Lab1GroupReport

aswma317, akssr921, varsi146

2023-09-07

Question 1: Show that multiple runs of the hill-climbing algorithm can return non-equivalent Bayesian network (BN) structures.

Answer: 2 BNs are said to be equivalent if,

i) they have the same adjancancies(skeleton)

ii) they have the same unshielded collider

HC starts with an empty DAG and iteratively improves the Bayesian Score by adding/removing/reversing the edges in the DAG. But it may not converge to the optimal solution(true graph) as it may get stuck in the local optimum. Hence, it is not asymptotically correct under faithfulness; i.e. regardless of the number of samples we have, it does not converge to the optimum solution.

This is visible from the below experiment:

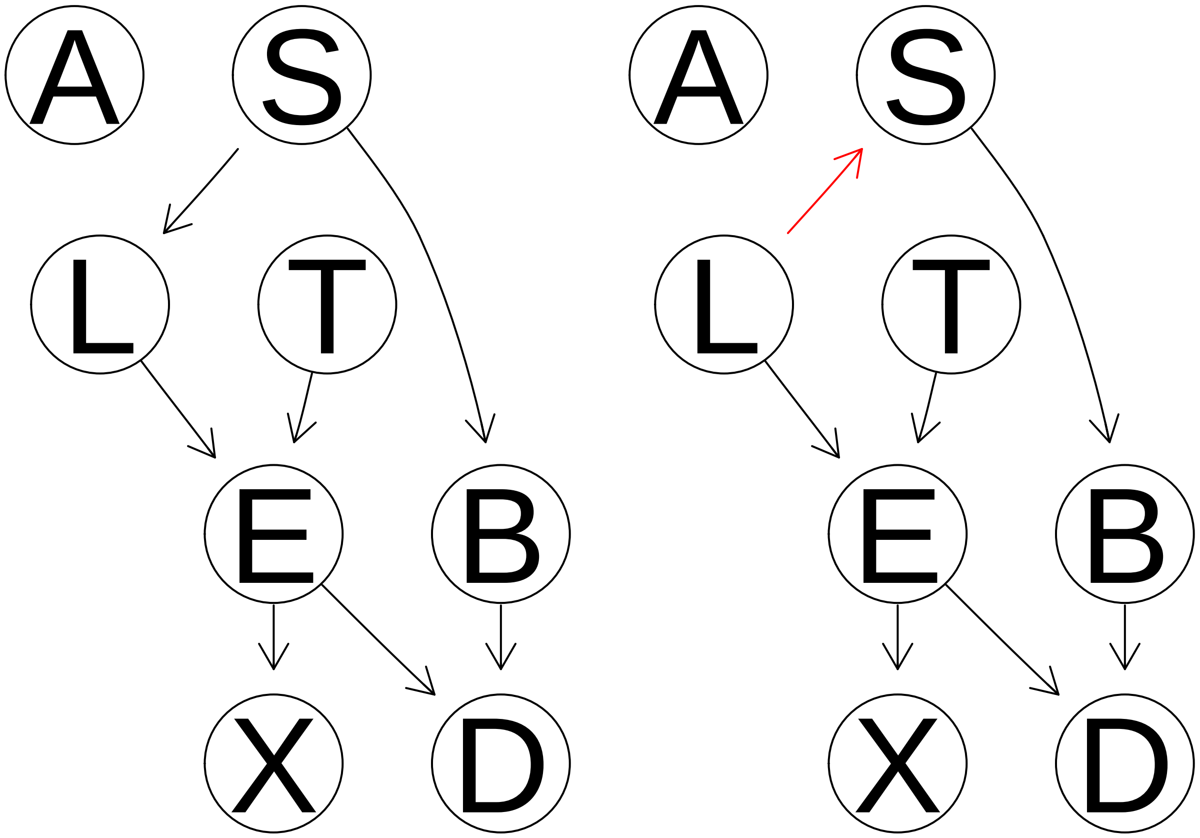
```r
graph1 <- bnlearn::hc(asia)

#Restart: when it hits the local optimum, it restarts again to explore new areas in the graph.
graph2 <- bnlearn::hc(asia, restart = 5)

print(paste("Graph 1 and Graph 2 equivalency:", all.equal(graph1, graph2)))
```

```
## [1] "Graph 1 and Graph 2 equivalency: Different arc sets"
```

