Group_Lab1_AML

Alejo Perez, Joel Nilsson, Martynas Lukosevicius

14/09/2021

Statement on contribution:

Our group formed by Joel Nilsson, Martynas Lukosevicius & Alejo Pérez Task 1: code (Martynas & Alejo) Discussion (Joel, Martynas, Alejo) Task 2: code (Martynas) Discussion (Joel, Martynas, Alejo) Task 3: code (Joel) Discussion (Joel, Martynas, Alejo) Task 4: code (Martynas) Discussion (Joel, Martynas, Alejo) Task 5: Discussion (Joel, Martynas, Alejo)

1

In this assignment we are required to show that it is possible to yield non-equivalent graphs when running hill-climbing algorithm multiple times. In order to do so, we approached this purpose by two different procedures. Both of them involved a loop that just could be exited when a constraint was violated. In the first approach we started with an empty graph, running the loop as many times as necessary (same setup of parameters) until we get 2 non-equivalent networks ("different arc sets" criterion). The
second one starts with 2 different randomized graphs and runs the hc algorithm multiple times with same parameters until it yields 2 graphs that have "Different number of directed/undirected arcs".

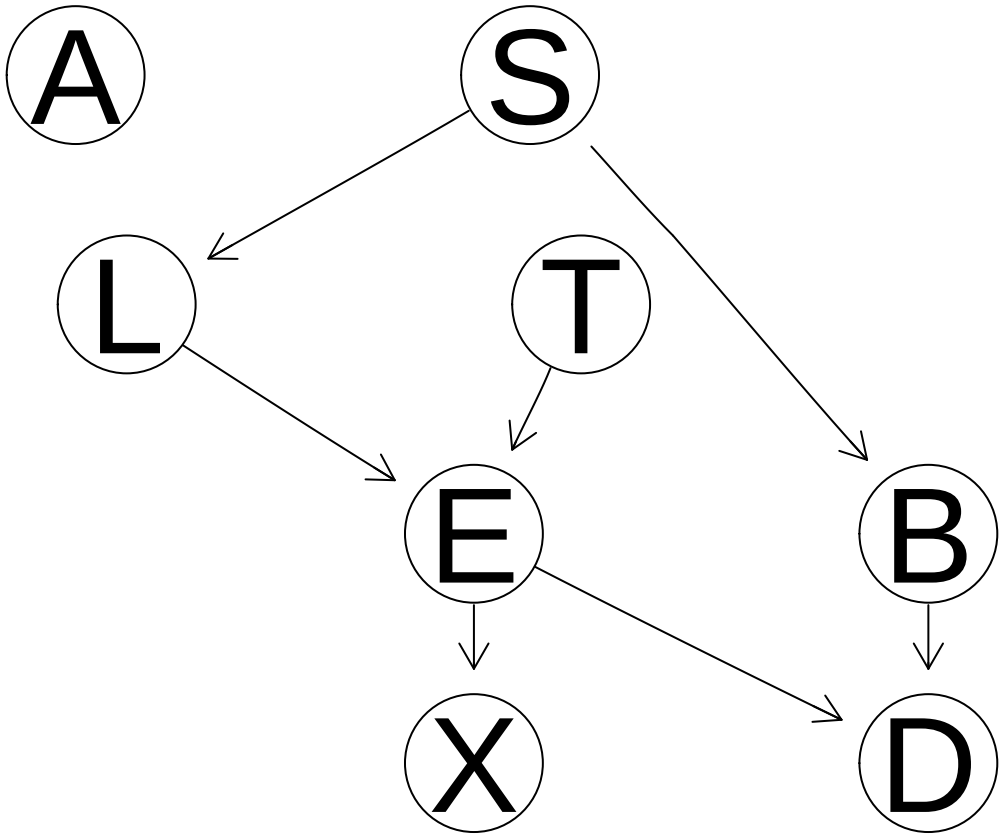
```
library(bnlearn)
library(Rgraphviz)

data(asia)
stat <- TRUE
stat <- "True"

while (stat != "Different arc sets") {
hcnetwork1 <- hc(asia, restart = 100,  score = "bde", iss = 1)
hc_network2 <- hc(asia, restart = 100, score = "bde", iss = 1)



stat = all.equal(hcnetwork1, hc_network2)
}

cat("BDE score first graph:", score(hcnetwork1, asia, type = "bde", iss = 1))
```

