Advanced Machine Learning lab1

Akshath Srinivas(akssr921)

2023-09-06

Question 1

Answer:

The Hill-Climbing algorithm starts with an initial BN structure with empty dag and iteratively makes small local changes by greedy search to improve the structure's score based on a scoring metric (e.g. BIC). These multiple runs can return non-equivalent Bayesian network(BN) structures and does not guarantee BN global optima structure. in other words, HC is not asymptotically correct under faithfulness, i.e. it may get trapped in local optima.

```r
#question 1

data(asia)

hill_climbing1 <- hc(asia,score ='bic',iss=2,restart = 10,max.iter = 100)

hill_climbing2 <- hc(asia, score = "bde", iss = 2)

hill_climbing1
```

```
##
##   Bayesian network learned via Score-based methods
##
##   model:
##    [A][T][B][S|B][L|S][E|T:L][X|E][D|B:E]
##   nodes:                                 8
##   arcs:                                  7
##     undirected arcs:                     0
##     directed arcs:                       7
##   average markov blanket size:           2.25
##   average neighbourhood size:            1.75
##   average branching factor:              0.88
##
##   learning algorithm:                    Hill-Climbing
##   score:                                 BIC (disc.)
##   penalization coefficient:              4.258597
##   tests used in the learning procedure:  284
##   optimized:                             TRUE
```

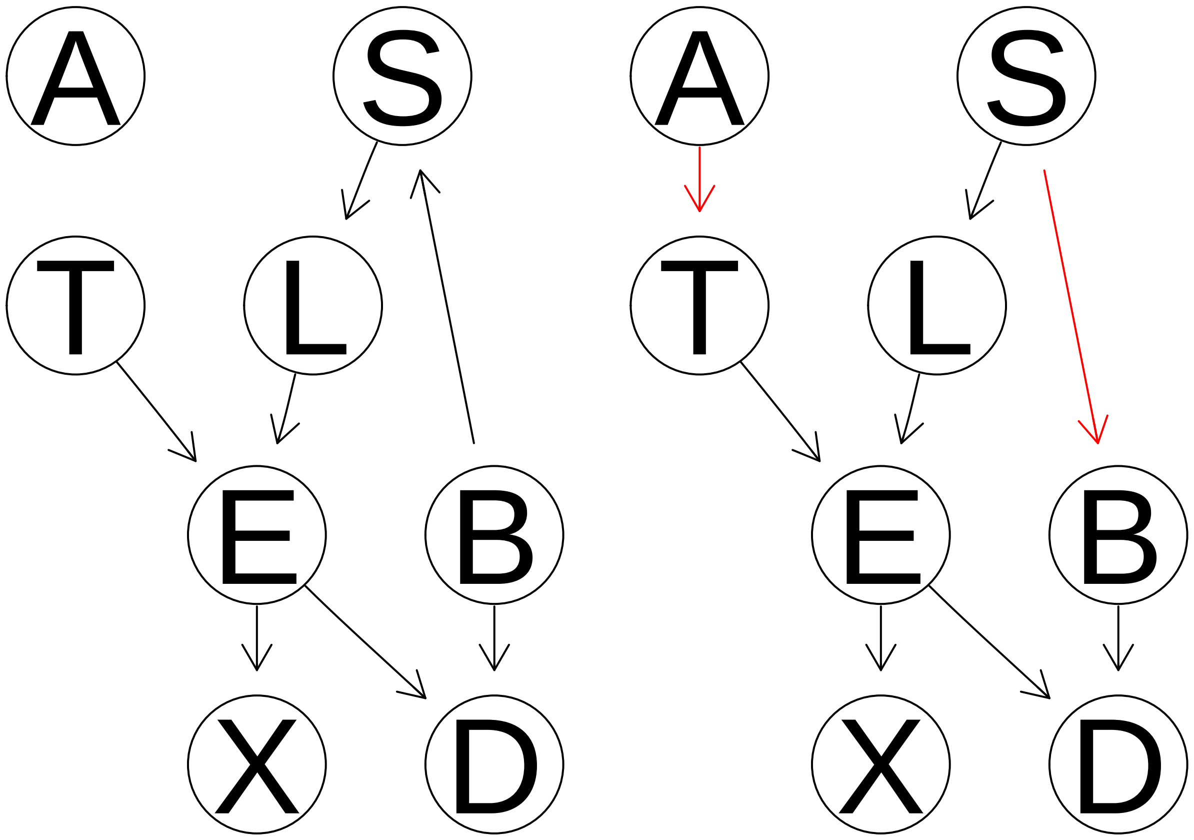
```r
hill_climbing2
```

```
##
##   Bayesian network learned via Score-based methods
##
##   model:
##    [A][S][T|A][L|S][B|S][E|T:L][X|E][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:                                 Bayesian Dirichlet (BDe)
##   graph prior:                           Uniform
##   imaginary sample size:                 2
##   tests used in the learning procedure:  84
##   optimized:                             TRUE
```

