Lab1_Individual_Report

varsi146

2023-09-07

Question 1

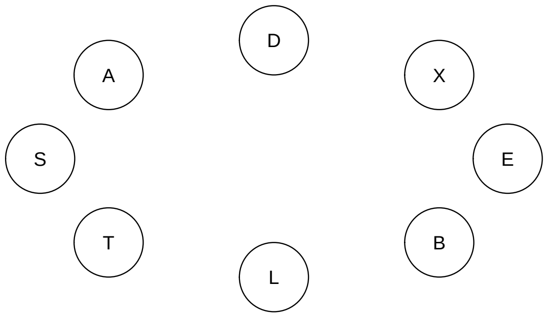
We see below the various runs of hill-climbing.

However, the multiple runs can return non-equivalent BN structures because, due to its greedy search that is an approach where it iteratively tries to maximize the score.

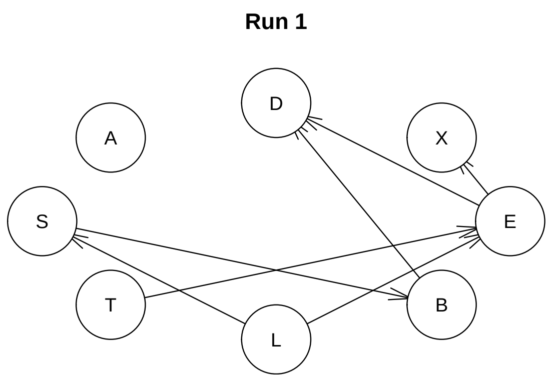
The algorithm usually starts with an empty DAG, G and a score function $Score(G, D)$, where D is the data-set. It first computes a score S_G for G and sets $S_{max} = S_G$ and $G_{max} = G$, where G_{max} is a DAG that maximizes $Score(G, D)$. It then, repeatedly adds, removes or reverses any edge in G_*, the modified DAG, that improves the score function the most by checking if $S_{G*} > S_G$ and $S_{G*} > S_{max}$ and setting $S_G = S_{G*}$ if true, and then, if $S_G > S_{max}$, then it sets $S_G = S_{max}$ and $G_{max} = G$.

During the above greedy process, it is possible that G_{max} is not a global optimum, and may perform further steps where G_{max} reaches a sub-optimal local optima. Additionally, the HC algorithm is not asymptotically correct i.e., as the data increases to a very large scale there is no guarantee of the algorithm converging to the correct solution.

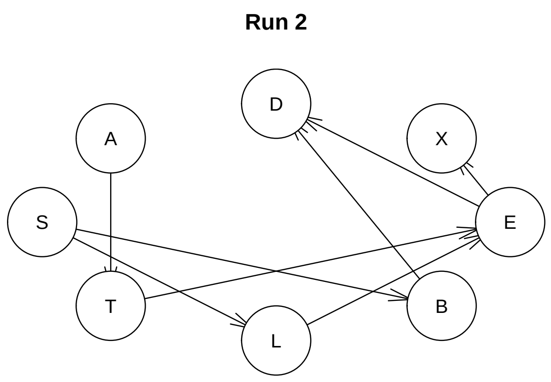
```
empty <- empty.graph(nodes = colnames(asia))
plot(empty)
```



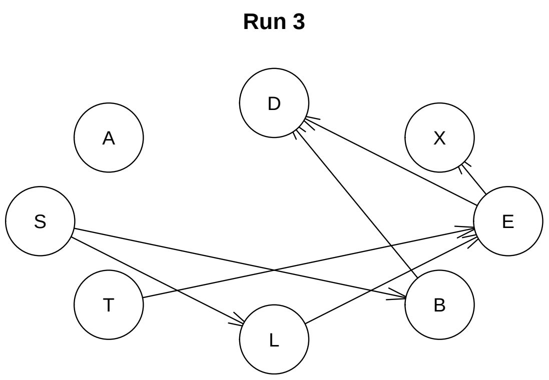
```
dag_run1 <-  hc(asia, start = empty, restart = 100, max.iter = 100, score = 'bic')
plot(dag_run1, main = 'Run 1')
```