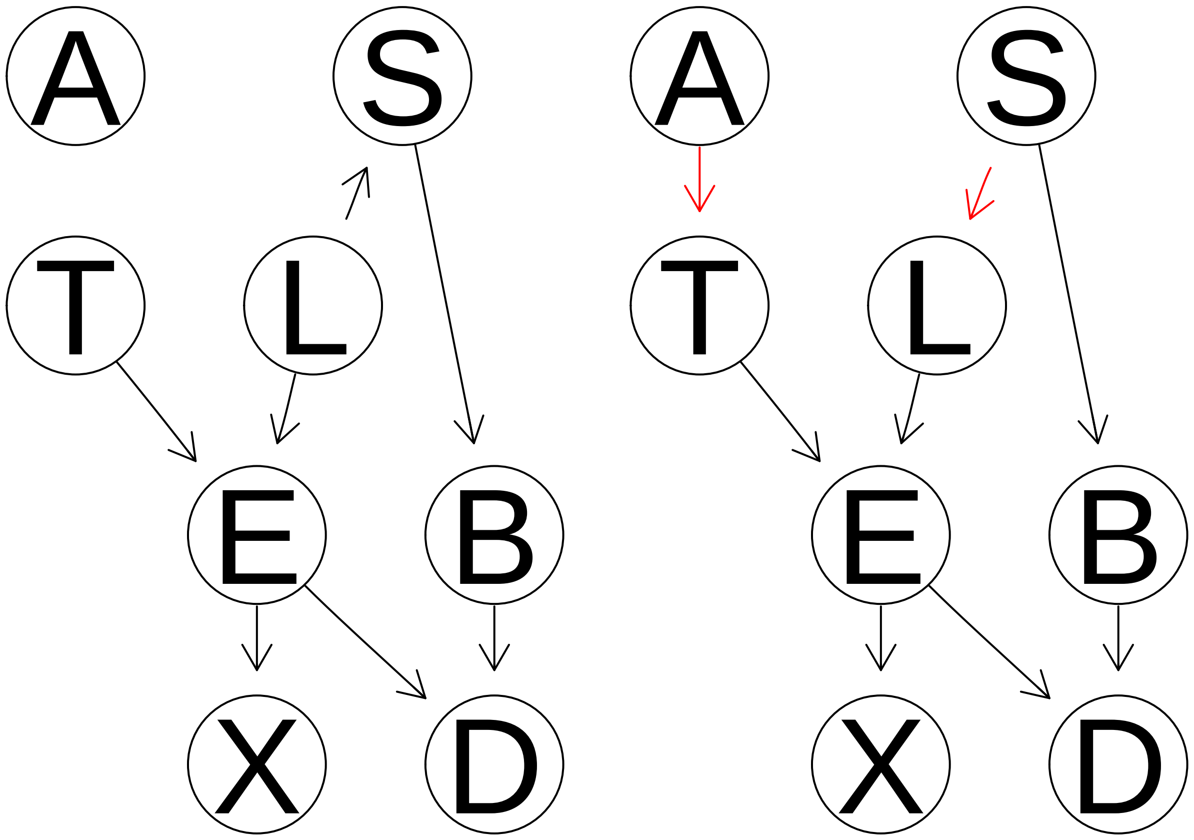
```r
par(mfrow = c(1, 2))
graphviz.compare(graph1, graph2)

#Change the initial structure
custom_initial_structure <- random.graph(names(asia), num = 1, method = "ordered")
graph3 <- bnlearn::hc(asia, start = custom_initial_structure)

print(paste("Graph 2 and Graph 3 equivalency:", all.equal(graph3, graph2)))
```

```
## [1] "Graph 2 and Graph 3 equivalency: TRUE"
```

```r
par(mfrow = c(1, 2))
graphviz.compare(graph1, graph3)
```



```r
graph4 <- bnlearn::hc(asia, score = "bde", iss = 2)
graphviz.compare(graph3, graph4)
```



In the above experiment, we changed the parameters like the restarts and the starting point of the graph. But, HC doesn't guarantee equivalent BN.

Question 2: Compute the posterior probability distribution of S for each case and classify it in the most likely class.