Comparing the two Hill Climbing dags.

```
par(mfrow = c(1, 2))

graphviz.compare(hill_climbing1,hill_climbing2)
```



Question 2

Answer: After comparing our result(approximate inference & exact inference) with True Asian BN, the confusion matrix and accuracy is same.

```r
#question 2
exact_approx_infernece<-function(dag,learn,test,exact_approx){
  #fitting the data
  model_dag <- bn.fit(dag,data=learn, method = 'mle')

  #compiling
  bn_comp <- compile(as.grain(model_dag))

  #setting the nodes, state of observed data for evidence
  nodes <- colnames(test)[-2]
  data_observed <- test[,nodes]

  #initializing prediction vector
  pred_vec<-rep(NA,nrow(data_observed))

  #looping over test points for prediction
  for (i in 1:nrow(data_observed)) {

    if(exact_approx==0){
      set_evid <- setEvidence(bn_comp, nodes = nodes, states =t(data_observed)[,i])
      pred_vec[i] <- names(which.max(querygrain(object = set_evid, nodes = 'S')[[1]]))
    }else if(exact_approx==1){
      posterior <- cpquery(model_dag, event = (S == "yes"), evidence = as.list(data_observed[i,]), meth
      pred_vec[i]<-ifelse(posterior > 0.5, "yes", "no")
    }
    # else if(exact_approx==2){
    #   str = paste("(", names(data_observed[i,]), " == '",
    #               sapply(data_observed[i,], as.character), "')",
    #               sep = "", collapse = " , ")
    #   print(list(str))
    #   post <- cpdist(model_dag,nodes='S', evidence = str)
    #   pred_vec[i] <- ifelse(mean(post=='yes') > 0.5, "yes", "no")
    # }
    else{
      break
    }

  }


  # True classes from the test data
  true_class <- test_data$S

  # Create a confusion matrix
  conf_matrix <- table(pred_vec, true_class)


  acc<-sum(diag(conf_matrix))/sum(conf_matrix)

  return(list(conf_matrix,acc))
}

# Split the dataset into 80% for learning and 20% for testing
set.seed(123)  # Setting random seed for reproducibility
sample_indices <- sample(1:nrow(asia), 0.8 * nrow(asia))
learn_data <- asia[sample_indices, ]
test_data <- asia[-sample_indices, ]
```

```
#structure learning
structure_learn <- hc(asia, score = "bde", iss = 2)
myexact_confusionMatrix <- exact_approx_infernece(structure_learn,
                                                  learn_data,
                                                  test_data,
                                                  exact_approx=0)[[1]]
myexact_acc <- exact_approx_infernece(structure_learn,
                                      learn_data,
                                      test_data,
                                      exact_approx=0)[[2]]
```

Our Bayesian network Confusion matrix and accuracy calculated as exact inference can be seen below.

```
print(myexact_confusionMatrix)
```

```
##          true_class
## pred_vec  no yes
##      no  359 116
##      yes 141 384
```

```
cat('The accuracy of our BN is: ',myexact_acc*100,'%')
```

```
## The accuracy of our BN is:  74.3 %
```

```
#approximate inference using cpquery
myapprox_confusionMatrix <- exact_approx_infernece(structure_learn,
                                                    learn_data,
                                                    test_data,
                                                 exact_approx=1)[[1]]
myapprox_acc <- exact_approx_infernece(structure_learn,
                                       learn_data,
                                       test_data,
                                       exact_approx=1)[[2]]
```