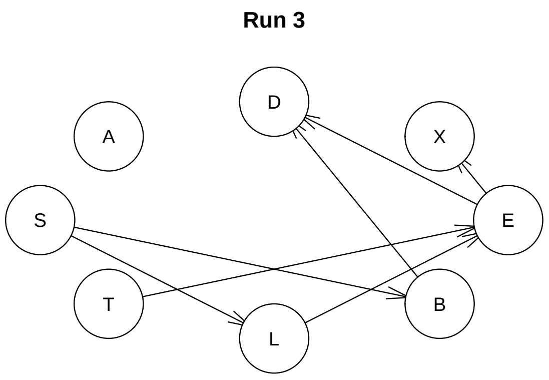
```
dag_run2 <-  hc(asia, start = empty, restart = 50, max.iter = 200, score = 'bde', iss = 2)
plot(dag_run2, main = 'Run 2')
```



```
dag_run3 <- hc(asia, start = empty, restart = 50, max.iter = 50, score = 'bde', iss = 1)
plot(dag_run3, main = 'Run 3')
```



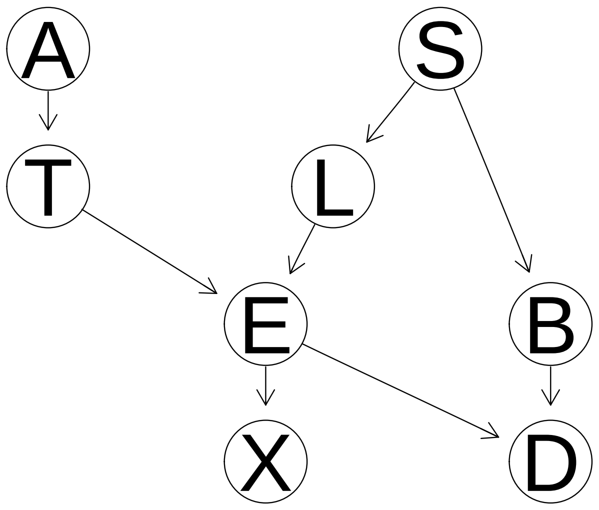
```
dag_run4 <- hc(asia)
plot(dag_run4, main = 'Run 3')
```

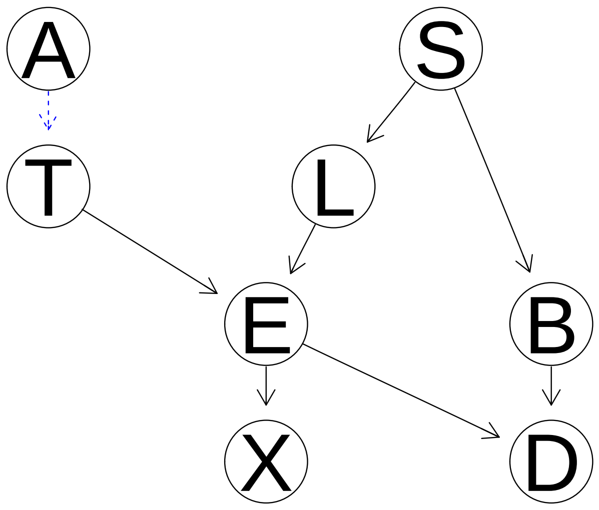


```
all.equal(dag_run3, dag_run4)
```

```
## [1] TRUE
```