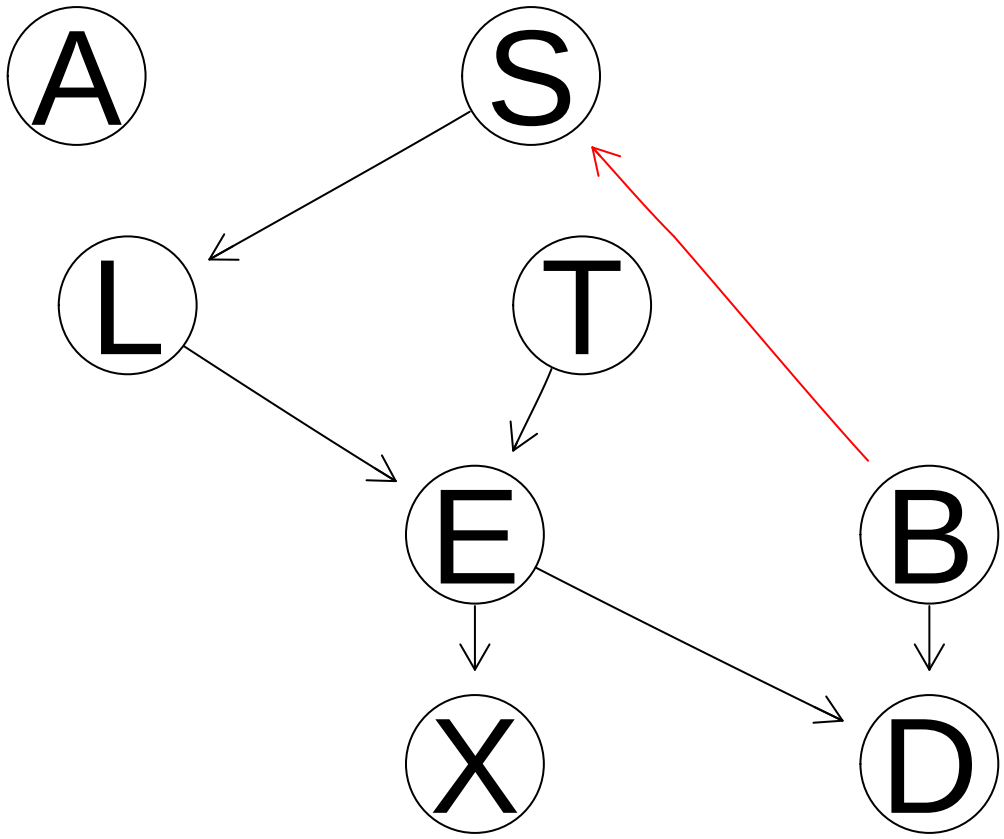
```
## BDE score first graph: -11095.79
```

```
cat("\nBDE score second graph:", score(hc_network2, asia, type = "bde", iss = 1))
```

```
##
## BDE score second graph: -11095.79
```

```
graphviz.compare(hc_network2, hcnetwork1)
```






```
stop=FALSE
while (stop==FALSE){



  dg1r = random.graph(colnames(asia))

  dg2r = random.graph(colnames(asia))

  set.seed(1234)

  dag1 = hc(asia, dg1r, restart=200, iss=1, score="bde")
  set.seed(1234)

  dag2 = hc(asia,dg2r, restart=200, iss=1, score="bde")
  criterion <- all.equal(dag1, dag2)
  if((criterion !=  "Different arc sets") & (criterion != "TRUE")){stop=TRUE}
  print(criterion)
}
```

