Advanced Machine Learning Group Report

Kangbo Chen, Wuhao Wang, Daniel Persson and Hugo Axandersson

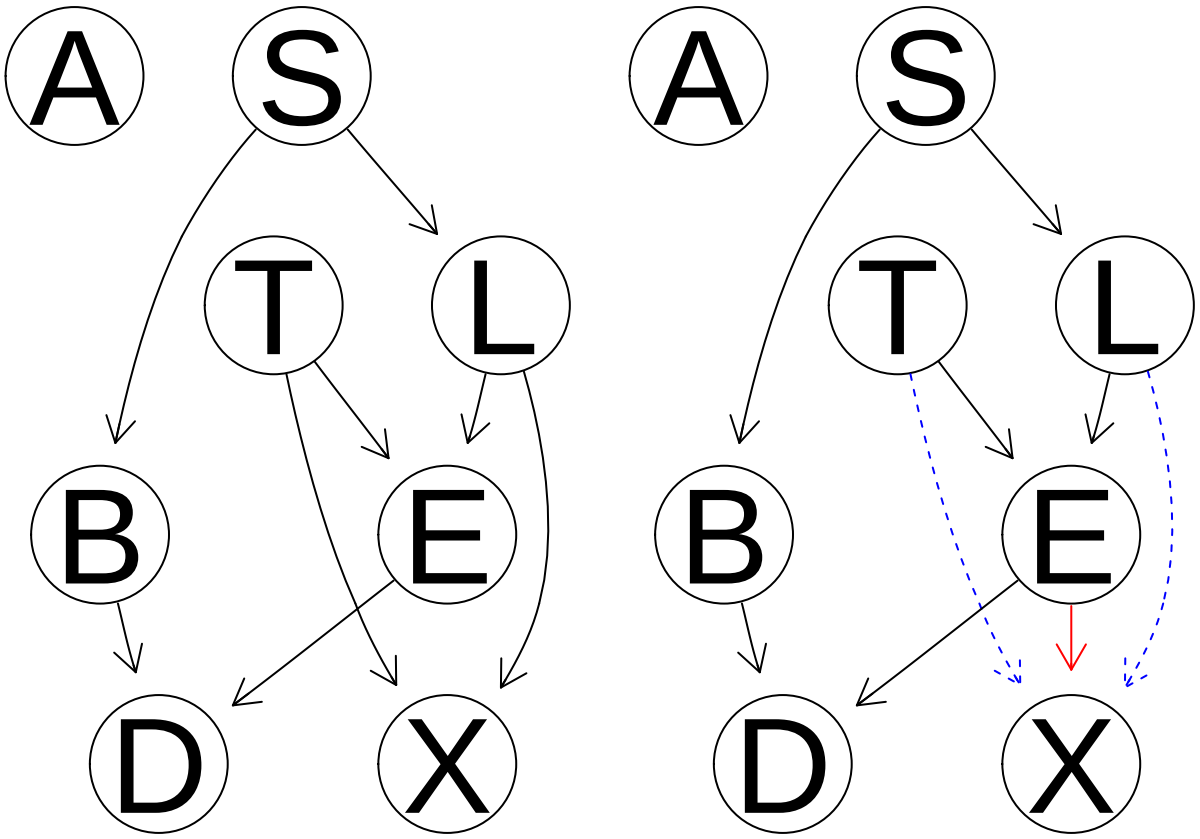
2022/9/7

Contribution

All exercises were completed individually and approaches/results were discussed within the group. Reports for exercise 1, 3 and 4 were produced primarily by Wuhao, Daniel and Hugo respectively, while those for exercises 2 and 5 were written by Kangbo.