Answer:

```r
#Split the data
set.seed(123)
train_idx <- sample(1:nrow(asia), size = nrow(asia) * 0.8)
data_train <- asia[train_idx,]
data_test <- asia[-train_idx,]

#Learn the structure
learned_graph <- bnlearn::hc(data_train)

#Learn the parameters/conditional probability distributions
bn_fit <- bn.fit(learned_graph, data = data_train)

#Perform probabilistic inference like computing posterior probabilities,
#conditional probabilities, classifications ...
#Convert the bn.fit object to a grain
#bn_grain <- as.grain(bn_fit)
junction = compile(as.grain(bn_fit))

pred_exact_inf <- character(length = nrow(data_test))
```

```
for (i in 1:nrow(data_test)) {
  # Create an evidence object with the observed variables from the test data
  evidence <- setEvidence(junction,
                          nodes = c("A", "T", "L", "B", "E", "X", "D"),
                          states = t(data_test[i ,-2]))

  # Query the posterior probability of S(S = yes and S = no)
  posterior <- querygrain(evidence,
                          nodes = "S",
                          type = "marginal")

  # Predict the correct class
  pred_exact_inf[i] <- ifelse(posterior[[1]][1] > posterior[[1]][2],
                              "no", "yes")
}

#Confusion matrix
true_labels <- as.vector(data_test$S)
exact_confusion_matrix <- table(Actual = true_labels, Predicted = pred_exact_inf)
```

Comparing with the true BN

```
#Compare with the true DAG
true_dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

#Learn the parameters/conditional probability distributions for true dag
bn_fit_true <- bn.fit(true_dag, data = data_train)
junction = compile(as.grain(bn_fit_true))
pred_true_dag <- character(length = nrow(data_test))
for (i in 1:nrow(data_test)) {
  # Create an evidence object with the observed variables from the test data
  evidence <- setEvidence(junction,
                          nodes = c("A", "T", "L", "B", "E", "X", "D"),
                          states = t(data_test[i ,-2]))

  # Query the posterior probability of S(S = yes and S = no)
  posterior <- querygrain(evidence,
                          nodes = "S",
                          type = "marginal")

  # Predict the correct class
  pred_true_dag[i] <- ifelse(posterior[[1]][1] > posterior[[1]][2],
                             "no", "yes")
}

#Confusion matrix
true_labels <- as.vector(data_test$S)
confusion_matrix_true <- table(Actual = true_labels, Predicted = pred_true_dag)
print(paste("Exact confusion matrix:"))
```

```
## [1] "Exact confusion matrix:"
```

```r
exact_confusion_matrix
```

```
##       Predicted
## Actual  no yes
##    no  359 141
##    yes 116 384
```

```r
print(paste("True confusion matrix:"))
```

```
## [1] "True confusion matrix:"
```

```r
confusion_matrix_true
```

```
##       Predicted
## Actual  no yes
##    no  359 141
##    yes 116 384
```

```r
print(paste("Accuracy of posterior prediction using exact inference",
            sum(diag(exact_confusion_matrix))/sum(exact_confusion_matrix)))
```

```
## [1] "Accuracy of posterior prediction using exact inference 0.743"
```

```r
print(paste("Accuracy of posterior prediction of true DAG",
            sum(diag(confusion_matrix_true))/sum(confusion_matrix_true)))
```

```
## [1] "Accuracy of posterior prediction of true DAG 0.743"
```

After comparing our exact BN with the true BN, we observed that the confusion matrix and accuracy exactly matches for both.

Estimating the posterior probability using approximate inference(cpquery):

```r
#Learn the structure
learned_graph <- bnlearn::hc(data_train)

#Learn the parameters/conditional probability distributions
bn_fit <- bn.fit(learned_graph, data = data_train)

#Perform probabilistic inference like computing posterior probabilities,
#conditional probabilities, classifications ...
pred_approx_inf <- character(length = nrow(data_test))
for (i in 1:nrow(data_test)) {
  # Create an evidence object with the observed variables from the test data
  evidence <- data_test[i ,-2]

  # Query the posterior probability of S(S = yes and S = no)
  posterior <- cpquery(bn_fit,
                       event = (S == "yes"),
                       evidence = as.list(evidence),
                       method = "lw")

  # Predict the correct class
  pred_approx_inf[i] <- ifelse(posterior >= 0.5, "yes", "no")
}

#Confusion matrix
true_labels <- as.vector(data_test$S)
confusion_matrix <- table(Actual = true_labels, Predicted = pred_approx_inf)
print(paste("Accuracy of posterior prediction using approx inference",
            sum(diag(confusion_matrix))/sum(confusion_matrix)))
```

```
## [1] "Accuracy of posterior prediction using approx inference 0.743"
```

As seen above, the approximate inference using cpquery yields the same accuracy as exact inference.