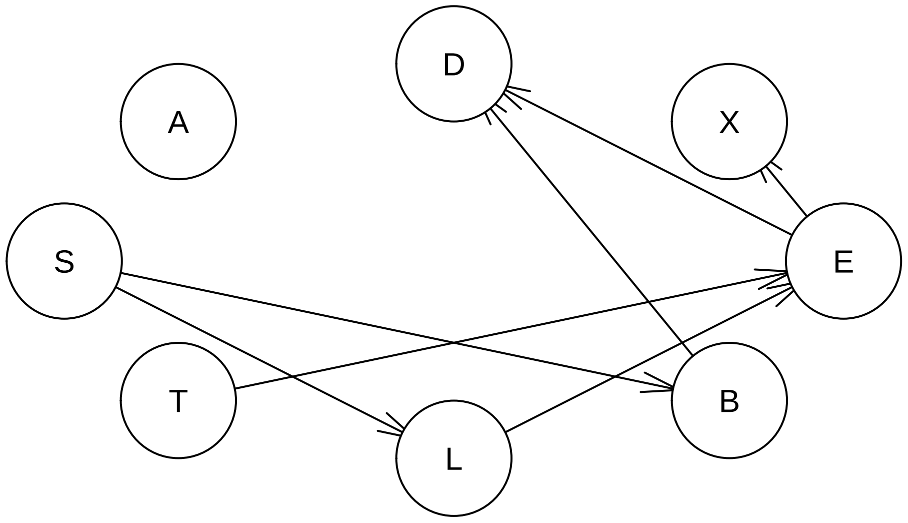
```
graphviz.compare(dag_run2, dag_run3)
```





Question 2

```
train_data <- asia[sample(dim(asia)[1], round(0.8*dim(asia)[1])),]
test_data <- asia[-as.numeric(row.names(train_data)),]

learn_dag <- hc(train_data)
dag_true <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
plot(learn_dag)
```



```r
fitted_dag <- bn.fit(x = learn_dag, data = train_data)
fitted_dag_true <- bn.fit(x = dag_true, data = train_data)

bayesNet_infer <- function(bn_fit, test_data, target_node,
                           markov_blanket = FALSE,
                           exact_inference = TRUE){
  # bn_fit = structure and parameter learned BN
  # Compiling the fitted Bayesian Network as a grain object
  bn_compile <- compile(as.grain(bn_fit))

  if (markov_blanket == FALSE) {
    # Evidence nodes are selected without markov blanket
    obs_node <- setdiff(colnames(test_data), target_node)
    obs_data <- as.data.frame(test_data[,obs_node])
  }else{
    # Evidence Nodes are selected based on markov-blanket of target node
    obs_node <- mb(x = fitted_dag, node = target_node)
    obs_data <- as.data.frame(test_data[,obs_node])
  }

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

  # We perform exact inference using setEvidence(), querygrain() here
  if (exact_inference == TRUE) {
    for (i in 1:nrow(obs_data)) {
      # Set evidence data of evidence nodes.
      set_obs_data <- setEvidence(object = bn_compile,
                                  nodes = obs_node, states = t(obs_data[i,]))
      #Querying the network for
      query_result <- querygrain(object = set_obs_data,
                                 nodes = target_node, type = 'marginal')

      predicted_S[i] <- ifelse(query_result$S['no'] > query_result$S['yes'],
                               'no', 'yes')
    }

  }else{
    #We perform approximate inference using cpquery() here.
    for (j in 1:nrow(obs_data)) {

      # Perform the conditional probability query of the event given evidence
      cp_query <- cpquery(fitted = fitted_dag, event = (S == 'yes'),
                          evidence = as.list(obs_data[j,]), method = "lw")
      # cp_dist <- cpdist(fitted_dag, nodes = 'S', evidence = as.list(obs_data[j, ]))
      # Store the result in the results vector
      # cp_results[j] <- mean(cp_dist['S'] == 'yes')
      predicted_S[j] <- ifelse(cp_query > 0.5, 'yes', 'no')
    }
  }

  return(predicted_S)

}
pred_exactInf_trueData <- bayesNet_infer(bn_fit = fitted_dag_true,
                                         test_data = test_data,
                                         target_node = 'S',markov_blanket = FALSE,
                                         exact_inference = TRUE)

pred_exactInf <- bayesNet_infer(bn_fit = fitted_dag, test_data = test_data,
                                target_node = 'S',markov_blanket = FALSE,
                                exact_inference = TRUE)

pred_approxInf <- bayesNet_infer(bn_fit = fitted_dag, test_data = test_data,
                                 target_node = 'S',markov_blanket = FALSE,
                                 exact_inference = FALSE)

pred_markovBlanket_exactInf <- bayesNet_infer(bn_fit = fitted_dag,
                                              test_data = test_data,
                                              target_node = 'S',
                                              markov_blanket = TRUE,
                                              exact_inference = TRUE)

pred_markovBlanket_apprInf <- bayesNet_infer(bn_fit = fitted_dag,
                                             test_data = test_data,
                                             target_node = 'S',
                                             markov_blanket = TRUE,
                                             exact_inference = FALSE)

# Function that computes the confusion matrix and accuracy
```

```r
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))
}
cfmat_exactInfer_true <- confusion_matrix(pred = pred_exactInf_trueData, true = test_data)
cfmat_exactInfer <- confusion_matrix(pred = pred_exactInf, true = test_data)
cfmat_approxInfer <- confusion_matrix(pred = pred_approxInf, true = test_data)
```