Question 1

```
set.seed(123456789)
s1 <- random.graph(names(asia), num = 1, method = "ordered")

restart_1 <- hc(asia)
restart_2 <- hc(asia,restart = 20)

initial_1 <- hc(asia,start = s1)
initial_2 <- hc(asia)

iss_1 <- hc(asia, score = "bde", iss = 1)
iss_2 <- hc(asia, score = "bde", iss = 2)

score_1 <- hc(asia, score = "bde")
score_2 <- hc(asia, score = "aic")

show_difference <- function(g1, g2)
{
  equal <- all.equal(g1, g2)
  cp1 <- cpdag(g1)
  cp2 <- cpdag(g2)
  par(mfrow = c(1, 2))
  graphviz.compare(g1, g2)
  res <- list("all.equal" = g1, "cpdag" = list(cp1, cp2))
  return(res)
}

#show_difference(restart_1, restart_2)
show_difference(initial_1, initial_2)
```




```
## $all.equal
##
##   Bayesian network learned via Score-based methods
##
##   model:
##    [A][S][T][L|S][B|S][E|T:L][X|T:L][D|B:E]
##   nodes:                                 8
##   arcs:                                  8
##     undirected arcs:                     0
##     directed arcs:                       8
##   average markov blanket size:           2.50
##   average neighbourhood size:            2.00
##   average branching factor:              1.00
##
##   learning algorithm:                    Hill-Climbing
##   score:                                 BIC (disc.)
##   penalization coefficient:              4.258597
##   tests used in the learning procedure:  99
##   optimized:                             TRUE
##
##
## $cpdag
## $cpdag[[1]]
##
##   Bayesian network learned via Score-based methods
##
```

```
##   model:
##     [partially directed graph]
##   nodes:                                 8
##   arcs:                                  8
##     undirected arcs:                     2
##     directed arcs:                       6
##   average markov blanket size:           2.50
##   average neighbourhood size:            2.00
##   average branching factor:              0.75
##
##   learning algorithm:                    Hill-Climbing
##   score:                                 BIC (disc.)
##   penalization coefficient:              4.258597
##   tests used in the learning procedure:  99
##   optimized:                             TRUE
##
##
## $cpdag[[2]]
##
##   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
```

```
#show_difference(score_1, score_2)
#show_difference(iss_1, iss_2)
rm(list=ls())
```

Since `hc` algorithm is based on greedy search concept and it's easy to be trapped by local minimum. So initial structure and scoring method will have a great impact on the results. As mentioned in Bayesian course, `iss` will influence posterior, so different value will also result in different ending.

Question 2

```
dag <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

# separate dataset
set.seed(123456789)
N <- dim(asia)
train_id <- sample(1:N,floor(N*0.8))
train <- asia[train_id,]
```

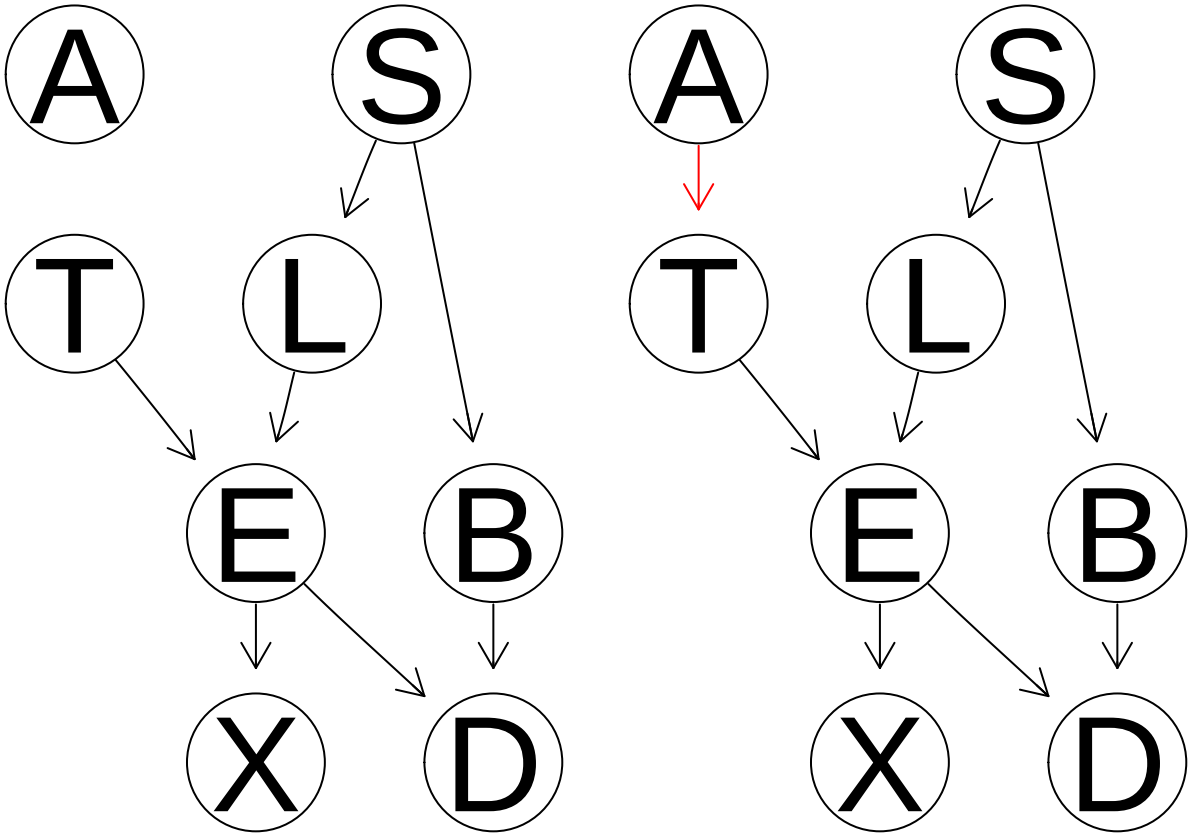
```r
test_id <- setdiff(1:N,train_id)
test <- asia[test_id,]
pre_train_dag <- dag
```

```r
BN_infer <- function(train, test, target, pre_train_dag, BN_flag = 1, appr = 0, use_mb = 0){

  # structure learning of BNs
  if(BN_flag == 1){
    bn_model_tr <- hc(train)
  }else{
    bn_model_tr <- pre_train_dag
  }

  # fit parameters
  fitted <- bn.fit(bn_model_tr,data = train)

  # tranform the class to grain class
  bn_gr <- as.grain(fitted)

  # compile the BNs
  bn_cp <- compile(bn_gr)

  # set Evidence
  if(use_mb == 0){
    evid_node <- colnames(test)
    evid_node <- setdiff(evid_node,target)
  }else{
    evid_node <- mb(fitted,target)
  }

  evid_val <- test[,evid_node]

  classifier <- function(evidence_val){
    tt <- setEvidence(bn_cp, nodes = evid_node, states = evidence_val)
    res <- names(which.max(querygrain(tt, nodes = target)[[1]]))
    return(res)
  }

  # results of test
  pred_y <- sapply(as.data.frame(t(evid_val)), FUN = classifier)
  return(list(pred_y,'BN structure' = bn_model_tr))

}

# test function
# train BN
res_1 <- BN_infer(train = train,
                  test = test,
                  target = 'S',
                  pre_train_dag = dag,
                  BN_flag = 1,
                  use_mb = 0)
# pretrain BN
res_2 <- BN_infer(train = train,
```

```
                  test = test,
                  target = 'S',
                  pre_train_dag = dag,
                  BN_flag = 0,
                  use_mb = 0)
```