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

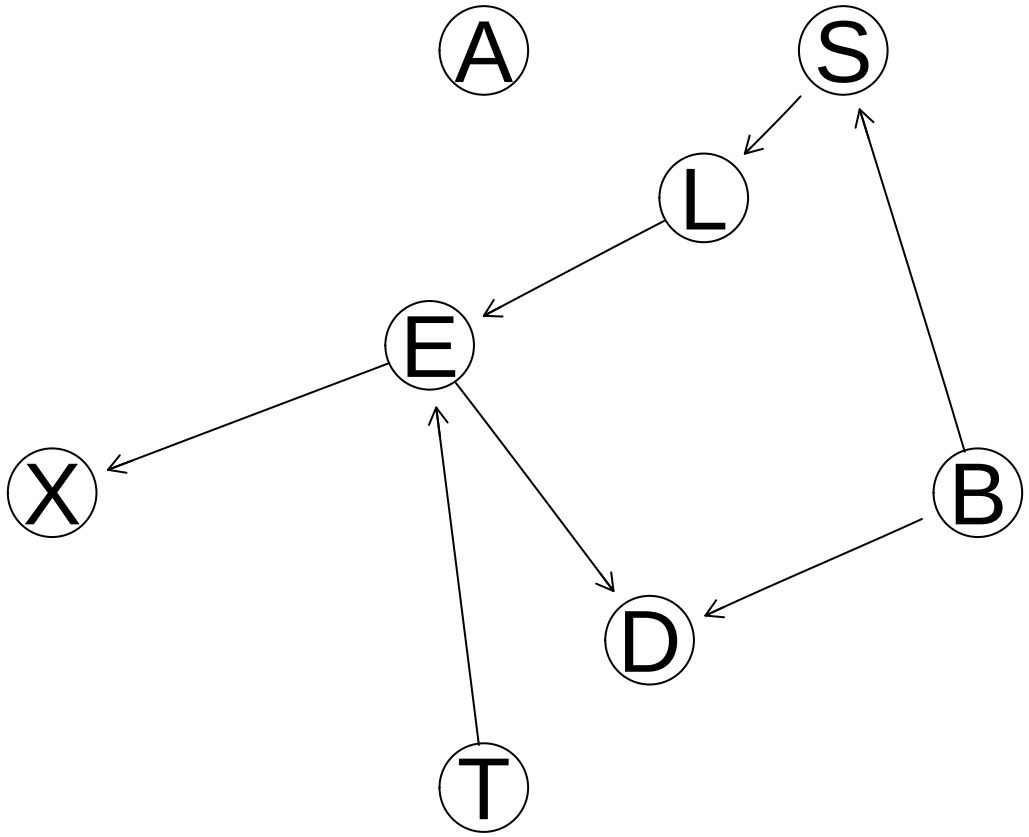
```
cat("BDE score first graph:", score(dag1, asia, type = "bde", iss = 1))
```

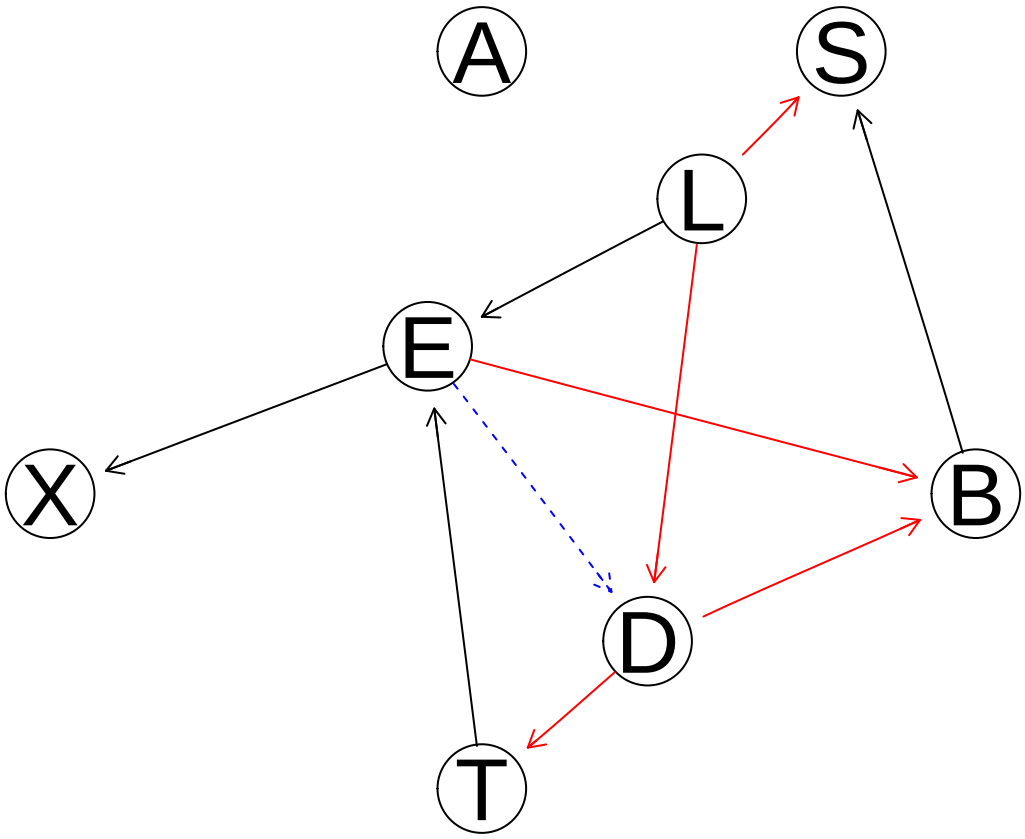
```
## BDE score first graph: -11095.79
```

```
cat("\nBDE score second graph:", score(dag2, asia, type = "bde", iss = 1))
```

```
##
## BDE score second graph: -11102.94
```

```
graphviz.compare(dag1,dag2)
```