Our Bayesian network Confusion matrix and accuracy calculated as Approximate inference can be seen below.

```
print(myapprox_confusionMatrix)
```

```
##          true_class
## pred_vec  no yes
##      no  359 116
##      yes 141 384
```

```
cat('The accuracy of our BN using Approximate inference is: ',myapprox_acc*100,'%')
```

```
## The accuracy of our BN using Approximate inference is:  74.3 %
```

```
#comparing with true asian BN dag
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
true_confusionMatrix <- exact_approx_infernece(dag,
                                                learn_data,
                                                test_data,
                                            exact_approx=1)[[1]]
true_acc <- exact_approx_infernece(dag,
                                    learn_data,
                                    test_data,
                                    exact_approx=1)[[2]]
```

True Asian BN Confusion matrix and accuracy can be seen below.

```
print(true_confusionMatrix)
```

```
##          true_class
## pred_vec  no yes
##      no  359 116
##      yes 141 384
```

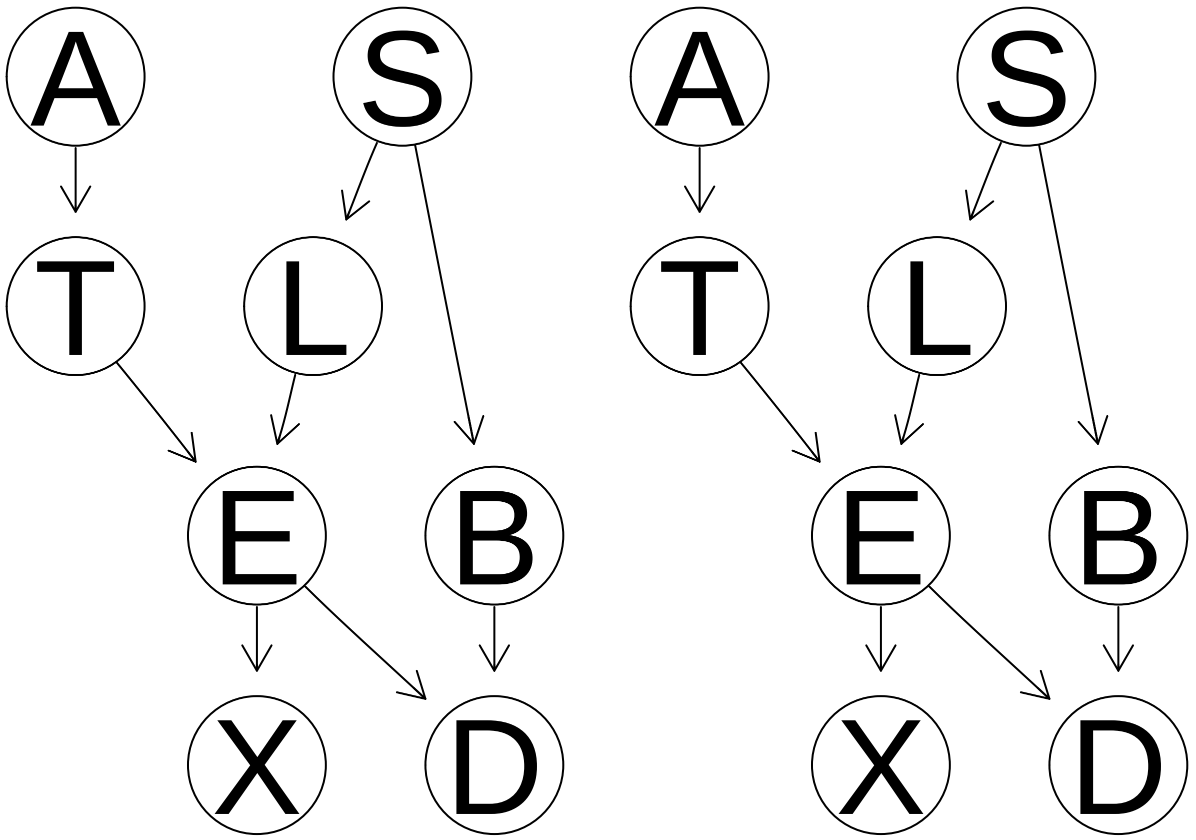
```
cat('The accuracy of our BN using Approximate inference is: ',true_acc*100,'%')
```

```
## The accuracy of our BN using Approximate inference is:  74.3 %
```

Comparing our DAG with True Asian DAG.