Fitted and Given Bayesian Network Structure Comparison

```
par(mfrow = c(1, 2))
graphviz.compare(res_1[[2]], dag)
```



As can be observed in the graph above, the only difference relies on the dependence between nodes A and T with the fitted DAG loses this information.

Confusion Matrix with fitted Bayesian Networks

```
tt <- table(True = test[,'S'],Prdiction = res_1[[1]])
accuracy <- sum(diag(tt))/sum(tt)
print(tt)
```

```
##       Prdiction
## True    no yes
##   no   330 153
##   yes 128 389
```

```
cat('The accuracy rate is ', accuracy*100, '%.', sep = '')
```

```
## The accuracy rate is 71.9%.
```

From the confusion matrix, it is clear that there are 153 false negative and 128 false positive and the accuracy rate of the inference is 71.9.

Confusion Matrix with given Bayesian Networks

```
tt <- table(True = test[,'S'],Prediction = res_2[[1]])
accuracy <- sum(diag(tt))/sum(tt)
print(tt)
```

```
##       Prediction
## True    no yes
##   no   330 153
##   yes 128 389
```

```
cat('The accuracy rate is ', accuracy*100, '%.', sep = '')
```

```
## The accuracy rate is 71.9%.
```

The same as fitted Network structure, 153 negative samples are wrongly classified with the figure for positive samples is 128, and the accuracy rate of the inference is still 71.9.