Below, we can see the comparison of Confusion Matrix and Accuracy of exact inference and approximate inference.

```r
print('The confusion matrix and the resulting accuracy for exact inference is as follows:')
```

```
## [1] "The confusion matrix and the resulting accuracy for exact inference is as follows:"
```

```r
cfmat_exactInfer
```

```
## $Confusion_Matrix
##      pred
##        no yes
##   no  343 150
##   yes 130 377
##
## $Accuracy
## [1] 72
```

```r
print('The confusion matrix and the resulting accuracy for approximate inference is as follows:')
```

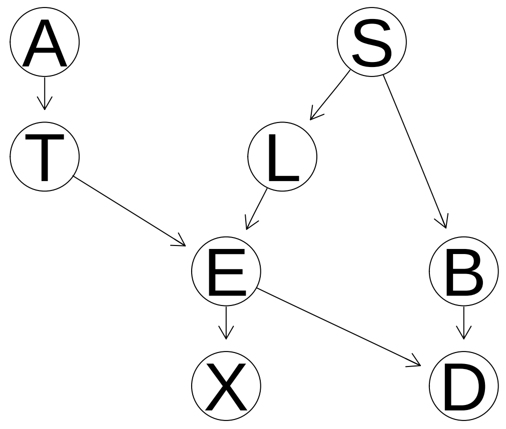
```
## [1] "The confusion matrix and the resulting accuracy for approximate inference is as follows:"
```

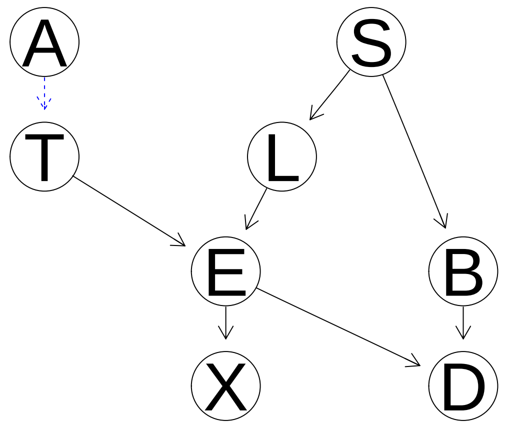
```r
cfmat_approxInfer
```

```
## $Confusion_Matrix
##      pred
##        no yes
##   no  343 150
##   yes 130 377
##
## $Accuracy
## [1] 72
```

Below is the BN comparison between True Asia BN and our BN.

```r
graphviz.compare(dag_true, learn_dag)
```





Question 3

Below is the reported Confusion Matrix and Accuracy of BN used to classify only the Markov Blanket of node S using both exact and approximate inference.

```
print('The confusion matrix and the resulting accuracy for exact inference using markov blanket is as fo
```

```
## [1] "The confusion matrix and the resulting accuracy for exact inference using markov blanket is as f
```

```
confusion_matrix(pred = pred_markovBlanket_exactInf, true = test_data)
```

```
## $Confusion_Matrix
##      pred
##         no yes
##   no  343 150
##   yes 130 377
##
## $Accuracy
## [1] 72
```

```
print('The confusion matrix and the resulting accuracy for approximate inference using markov blanket is
```

```
## [1] "The confusion matrix and the resulting accuracy for approximate inference using markov blanket i
```

```
confusion_matrix(pred = pred_markovBlanket_apprInf, true = test_data)
```

