Advanced Machine Learning Lab 1

Simge Cinar

2024-09-08

Question 1

Show that multiple runs of the hill-climbing algorithm can return non-equivalent Bayesian network (BN) structures. Explain why this happens. Use the Asia dataset which is in- cluded in the bnlearn package. To load the data, run data("asia"). Recall from the lectures that the concept of non-equivalent BN structures has a precise meaning.

Answer:

```
# Get the dataset
data("asia")
head(asia)
```

```
##    A   S   T  L   B   E   X   D
## 1 no yes  no no yes  no  no yes
## 2 no yes  no no  no  no  no  no
## 3 no  no yes no  no yes yes yes
## 4 no  no  no no yes  no  no yes
## 5 no  no  no no  no  no  no yes
## 6 no yes  no no  no  no  no yes
```

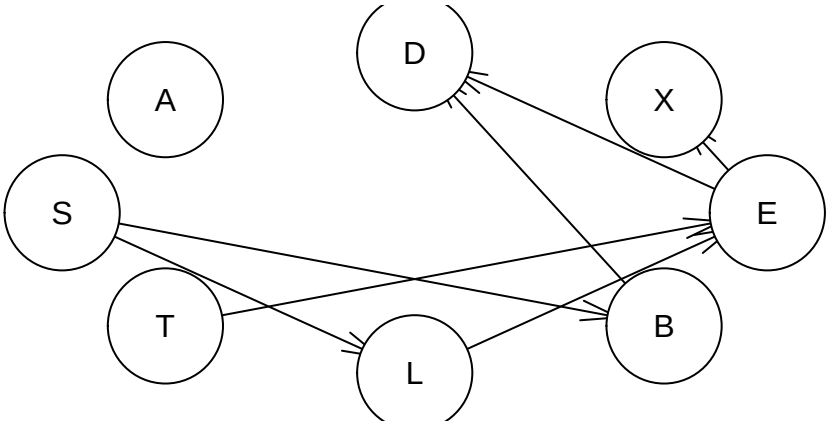
```
set.seed(12345)

# Run the hill-climbing algorithm three times
bn1 <- hc(asia) # graph 1
bn2 <- hc(asia, restart = 100) # graph 2

# Generate a random graph to change initialization
init_bn <- empty.graph(names(asia))
arcs(init_bn) <- matrix(c("A", "D", "T", "L"), ncol = 2, byrow = TRUE)
bn3 <- hc(asia, start = init_bn) # graph 3
```

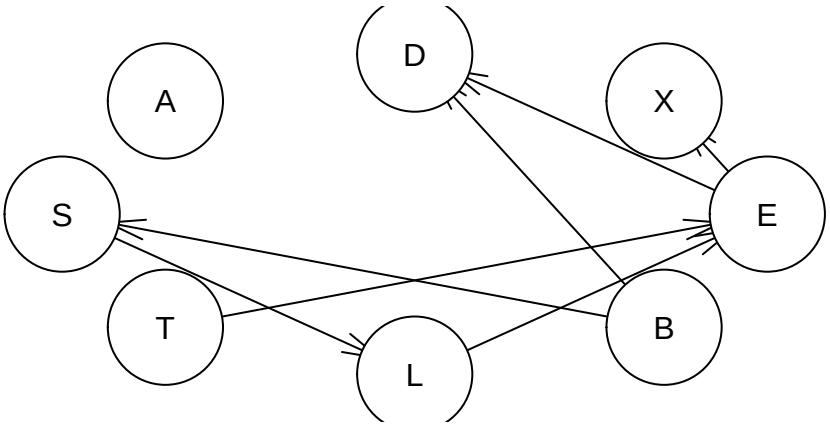
```
plot(bn1, main = "Bayesian Network 1")
```