Question 3

The Markov blanket of S:

```
# Hill-Climbing on train set
model2 <- hc(train, restart = 10)

model2_fit <- bn.fit(model2, train)
model2_grain <- as.grain(model2_fit)
model2_compile <- compile(model2_grain)

mb(model2_fit, "S")
```

```
## [1] "L" "B"
```

```
# prediction function
pred_bn <- function(model, data, ini_index = c(1,3:8)) {

  node_names <- colnames(data[ini_index])

  pred <- c()

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

    state <- c()

    for (j in 1:ncol(data)) {

      if (data[i,j] == "yes") {
        state <- c(state, "yes")
      } else {
        state <- c(state, "no")
      }
```

```
    }

    evid <- setEvidence(model, nodes = node_names, states = state[ini_index])

    query <- querygrain(evid)

    pred <- c(pred, ifelse(query$S[1] > query$S[2], "no", "yes"))

  }

  confmatrix <- table(True = data$S, Prediction = pred)

  return(confmatrix)
}

#predict using L and B (column 4 and 5)
markov_blanket_pred <- pred_bn(model2_compile, test, ini_index = c(4, 5))
markov_blanket_pred
```

```
##      Prediction
## True   no yes
##   no  330 153
##   yes 128 389
```

```
accuracy <- sum(diag(markov_blanket_pred))/sum(markov_blanket_pred)
cat('The accuracy rate is ', accuracy*100, '%.', sep = '')
```

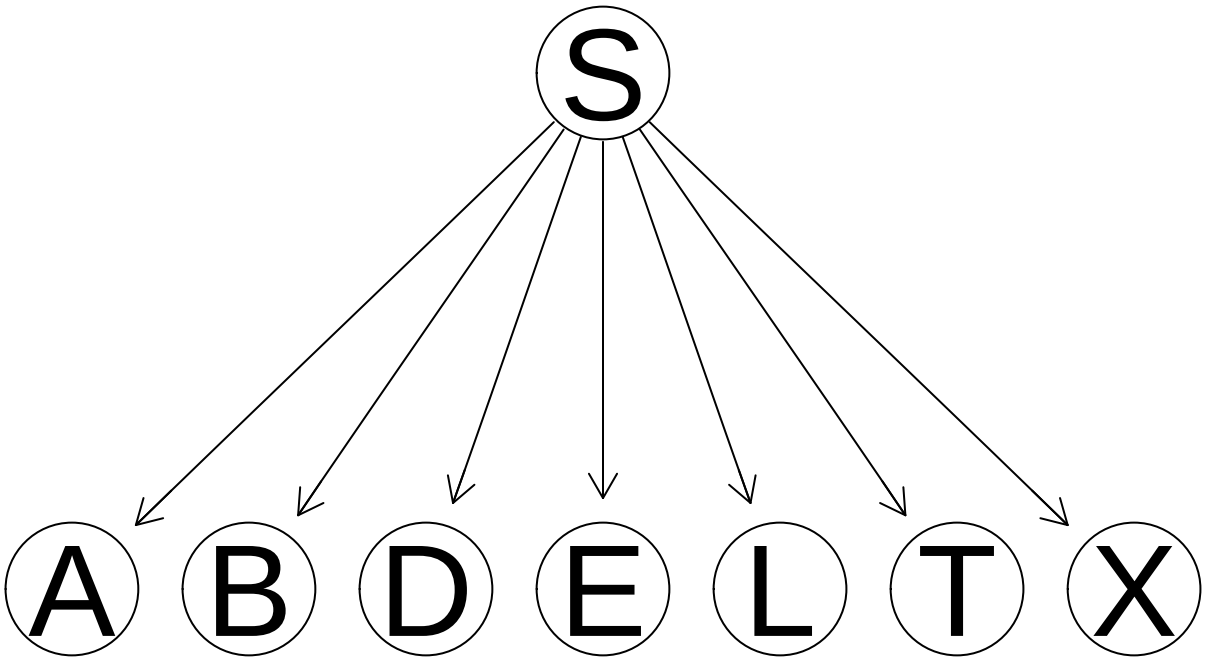
```
## The accuracy rate is 71.9%.
```

From the confusion matrix, it is clear that there are 153 false negative and 128 false positive and the accuracy rate of the inference is 71.9.

Question 4

The Naive Bayes classifier assumes that the features are independent pairwise, therefore, the dependencies between nodes are changed as is shown in the following figure:

```
naivebn <-  model2network("[A|S][S][T|S][L|S][B|S][D|S][E|S][X|S]")
graphviz.plot(naivebn)
```




```
predictbn <- function(bn, bnLabels = labels(asia)[[2]][-2]){
  fit <- bn.fit(bn, train, method = "bayes")
  grain <- as.grain(fit)
  grain <- compile(grain)
  predS <- factor(rep(0,dim(test)[1]), c("yes", "no"))
  labelindex <- NULL
  for (var in bnLabels) {
    labelindex <-  c(labelindex, which(labels(asia)[[2]] == var))
  }
  for (i in 1:dim(test)[1]) {
    evi <-  rep("yes", length(labelindex))
    evi[test[i,labelindex] == "no"]  <-  "no"
    tempgrain  <-  setEvidence(grain, bnLabels, evi)
    predS[i]  <-  ifelse(querygrain(tempgrain, "S")$S[2]>0.5, "yes", "no")
  }
  return(predS)
}


predS4 <-  predictbn(naivebn)
mat <- table(True =test$S, Prediction = predS4)
mat
```

```
##      Prediction
## True  yes  no
##   no  130 353
##   yes 320 197
```

```
accuracy <- 1-sum(diag(mat))/sum(mat)
cat('The accuracy rate is ', accuracy*100, '%.', sep = '')
```

```
## The accuracy rate is 67.3%.
```