Estimating the posterior probability using approximate inference(cpdist):

```
#Learn the structure
learned_graph <- bnlearn::hc(data_train)

#Learn the parameters/conditional probability distributions
bn_fit <- bn.fit(learned_graph, data = data_train)

#Perform probabilistic inference like computing posterior probabilities,
#conditional probabilities, classifications ...
pred_approx_inf <- character(length = nrow(data_test))
for (i in 1:nrow(data_test)) {
  # Create an evidence object with the observed variables from the test data
  evidence <- data_test[i ,-2]

   # Query the posterior probability of S(S = yes and S = no)
  posterior_sim <- cpdist(bn_fit,
                      nodes = "S",
                       evidence = as.list(evidence),
                       method = "lw")

  # Predict the correct class
  pred_approx_inf[i] <- ifelse(mean(posterior_sim == 'yes') >= 0.5, "yes", "no")
}

#Confusion matrix
true_labels <- as.vector(data_test$S)
confusion_matrix <- table(Actual = true_labels, Predicted = pred_approx_inf)
print(paste("Accuracy of posterior prediction using approx inference",
            sum(diag(confusion_matrix))/sum(confusion_matrix)))
```

```
## [1] "Accuracy of posterior prediction using approx inference 0.499"
```

The alternate method in approximate inference using cpdist gives an incorrect result, and we would appreciate any input leading us to the right answers.

Question 3: Classify S given observations only for the so-called Markov blanket of S

Answer:

```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
  predictions <- 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
    predictions[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(predictions , 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
## predictions  no yes
##         no  359 116
##          yes 141 384
```

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

```
## The accuracy of our BN using MB Exact inference is:  0.743
```

Markhov blanket for S is:

it's parents

its' children

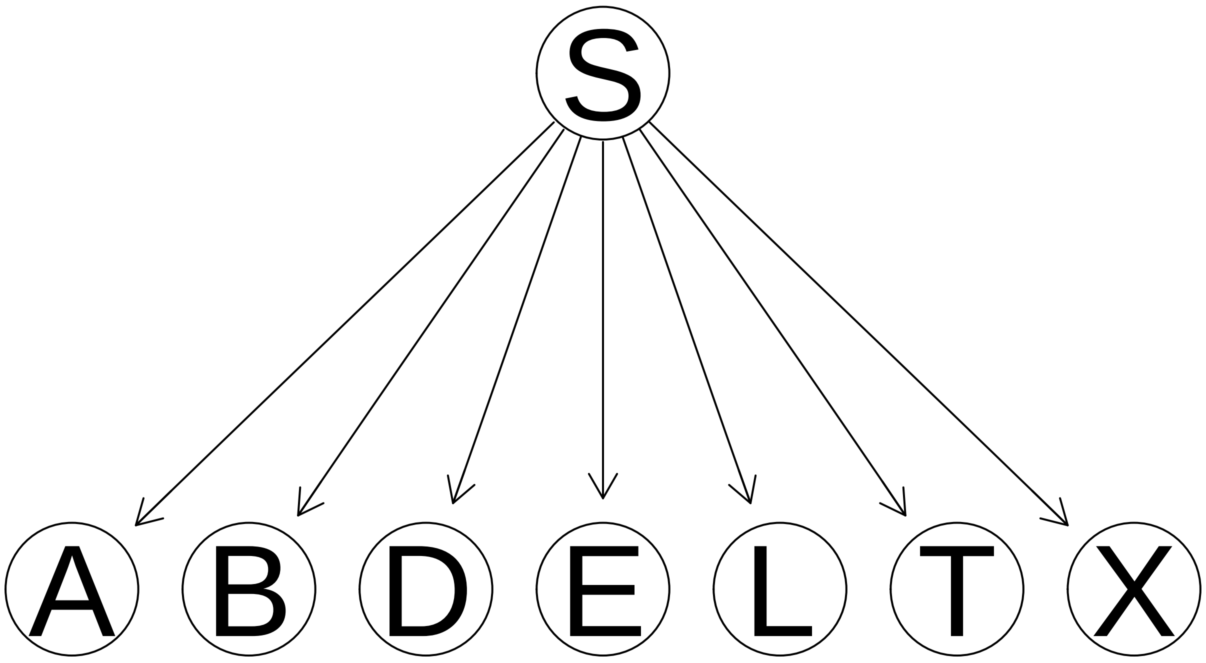
it's children's parents except S

In this case, only the children L and B are the MB for S. Due to this, the results in this section is same as the above section.