Hill climbing algorithm is a greedy algorithm and it doesn't guarantee global optimum. Since there are many local optima, we may end up in any of them, and they sometimes correspond to non-equivalent graph structures.

First graph even if direction of the arrow is different, it doesnt mean that graphs are not Markovian equivalent. Actually it is Markovian equivalent as we have same adjacencies and same unshielded collides. BDEU score returns equivalent scores for markovian equal graphs. As arrow direction doesn't necessarily change markovian equivalence HC algorithm can return visually different graphs

In second plot we can see that graphs are not markovian equivalent. The reason why we get non-equivalent graphs is because hill climbing algorithm starts from different initial graphs, as a result it end up in different local optima.

2

For this assignment we used gRain and bnlearn to fit a hill-climbing BN model with default setting and afterwards classify a testing set according to the higher probability obtained in the probabilistic inference. First off, we did the graph learning and the parameters were estimated by maximum likelihood estimation.

```
library(gRain)
train_idx <- sample(c(1:dim(asia)[1]), size = 0.8* dim(asia)[1])

train_data <- asia[train_idx, ]
test_data <- asia[-train_idx, ]

model <- hc(train_data)
```

```
fit <- bn.fit(model, train_data)

# convert to grain

model_grain <- as.grain(fit)

target <- test_data$S
pred_data <- subset(test_data, select = -c(S))

data_char2 <- pred_data # Duplicate data
fac_cols <- sapply(data_char2, is.factor) # Identify all factor columns
data_char2[fac_cols] <- lapply(data_char2[fac_cols], as.character) # Convert all factors to characters

predict <- function(row, model){
res <- querygrain(model, evidence = as.list(row), nodes=c("S"))
return(names(which.max(res$S)))
}

predictions <- apply(data_char2, 1, predict, model = model_grain)


conf_matrix_hc <- table(predictions, as.character(target))
acc_hc <- sum(diag(conf_matrix_hc))/sum(conf_matrix_hc)

knitr::kable(conf_matrix_hc, caption = "confusion matrix of model")
```