Below we can see the plot for our BN and True Asian BN with no difference.

```
par(mfrow = c(1, 2))
graphviz.compare(structure_learn,dag)
```



Question 3

Answer: Below we can see the confusion matrix and accuracy calculated using Markov Blanket.

```r
# Define a function for classification using the Markov blanket
classify_with_markov_blanket <- function(dag, learn, test) {
  # Learn the model
  model_dag <- bn.fit(dag, data = learn, method = 'mle')

  # Compile the model
  bn_comp <- compile(as.grain(model_dag))

  # Extract the Markov blanket of S
  mb_of_S <- mb(model_dag, "S")

  # Initialize the prediction vector
  pred_vec <- rep(NA, nrow(test))

  evidence <- test[, mb_of_S]

  # Loop over test points for prediction
  for (i in 1:nrow(test)) {

    # Set the evidence in the model
    set_evid <- setEvidence(bn_comp, nodes = names(evidence), states = t(evidence)[,i])

    # Perform classification
    pred_vec[i] <- names(which.max(querygrain(object = set_evid, nodes = 'S')[[1]]))
  }

  # True classes from the test data
  true_class <- test$S

  # Create a confusion matrix
  conf_matrix <- table(pred_vec, true_class)

  acc <- sum(diag(conf_matrix)) / sum(conf_matrix)

  return(list(conf_matrix, acc))
}

structure_learn <- hc(asia, score = "bde", iss = 2)
# Split the dataset into 80% for learning and 20% for testing
set.seed(123)  # Setting random seed for reproducibility
sample_indices <- sample(1:nrow(asia), 0.8 * nrow(asia))
learn_data <- asia[sample_indices, ]
test_data <- asia[-sample_indices, ]

# Classify using the Markov blanket of S for the learned structure
markov_blanket_confusionMatrix <- classify_with_markov_blanket(structure_learn, learn_data, test_data)
markov_blanket_acc <- markov_blanket_confusionMatrix[[2]]

print(markov_blanket_confusionMatrix)
```

```
## [[1]]
##          true_class
## pred_vec  no yes
##      no  359 116
##      yes 141 384
##
## [[2]]
## [1] 0.743
```

```
cat('The accuracy of our BN using MB Exact inference is: ',markov_blanket_acc*100,'%')
```