Question 4: Model the naive Bayes classifier as a BN

Answer:

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



The above is the BN created by hand.

```r
naiveBayes_classifier <- function(train_data, test_data, model_string,
                                  target_node){
  # Creating the BN by hand.
  naiveBayes_dag <- empty.graph(nodes = colnames(asia))
  modelstring(naiveBayes_dag) <- model_string

# Fitting the created BN, compiling and performing exact inference as above.
  naiveBayes_fit <- bn.fit(x = naiveBayes_dag,
                           data = train_data,
                           method = 'bayes')

  naiveBayes_compile <- compile(object = as.grain(naiveBayes_fit))
  # Data preparation
  obs_node <- setdiff(colnames(test_data), target_node)
  obs_data <- as.data.frame(test_data[,obs_node])

  pred_naiveBayes <- vector("character", length = nrow(obs_data))

  for (i in 1:nrow(obs_data)) {
    # Exact Inference.
    set_obs_data <- setEvidence(object = naiveBayes_compile, nodes = obs_node,
                                states = t(obs_data[i,]))
    query_result <- querygrain(object = set_obs_data,
                               nodes = 'S', type = 'marginal')
    pred_naiveBayes[i] <- ifelse(query_result$S['yes'] > 0.5,
                                 'yes', 'no')
  }
  return(list(pred_naiveBayes, naiveBayes_dag))
}
confusion_matrix <- function(pred, true){
  conf_matrix <- table(true$S, pred)
  accuracy <- (sum(diag(conf_matrix))/sum(conf_matrix))*100
  return(list('Confusion_Matrix' = conf_matrix, 'Accuracy' = accuracy))
}

naiveBayes_result <- naiveBayes_classifier(train_data = data_train,
                                           model_string = '[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]',
                                           test_data = data_test, target_node = 'S')


naiveBaye_cfmat <- confusion_matrix(pred = naiveBayes_result[[1]], true = data_test)

true_naiveBayes <- naive.bayes(x = data_train, training = 'S',
                               explanatory = setdiff(colnames(data_test), 'S'))

true_pred <- predict(true_naiveBayes, data_test)

true_naiveBayes_cfmat <- confusion_matrix(true_pred, data_test)

print(naiveBaye_cfmat[[1]])
```

```
##      pred
##        no yes
##   no  390 110
##   yes 179 321
```

```
cat('The accuracy of the Naive Bayes classifier: ',naiveBaye_cfmat[[2]])
```

```
## The accuracy of the Naive Bayes classifier:  71.1
```

```
print(true_naiveBayes_cfmat[[1]])
```

```
##      pred
##        no yes
##   no  390 110
##   yes 179 321
```

```
cat('The accuracy of the inbuilt Naive Bayes classifier: ',true_naiveBayes_cfmat[[2]])
```

```
## The accuracy of the inbuilt Naive Bayes classifier:  71.1
```

As seen above, our implementation results matches with the inbuilt Bayes classifier.

Question 5: Explain the difference in results between 2 and 4

The posterior probability distribution of S, given the observed variables when we perform exact or approximate inference as per Question 2 is as follows:

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

In the above expression, we consider only children of S, because from the graph it is evident that given B and L, S is independent of other variables.

In the case of Naive Bayes classifier, the posterior probability of S is as follows:

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

From the expression above it is evident that the posterior probability of S now relies on all the children A,T,L,B,E,X,D, and hence the difference in accuracy and confusion matrix i.e., we report a lower accuracy compared to previous methods.

Contribution:

All the 3 members worked on all the questions, and then the solutions were compared. All 3 members collaborated in documenting the report. For the 1st 2 questions, Aswath's solution was used, for the 3rd question, Akshath's solution was used, for the 4th question, Varun's solution was used and the final question was discussed and answered as a group in the report.