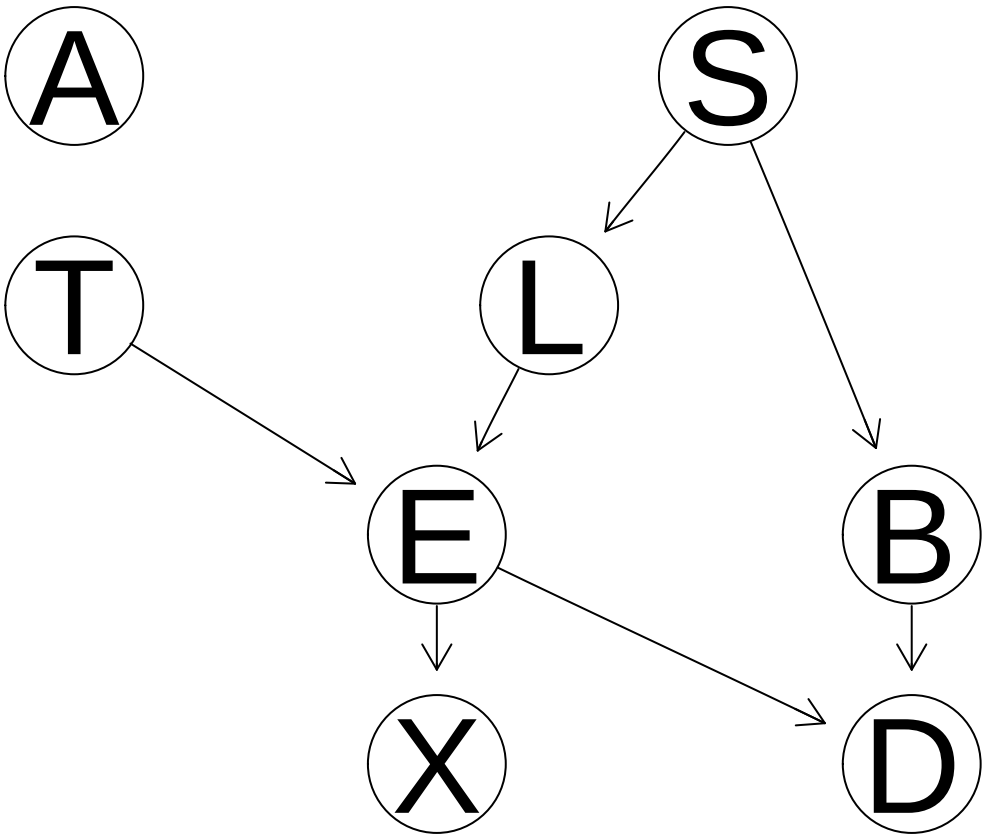
Table 1: confusion matrix of model

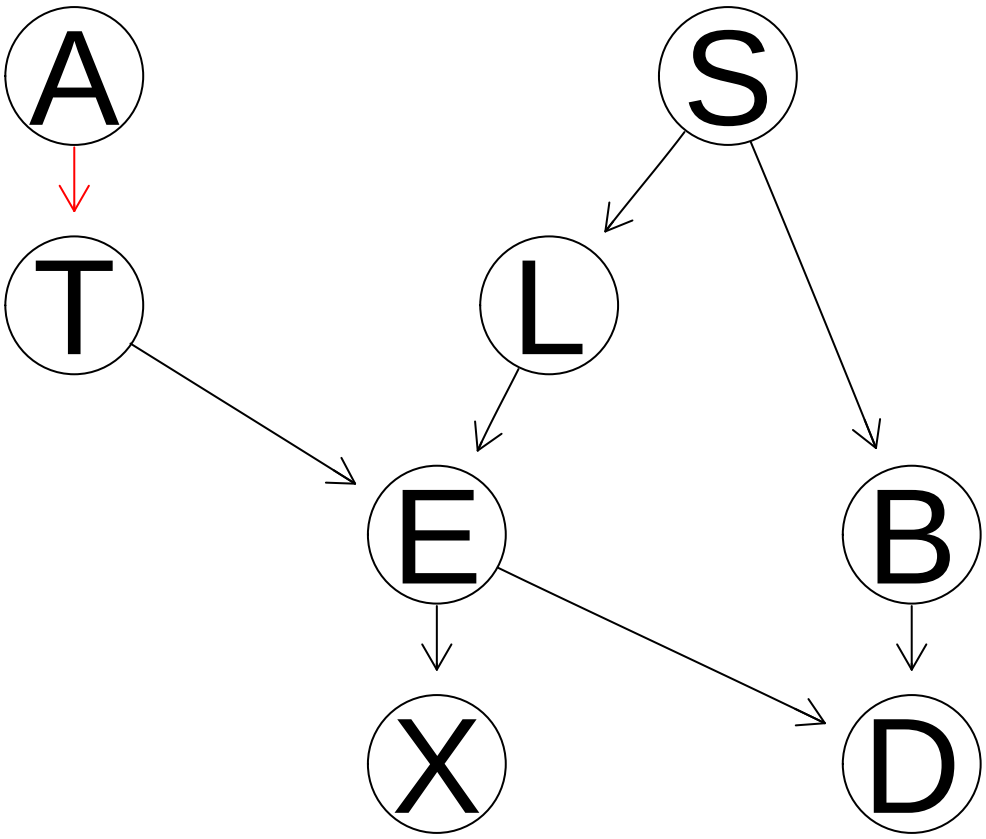
	no	yes
no	374	124
yes	140	362

accuracy of model 0.736

In regards to the true network given, we achieve the same performance.

```
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
graphviz.compare(model, dag)
```


A
S
T
L
E
B
X
D




```
fit_true <- bn.fit(dag, train_data)
model_true_grain <- as.grain(fit_true)

predictions_true <- apply(data_char2, 1, predict, model = model_true_grain)

conf_matrix_true <- table(predictions_true, as.character(target))
acc_true <- sum(diag(conf_matrix_true))/sum(conf_matrix_true)

knitr::kable(conf_matrix_true, caption = "confusion matrix of model with true graph")
```

Table 2: confusion matrix of model with true graph

	no	yes
no	374	124
yes	140	362

accuracy of model using true graph 0.736

confusion matrix of true dag and constructed using hill climbing algorithm are the same

3

We will use the Markov Blanket to predict the test instances.

```r
data(asia)

train_data <- asia[train_idx, ]

model <- hc(train_data)
fit <- bn.fit(model, train_data)

# convert to grain

model_grain <- as.grain(fit)


nodes=c("A","T", "L", "B", "E", "X", "D")


lev <- levels(asia[,"S"])
MarkovBlanket <- mb(model, "S")


test_data <- asia[-train_idx,MarkovBlanket ]




mb_s_pred <- {}
for(i in 1:dim(test_data)[1]){
  mb_query <- querygrain(setEvidence(model_grain,
                                     nodes=MarkovBlanket,
                                     states=sapply(test_data[i,], toString),
                                     propagate = TRUE),