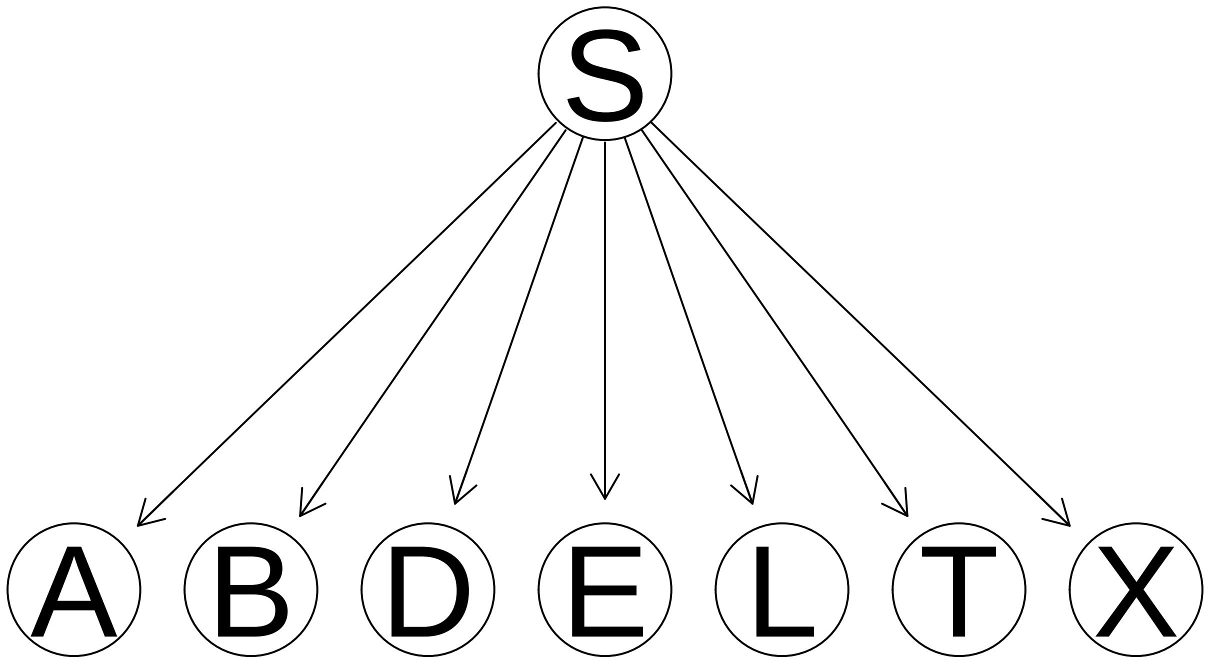
```
## The accuracy of our BN using MB Exact inference is:  74.3 %
```

Question 4

Answer: Below we can see the confusion matrix and accuracy calculated using Naive bayes classifier.

```
#question 4

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



```
naive_bayes_infernece<-function(learn,test,mod_str){

  nb_dag <- empty.graph(nodes = colnames(asia))
  modelstring(nb_dag) <- mod_str
  #fitting the data
  model_dag <- bn.fit(nb_dag,data=learn, method = 'bayes')

  #compiling
  bn_comp <- compile(as.grain(model_dag))

  #setting the nodes, state of observed data for evidence
  nodes <- colnames(test)[-2]
  data_observed <- test[,nodes]

  #initializing prediction vector
  pred_vec<-rep(NA,nrow(data_observed))

  #looping over test points for prediction
  for (i in 1:nrow(data_observed)) {
    set_evid <- setEvidence(bn_comp, nodes = nodes, states =t(data_observed)[,i])
    querygr<- querygrain(object = set_evid, nodes = 'S')[[1]]
    pred_vec[i]<-ifelse(querygr['yes'] > 0.5, "yes", "no")

  }

  # True classes from the test data
  true_class <- test_data$S

  # Create a confusion matrix
  conf_matrix <- table(pred_vec, true_class)


  acc<-sum(diag(conf_matrix))/sum(conf_matrix)

  return(list(conf_matrix,acc))
}

naivebayes_confusionMatrix <- naive_bayes_infernece(learn_data,
                                                    test_data,
                                                    mod_str='[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]')[[
naivebayes_acc <- naive_bayes_infernece(learn_data,
                                        test_data,                                mod_str='[S][A|S][T

print(naivebayes_confusionMatrix)
```

```
##         true_class
## pred_vec  no yes
##      no  390 179
##      yes 110 321
```

```
cat('The accuracy of our BN using Approximate inference is: ',naivebayes_acc*100,'%')
```

```
## The accuracy of our BN using Approximate inference is:  71.1 %
```

Question 5

Answer:

Exercise 2 Inference:

- Inference Expression:
$$p(S|Y \setminus S) = \frac{p(S) \cdot p(L|S) \cdot p(B|S)}{p(B, L)}$$
- Dependencies: Node 'S' relies on variables B and L.

Exercise 3 Inference, Markov Blanket(MB):

- Inference Expression:
$$p(S|MB) = \frac{p(S) \cdot p(L|S) \cdot p(B|S)}{p(B, L)}$$
- Dependencies: Also relies on variables B and L.

Exercise 4 Inference, Naive Bayes Classifier:

- Inference Expression:
$$p(S|Y \quad S) = \frac{p(S) \cdot p(A|S) \cdot p(B|S) \cdot p(D|S) \cdot p(E|S) \cdot p(L|S) \cdot p(T|S)}{p(A, B, D, E, L, T)}$$
- Dependencies: Relies on all other nodes (A,B,D,E,L,T), unlike the previous exercises.

From the above expressions we can say that, the Naive Bayes classifier relies on all other nodes, making it distinct from the previous exercises. Therefore, we get different results.

In the Naive Bayes classifier, the results depend on all available features (A,B,D,E,L,T), assuming independence between these features given the target variable 'S'. This is a simplifying assumption, which makes it "naive" and it's often used in tasks when you assume that the features are conditionally independent given the target.

Appendix