The Naive Bayes network was created with respect to S. Predictions were then made and a confusion matrix was created. This time the confusion matrix had different values compared to the earlier matrices as can be observed above. There are 130 false negative samples and 197 false positive samples with accuracy rate equals to 67.3 .

Question 5

By comparing the results in exercises 2-4, exercise 2 and 3 hold the same confusion matrix, which possibly results from the similar inference expression

Suppose the whole set of random variables is Y, and the target is S. The inference in exercise 2 can be indicated by the following derivation.

$$p(S|Y \setminus S) = \frac{p(Y)}{p(Y \setminus S)} \tag{4.1}$$

$$= \frac{\prod_{i=1}^{N} p(X_i|Pa(X_i))}{\sum_S \prod_{i=1}^{N} p(X_i|Pa(X_i))} \tag{4.2}$$

$$= \frac{p(S)p(L|S)p(B|S)}{p(B,L)} \tag{4.3}$$

From the equation above, the distribution of S given other variables only relies on B and L. Similar to the exercise 2, inference in exercise 3 is shown following:

$$p(S|MB) = \frac{p(S,MB)}{p(MB)} \tag{4.4}$$

$$= \frac{\prod_{X_i \in (MB,S)} p(X_i|Pa(X_i))}{\sum_S \prod_{X_i \in (MB,S)} p(X_i|Pa(X_i))} \tag{4.5}$$

$$= \frac{p(S)p(L|S)p(B|S)}{p(B,L)} \tag{4.6}$$

From the equation (4.3) and (4.6), it is clear that both inferences are the same as the set of nodes with which S has dependencies equals to the Markov blanket.

For Naive Bayes classifier the equation is given as follow:

$$p(S|Y \quad S) = \frac{p(Y)}{p(Y \quad S)} \tag{4.7}$$

$$= \frac{\prod_{i=1}^{N} p(X_i|Pa(X_i))}{\sum_S \prod_{i=1}^{N} p(X_i|Pa(X_i))} \tag{4.8}$$

$$= \frac{p(S)p(A|S)p(B|S)p(D|S)p(E|S)p(L|S)p(T|S)}{p(A,B,D,E,L,T)} \tag{4.9}$$

The equation (4.9) shows that the distribution of S relies on all other nodes differing from other two exercises. Therefore, the classification results are not expected to be the same.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
library(bnlearn)
library(gRain)
data('asia')

set.seed(123456789)
s1 <- random.graph(names(asia), num = 1, method = "ordered")

restart_1 <- hc(asia)
restart_2 <- hc(asia,restart = 20)

initial_1 <- hc(asia,start = s1)
initial_2 <- hc(asia)

iss_1 <- hc(asia, score = "bde", iss = 1)
iss_2 <- hc(asia, score = "bde", iss = 2)

score_1 <- hc(asia, score = "bde")
score_2 <- hc(asia, score = "aic")

show_difference <- function(g1, g2)
{
  equal <- all.equal(g1, g2)
  cp1 <- cpdag(g1)
  cp2 <- cpdag(g2)
  par(mfrow = c(1, 2))
  graphviz.compare(g1, g2)
  res <- list("all.equal" = g1, "cpdag" = list(cp1, cp2))
  return(res)
}

#show_difference(restart_1, restart_2)
show_difference(initial_1, initial_2)
#show_difference(score_1, score_2)
#show_difference(iss_1, iss_2)
rm(list=ls())
dag <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

# separate dataset
set.seed(123456789)
N <- dim(asia)
train_id <- sample(1:N,floor(N*0.8))
train <- asia[train_id,]
test_id <- setdiff(1:N,train_id)
test <- asia[test_id,]
pre_train_dag <- dag
BN_infer <- function(train, test, target, pre_train_dag, BN_flag = 1, appr = 0, use_mb = 0){