                         nodes=c("S"))
  if(mb_query$S[1] > 0.5)
    mb_s_pred <- c(mb_s_pred, lev[1])
  else
    mb_s_pred <- c(mb_s_pred, lev[2])
}



mb_true_result <- asia[-train_idx,"S" ]



knitr::kable(table(mb_s_pred,
      mb_true_result), caption = "confusion matrix of model with Markov Blanket")
```

Table 3: confusion matrix of model with Markov Blanket

	no	yes
no	374	124
yes	140	362

```
acc_true <- sum(diag(conf_matrix_true))/sum(conf_matrix_true)
```

4

```
dag_nb = model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")
fit_nb <-  bn.fit(dag_nb, train_data)
model_nb <- as.grain(fit_nb)

predictions <- apply(data_char2, 1, predict, model = model_nb)
conf_matrix_nb <- table(predictions, as.character(target))
acc_nb <- sum(diag(conf_matrix_nb))/sum(conf_matrix_nb)

knitr::kable(conf_matrix_nb, caption = "confusion matrix for naive bayes")
```

Table 4: confusion matrix for naive bayes

	no	yes
no	395	182
yes	119	304

accuracy of naive Bayes 0.699

5

Generally throughout this report, as long as S keeps the same Markov Blankets we will obtain same predictions. That explains why we get the same result in the True Network of task to and the learnt model. We also got the same prediction results with the graphs in task 2 and 3. This is because the states of the variables not in the Markov blanket of S does not influence the predicted probability of S if we condition on the variables in the Markov blanket. The predictions in task 4 were different because the graph structure of the naive Bayes model is not equivalent to the other graphs - S depends on the state of all other variables.