```r
knitr::opts_chunk$set(warning = FALSE,echo = TRUE)
library(bnlearn)
library(gRain)
library(gridExtra)

#question 1

data(asia)

hill_climbing1 <- hc(asia,score ='bic',iss=2,restart = 10,max.iter = 100)

hill_climbing2 <- hc(asia, score = "bde", iss = 2)

hill_climbing1
hill_climbing2


par(mfrow = c(1, 2))
```

```
graphviz.compare(hill_climbing1,hill_climbing2)


#question 2
exact_approx_infernece<-function(dag,learn,test,exact_approx){
  #fitting the data
  model_dag <- bn.fit(dag,data=learn, method = 'mle')

  #compiling
  bn_comp <- compile(as.grain(model_dag))

  #setting the nodes, state of observed data for evidence
  nodes <- colnames(test)[-2]
  data_observed <- test[,nodes]

  #initializing prediction vector
  pred_vec<-rep(NA,nrow(data_observed))

  #looping over test points for prediction
  for (i in 1:nrow(data_observed)) {

    if(exact_approx==0){
      set_evid <- setEvidence(bn_comp, nodes = nodes, states =t(data_observed)[,i])
      pred_vec[i] <- names(which.max(querygrain(object = set_evid, nodes = 'S')[[1]]))
    }else if(exact_approx==1){
      posterior <- cpquery(model_dag, event = (S == "yes"), evidence = as.list(data_observed[i,]), meth
      pred_vec[i]<-ifelse(posterior > 0.5, "yes", "no")
    }
    # else if(exact_approx==2){
    #   str = paste("(", names(data_observed[i,]), " == '",
    #               sapply(data_observed[i,], as.character), "')",
    #               sep = "", collapse = " , ")
    #   print(list(str))
    #   post <- cpdist(model_dag,nodes='S', evidence = str)
    #   pred_vec[i] <- ifelse(mean(post=='yes') > 0.5, "yes", "no")
    # }
    else{
      break
    }

  }


  # True classes from the test data
  true_class <- test_data$S

  # Create a confusion matrix
  conf_matrix <- table(pred_vec, true_class)


  acc<-sum(diag(conf_matrix))/sum(conf_matrix)

  return(list(conf_matrix,acc))
}

# Split the dataset into 80% for learning and 20% for testing
set.seed(123)  # Setting random seed for reproducibility
sample_indices <- sample(1:nrow(asia), 0.8 * nrow(asia))
learn_data <- asia[sample_indices, ]
test_data <- asia[-sample_indices, ]


#structure learning
structure_learn <- hc(asia, score = "bde", iss = 2)
myexact_confusionMatrix <- exact_approx_infernece(structure_learn,
                                                learn_data,
                                                test_data,
                                                exact_approx=0)[[1]]
myexact_acc <- exact_approx_infernece(structure_learn,
                                      learn_data,
                                      test_data,
                                      exact_approx=0)[[2]]

print(myexact_confusionMatrix)
cat('The accuracy of our BN is: ',myexact_acc*100,'%')

#approximate inference using cpquery
myapprox_confusionMatrix <- exact_approx_infernece(structure_learn,
                                                   learn_data,
                                                   test_data,
                                                exact_approx=1)[[1]]
myapprox_acc <- exact_approx_infernece(structure_learn,
                                       learn_data,
                                       test_data,
                                       exact_approx=1)[[2]]

print(myapprox_confusionMatrix)
cat('The accuracy of our BN using Approximate inference is: ',myapprox_acc*100,'%')