  # structure learning of BNs
  if(BN_flag == 1){
    bn_model_tr <- hc(train)
  }else{
```

```
    bn_model_tr <- pre_train_dag
  }

  # fit parameters
  fitted <- bn.fit(bn_model_tr,data = train)

  # tranform the class to grain class
  bn_gr <- as.grain(fitted)

  # compile the BNs
  bn_cp <- compile(bn_gr)

  # set Evidence
  if(use_mb == 0){
    evid_node <- colnames(test)
    evid_node <- setdiff(evid_node,target)
  }else{
    evid_node <- mb(fitted,target)
  }

  evid_val <- test[,evid_node]

  classifier <- function(evidence_val){
    tt <- setEvidence(bn_cp, nodes = evid_node, states = evidence_val)
    res <- names(which.max(querygrain(tt, nodes = target)[[1]]))
    return(res)
  }

  # results of test
  pred_y <- sapply(as.data.frame(t(evid_val)), FUN = classifier)
  return(list(pred_y,'BN structure' = bn_model_tr))

}

# test function
# train BN
res_1 <- BN_infer(train = train,
                  test = test,
                  target = 'S',
                  pre_train_dag = dag,
                  BN_flag = 1,
                  use_mb = 0)
# pretrain BN
res_2 <- BN_infer(train = train,
                    test = test,
                    target = 'S',
                    pre_train_dag = dag,
                    BN_flag = 0,
                    use_mb = 0)
par(mfrow = c(1, 2))
graphviz.compare(res_1[[2]], dag)
tt <- table(True = test[,'S'],Prdiction = res_1[[1]])
accuracy <- sum(diag(tt))/sum(tt)
```

```
print(tt)
cat('The accuracy rate is ', accuracy*100, '%.', sep = '')
tt <- table(True = test[,'S'],Prediction = res_2[[1]])
accuracy <- sum(diag(tt))/sum(tt)
print(tt)
cat('The accuracy rate is ', accuracy*100, '%.', sep = '')
# Hill-Climbing on train set
model2 <- hc(train, restart = 10)

model2_fit <- bn.fit(model2, train)
model2_grain <- as.grain(model2_fit)
model2_compile <- compile(model2_grain)

mb(model2_fit, "S")
# prediction function
pred_bn <- function(model, data, ini_index = c(1,3:8)) {

  node_names <- colnames(data[ini_index])

  pred <- c()

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

    state <- c()

    for (j in 1:ncol(data)) {

      if (data[i,j] == "yes") {
        state <- c(state, "yes")
      } else {
        state <- c(state, "no")
      }

    }

    evid <- setEvidence(model, nodes = node_names, states = state[ini_index])

    query <- querygrain(evid)

    pred <- c(pred, ifelse(query$S[1] > query$S[2], "no", "yes"))

  }

  confmatrix <- table(True = data$S, Prediction = pred)

  return(confmatrix)
}

#predict using L and B (column 4 and 5)
markov_blanket_pred <- pred_bn(model2_compile, test, ini_index = c(4, 5))
markov_blanket_pred

accuracy <- sum(diag(markov_blanket_pred))/sum(markov_blanket_pred)
```

```
cat('The accuracy rate is ', accuracy*100, '%.', sep = '')
naivebn <-  model2network("[A|S][S][T|S][L|S][B|S][D|S][E|S][X|S]")
graphviz.plot(naivebn)
predictbn <- function(bn, bnLabels = labels(asia)[[2]][-2]){
  fit <- bn.fit(bn, train, method = "bayes")
  grain <- as.grain(fit)
  grain <- compile(grain)
  predS <- factor(rep(0,dim(test)[1]), c("yes", "no"))
  labelindex <- NULL
  for (var in bnLabels) {
    labelindex <-  c(labelindex, which(labels(asia)[[2]] == var))
  }
  for (i in 1:dim(test)[1]) {
    evi <-  rep("yes", length(labelindex))
    evi[test[i,labelindex] == "no"]  <-  "no"
    tempgrain  <-  setEvidence(grain, bnLabels, evi)
    predS[i]  <-  ifelse(querygrain(tempgrain, "S")$S[2]>0.5, "yes", "no")
  }
  return(predS)
}


predS4 <-  predictbn(naivebn)
mat <- table(True =test$S, Prediction = predS4)
mat
accuracy <- 1-sum(diag(mat))/sum(mat)
cat('The accuracy rate is ', accuracy*100, '%.', sep = '')
```