




**DAG obtained from Hill Climbing Algorithm**



## Appendix: All code for this report

```
if (!requireNamespace("BiocManager", quietly = TRUE))
install.packages("BiocManager")
BiocManager::install("RBGL")
BiocManager::install("Rgraphviz")
BiocManager::install("gRain")
library(bnlearn)
library(Rgraphviz)
library(gRain)


library(bnlearn)
library(caret)
library(e1071)
library(gRain)

data("asia")
set.seed(42)

head(asia)


r1 = random.graph(colnames(asia))
r2 = random.graph(colnames(asia))

while (all.equal(cpdag(r1), cpdag(r2)) == T) {
  r2 = random.graph(colnames(asia)) # ensure r2 is different from r1
}

hc1 = hc(asia, r1)
hc2 = hc(asia, r2)

graphviz.compare(hc1, hc2)


equivalent_graphs = function(hc1, hc2){
  flag = F
  if (all.equal(cpdag(hc1), cpdag(hc2)) != T) {
    print("The two runs produced graphs with different adjacencies")
    } else {
      hc1_uc <- vstructs(hc1, arcs = T)
      hc2_uc <- vstructs(hc2, arcs = T)
      hc1_uc_from <- hc1_uc[, 1]
      hc1_uc_to <- hc1_uc[, 2]
      hc2_uc_from <- hc2_uc[, 1]
      hc2_uc_to <- hc2_uc[, 2]
      if (nrow(hc1_uc) != nrow(hc2_uc)) {
        print("The two runs produced graph with different unshielded colliders")
        } else {
          for (i in 1:nrow(hc1_uc)) {
            match_list = which(hc2_uc_from %in% hc1_uc_from[i])
            if (length(match_list) == 0) {
```

```
              flag = T
              break
              } else {
                for (j in match_list) {
                  if (hc2_uc_to[j] != hc1_uc_to[i]) {
                    flag = T
                    break
                  }
                }
              }
              if (flag) {break}
            }
        }
  }
  if (flag) {print("The two runs produced graph with different unshielded colliders")}
}

equivalent_graphs(hc1,hc2)


# split the data
set.seed(42)
n <- dim(asia)[1]
id <- sample(1:n, floor(n*0.8))
train <- asia[id, ]
test <- asia[-id, ]


hc_train = hc(train)
# in case result has undirected arcs, force directions; choice of ordering arbitrary
hc_train_dag = pdag2dag(hc_train, ordering = colnames(asia))
params_hc_train = bn.fit(hc_train_dag, train)
pred_S <- function(x, params) { # assume x is named
  evidence_text = paste(
      "(A == \"", x$A, "\") & ",
      "(T == \"", x$T, "\") & ",
      "(L == \"", x$L, "\") & ",
      "(B == \"", x$B, "\") & ",
      "(E == \"", x$E, "\")",
    sep = "")
  query_text = paste("cpquery(params, (S == \"yes\"), ", evidence_text, ")", sep = "")
  return(ifelse(eval(parse(text = query_text)) > 0.5, "yes", "no"))
}
preds = sapply(1:nrow(test), function(i) pred_S(test[i, ], params_hc_train))
cat("Confusion Matrix:")
print(confusionMatrix(as.factor(preds), test$S))

plot(hc_train,main="DAG obtained from Hill Climbing Algorithm")


# This function predicts S given the evidence data and the DAG
predict_S<-function(evidence_data,graph){
  mat<-as.matrix(evidence_data)
```

```
  x=1:nrow(mat)
  for (i in 1:nrow(mat)){
    sE=setEvidence(graph,nodes=names(evidence_data),states=mat[i,])
    x[i]=querygrain(sE,nodes=c("S"))$S[2] #The index 2 returs yes
  }
  S_predict <- ifelse(x >=0.5, "yes", "no")
  confusionMatrix(as.factor(test$S),as.factor(S_predict))
}

test_modified<-test[ , !(names(test) %in% c("S"))] #Evidence data

bn2<-bn.fit(hc(train),train) #Training the graph on Train data
bn_grain<-as.grain(bn2) #Graining and compiling it

c1<-compile(bn_grain)

plot(hc(train),main="DAG obtained from Hill Climbing Algorithm")
print("Following is the confusion matrix for the DAG obtained from the Hill Climbing Algorithm")
predict_S(test_modified,c1)


dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
plot(dag,main="Actual DAG")
print("Following is the confusion matrix for the actual DAG")
predict_S(test_modified,compile(as.grain(bn.fit(dag,train))))


graph<-hc(train)
mb<-mb(graph,"S")
test_mb<-test[ ,mb]
print("The confusion matrix given only the Markov Blanket of S")
predict_S(test_mb,compile(as.grain(bn.fit(graph,train))))


nb = empty.graph(colnames(asia))
arc.set = cbind(rep("S", ncol(asia) - 1), colnames(asia)[colnames(asia) != "S"])
colnames(arc.set) = c("from", "to")
arcs(nb) = arc.set
params_nb_train = bn.fit(nb, train)
preds_nb = sapply(1:nrow(test), function(i) pred_S(test[i, ], params_nb_train))
confusionMatrix(as.factor(preds_nb), test$S)
plot(nb, main="Naive Bayes DAG")


plot(nb, main="Naive Bayes DAG")
# plot(hc_train,main="DAG obtained from Hill Climbing Algorithm")

plot(hc_train,main="DAG obtained from Hill Climbing Algorithm")
```