#comparing with true asian BN dag
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
true_confusionMatrix <- exact_approx_infernece(dag,
                                                    learn_data,
                                                    test_data,
                                                exact_approx=1)[[1]]
true_acc <- exact_approx_infernece(dag,
                                        learn_data,
                                        test_data,
                                        exact_approx=1)[[2]]

print(true_confusionMatrix)
cat('The accuracy of our BN using Approximate inference is: ',true_acc*100,'%')


par(mfrow = c(1, 2))
graphviz.compare(structure_learn,dag)
```

```r
# Define a function for classification using the Markov blanket
classify_with_markov_blanket <- function(dag, learn, test) {
  # Learn the model
  model_dag <- bn.fit(dag, data = learn, method = 'mle')

  # Compile the model
  bn_comp <- compile(as.grain(model_dag))

  # Extract the Markov blanket of S
  mb_of_S <- mb(model_dag, "S")

  # Initialize the prediction vector
  pred_vec <- rep(NA, nrow(test))

  evidence <- test[, mb_of_S]

  # Loop over test points for prediction
  for (i in 1:nrow(test)) {

    # Set the evidence in the model
    set_evid <- setEvidence(bn_comp, nodes = names(evidence), states = t(evidence)[,i])

    # Perform classification
    pred_vec[i] <- names(which.max(querygrain(object = set_evid, nodes = 'S')[[1]]))
  }

  # True classes from the test data
  true_class <- test$S

  # Create a confusion matrix
  conf_matrix <- table(pred_vec, true_class)

  acc <- sum(diag(conf_matrix)) / sum(conf_matrix)

  return(list(conf_matrix, acc))
}

structure_learn <- hc(asia, score = "bde", iss = 2)
# Split the dataset into 80% for learning and 20% for testing
set.seed(123)  # Setting random seed for reproducibility
sample_indices <- sample(1:nrow(asia), 0.8 * nrow(asia))
learn_data <- asia[sample_indices, ]
test_data <- asia[-sample_indices, ]

# Classify using the Markov blanket of S for the learned structure
markov_blanket_confusionMatrix <- classify_with_markov_blanket(structure_learn, learn_data, test_data)
markov_blanket_acc <- markov_blanket_confusionMatrix[[2]]

print(markov_blanket_confusionMatrix)
cat('The accuracy of our BN using MB Exact inference is: ',markov_blanket_acc*100,'%')

#question 4
```

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


naive_bayes_infernece<-function(learn,test,mod_str){

  nb_dag <- empty.graph(nodes = colnames(asia))
  modelstring(nb_dag) <- mod_str

  #fitting the data
  model_dag <- bn.fit(nb_dag,data=learn, method = 'bayes')

  #compiling
  bn_comp <- compile(as.grain(model_dag))

  #setting the nodes, state of observed data for evidence
  nodes <- colnames(test)[-2]
  data_observed <- test[,nodes]

  #initializing prediction vector
  pred_vec<-rep(NA,nrow(data_observed))

  #looping over test points for prediction
  for (i in 1:nrow(data_observed)) {
    set_evid <- setEvidence(bn_comp, nodes = nodes, states =t(data_observed)[,i])
    querygr<- querygrain(object = set_evid, nodes = 'S')[[1]]
    pred_vec[i]<-ifelse(querygr['yes'] > 0.5, "yes", "no")

  }

  # True classes from the test data
  true_class <- test_data$S

  # Create a confusion matrix
  conf_matrix <- table(pred_vec, true_class)


  acc<-sum(diag(conf_matrix))/sum(conf_matrix)

  return(list(conf_matrix,acc))
}

naivebayes_confusionMatrix <- naive_bayes_infernece(learn_data,
                                                    test_data,
                                                    mod_str='[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]')[[
naivebayes_acc <- naive_bayes_infernece(learn_data,
                                        test_data,                                  mod_str='[S][A|S][T

print(naivebayes_confusionMatrix)
cat('The accuracy of our BN using Approximate inference is: ',naivebayes_acc*100,'%')
```