Appendix

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

data(asia)
stat <- TRUE
stat <- "True"

while (stat != "Different arc sets") {
hcnetwork1 <- hc(asia, restart = 100,  score = "bde", iss = 1)
hc_network2 <- hc(asia, restart = 100, score = "bde", iss = 1)
```

```
stat = all.equal(hcnetwork1, hc_network2)
}

cat("BDE score first graph:", score(hcnetwork1, asia, type = "bde", iss = 1))
cat("\nBDE score second graph:", score(hc_network2, asia, type = "bde", iss = 1))

graphviz.compare(hc_network2, hcnetwork1)

stop=FALSE
while (stop==FALSE){



  dg1r = random.graph(colnames(asia))

  dg2r = random.graph(colnames(asia))

  set.seed(1234)

  dag1 = hc(asia, dg1r, restart=200, iss=1, score="bde")
  set.seed(1234)

  dag2 = hc(asia,dg2r, restart=200, iss=1, score="bde")
  criterion <- all.equal(dag1, dag2)
  if((criterion !=  "Different arc sets") & (criterion != "TRUE")){stop=TRUE}
  print(criterion)
}

cat("BDE score first graph:", score(dag1, asia, type = "bde", iss = 1))
cat("\nBDE score second graph:", score(dag2, asia, type = "bde", iss = 1))
graphviz.compare(dag1,dag2)

library(gRain)
train_idx <- sample(c(1:dim(asia)[1]), size = 0.8* dim(asia)[1])

train_data <- asia[train_idx, ]
test_data <- asia[-train_idx, ]

model <- hc(train_data)

fit <- bn.fit(model, train_data)

# convert to grain

model_grain <- as.grain(fit)

target <- test_data$S
pred_data <- subset(test_data, select = -c(S))

data_char2 <- pred_data # Duplicate data
fac_cols <- sapply(data_char2, is.factor) # Identify all factor columns
data_char2[fac_cols] <- lapply(data_char2[fac_cols], as.character) # Convert all factors to characters
```

```r
predict <- function(row, model){
res <- querygrain(model, evidence = as.list(row), nodes=c("S"))
return(names(which.max(res$S)))
}

predictions <- apply(data_char2, 1, predict, model = model_grain)


conf_matrix_hc <- table(predictions, as.character(target))
acc_hc <- sum(diag(conf_matrix_hc))/sum(conf_matrix_hc)

knitr::kable(conf_matrix_hc, caption = "confusion matrix of model")
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
graphviz.compare(model, dag)
fit_true <- bn.fit(dag, train_data)
model_true_grain <- as.grain(fit_true)

predictions_true <- apply(data_char2, 1, predict, model = model_true_grain)

conf_matrix_true <- table(predictions_true, as.character(target))
acc_true <- sum(diag(conf_matrix_true))/sum(conf_matrix_true)

knitr::kable(conf_matrix_true, caption = "confusion matrix of model with true graph")

data(asia)

train_data <- asia[train_idx, ]

model <- hc(train_data)
fit <- bn.fit(model, train_data)

# convert to grain

model_grain <- as.grain(fit)


nodes=c("A","T", "L", "B", "E", "X", "D")


lev <- levels(asia[,"S"])
MarkovBlanket <- mb(model, "S")


test_data <- asia[-train_idx,MarkovBlanket ]




mb_s_pred <- {}
for(i in 1:dim(test_data)[1]){
  mb_query <- querygrain(setEvidence(model_grain,
                                     nodes=MarkovBlanket,
                                     states=sapply(test_data[i,], toString),
```

```
                                    propagate = TRUE),
                          nodes=c("S"))
  if(mb_query$S[1] > 0.5)
    mb_s_pred <- c(mb_s_pred, lev[1])
  else
    mb_s_pred <- c(mb_s_pred, lev[2])
}



mb_true_result <- asia[-train_idx,"S" ]



knitr::kable(table(mb_s_pred,
      mb_true_result), caption = "confusion matrix of model with Markov Blanket")

acc_true <- sum(diag(conf_matrix_true))/sum(conf_matrix_true)

dag_nb = model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")
fit_nb <-  bn.fit(dag_nb, train_data)
model_nb <- as.grain(fit_nb)

predictions <- apply(data_char2, 1, predict, model = model_nb)
conf_matrix_nb <- table(predictions, as.character(target))
acc_nb <- sum(diag(conf_matrix_nb))/sum(conf_matrix_nb)

knitr::kable(conf_matrix_nb, caption = "confusion matrix for naive bayes")
```