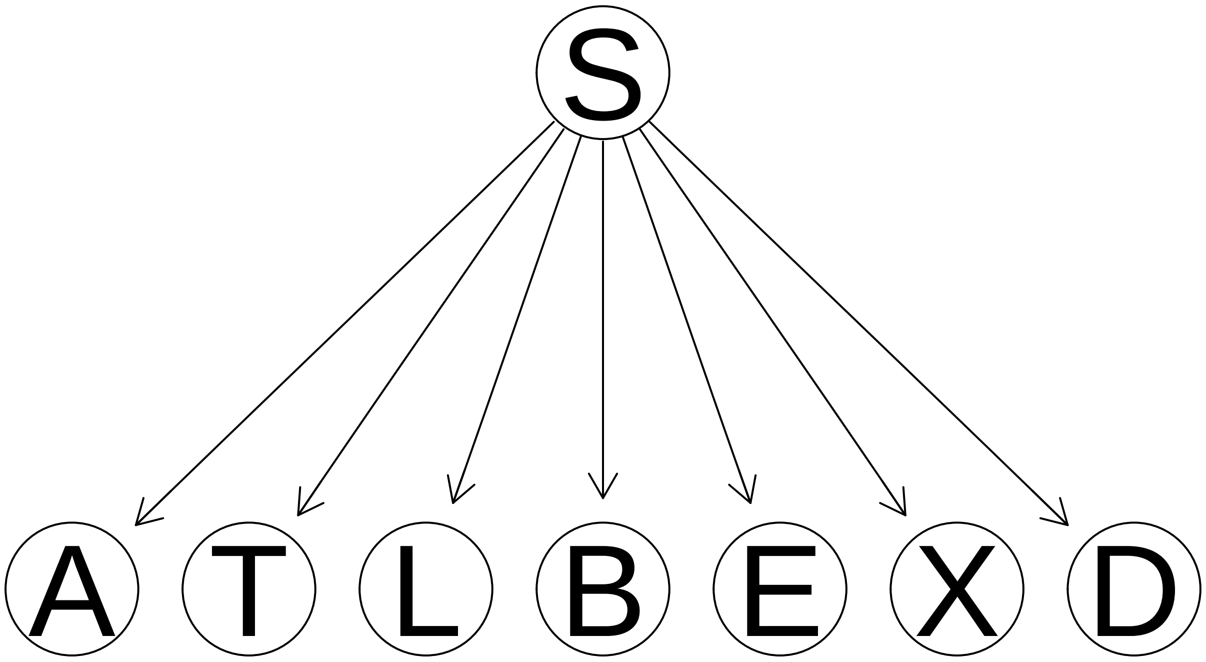
```
## $Confusion_Matrix
##      pred
##         no yes
##   no  343 150
##   yes 130 377
##
## $Accuracy
## [1] 72
```

Question 4

```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))
}

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


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

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

true_pred <- predict(true_naiveBayes, test_data)

true_naiveBayes_cfmat <- confusion_matrix(true_pred, test_data)
```

Below is the BN of the Naive-Bayes classifier created by hand.

```
graphviz.plot(naiveBayes_result[[2]])
```



Finally, we can see the comparison of the results of my implementation with respect to the in-built naive-bayes function.

```
print('The resulting confusion matrix and accuracy of the implemented Naive-Bayes Classifier is as foll
```

```
## [1] "The resulting confusion matrix and accuracy of the implemented Naive-Bayes Classifier is as fol
```

```
naiveBaye_cfmat
```

```
## $Confusion_Matrix
##      pred
##         no yes
##   no  365 128
##   yes 186 321
##
## $Accuracy
## [1] 68.6
```

```
print('The resulting confusion matrix and accuracy of the in-built Naive-Bayes function is as follows:')
```

```
## [1] "The resulting confusion matrix and accuracy of the in-built Naive-Bayes function is as follows:
```

```
true_naiveBayes_cfmat
```

```
## $Confusion_Matrix
##      pred
##        no yes
##   no  365 128
##   yes 187 320
##
## $Accuracy
## [1] 68.5
```

Question 5

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 it is most likely that they affect the posterior probability distribution of S due to conditional independence property.

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.