Bayesian Network 1



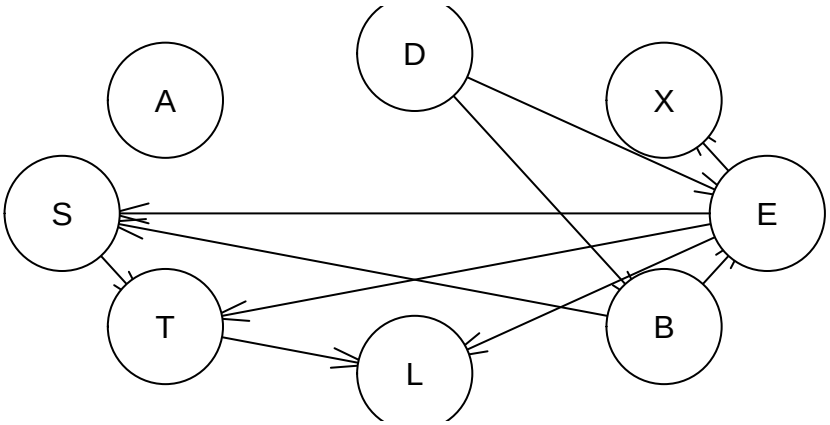

```
plot(bn2, main = "Bayesian Network 2")
```

Bayesian Network 2




```
plot(bn3, main = "Bayesian Network 3")
```

Bayesian Network 3



```r
# Check if they are equal
all.equal(bn1, bn2)
```

```
## [1] "Different arc sets"
```

```r
all.equal(bn1, bn3)
```

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

```r
all.equal(bn2, bn3)
```

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

```r
# Calculate the BDeu score, higher score is better
score_bn1 <- score(bn1, data = asia, type = "bde")
score_bn2 <- score(bn2, data = asia, type = "bde")
score_bn3 <- score(bn3, data = asia, type = "bde")

cat("BDeu score for bn1:", score_bn1, "\n")
```

```
## BDeu score for bn1: -11095.79
```

```r
cat("BDeu score for bn2:", score_bn2, "\n")
```

```
## BDeu score for bn2: -11095.79
```

```r
cat("BDeu score for bn3:", score_bn3, "\n")
```

```
## BDeu score for bn3: -11109.76
```

```r
modelstring(bn1) # Displays the selected model
```

```
## [1] "[A][S][T][L|S][B|S][E|T:L][X|E][D|B:E]"
```

I get 3 different bayesian network with different parameters in hill-climbing algorithm. Hill climbing algorithm finds the local maxima and doesn't guarantee optimality, it can stuck in local maximum hence different initilization yields different results.

BN 1 and BN 2 are equivalent, the graph is almost same except for the direction of one arc but direction of the arc is not important in the Markov equivalence. Also it should be not that their BDeu scores are the same. BN1 & BN2 and BN1 & BN3 are not equivalent.

Question 2

Learn a BN from 80 % of the Asia dataset. The dataset is included in the bnlearn package. To load the data, run data("asia"). Learn both the structure and the parameters. Use any learning algorithm and settings that you consider appropriate. Use the BN learned to classify the remaining 20 % of the Asia dataset in two classes: S = yes and S = no. In other words, compute the posterior probability distribution of S for each case and classify it in the most likely class. To do so, you have to use exact or approximate inference with the help of the bnlearn and gRain packages, i.e. you are not allowed to use functions such as predict. Report the confusion matrix, i.e. true/false positives/negatives. Compare your results with those of the true Asia BN, which can be obtained by running dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

Answer:

```r
set.seed(6)

# Split the data
train_idx <- sample(1:nrow(asia), size = 0.8 * nrow(asia))
train_asia <- asia[train_idx, ]
```

```r
test_asia <- asia[-train_idx, ]

cat("Number of rows in training data:", nrow(train_asia), "\n")
```

```
## Number of rows in training data: 4000
```

```r
cat("Number of rows in test data:", nrow(test_asia), "\n")
```

```
## Number of rows in test data: 1000
```

NOTE THAT: Conditional probability tables in grain objects must be completely specified; on the other hand, bn.fit allows NaN values for unobserved parents' configurations. Such bn.fit objects will be converted to m grain objects by replacing the missing conditional probability distributions with uniform distributions. Another solution to this problem is to fit another bn.fit with method = "bayes" and a low iss value, using the same data and network structure. (?as.grain)

```r
# Learn the parameters of the selected BN
fitted_bn1 <- bn.fit(bn1, data = train_asia, method = 'bayes')
fitted_bn1 # Shows the parameters of the model
```

```
##
##   Bayesian network parameters
##
##   Parameters of node A (multinomial distribution)
##
## Conditional probability table:
##           no         yes
## 0.990377406 0.009622594
##
##   Parameters of node S (multinomial distribution)
##
## Conditional probability table:
##         no       yes
## 0.4965009 0.5034991
##
##   Parameters of node T (multinomial distribution)
##
## Conditional probability table:
##           no         yes
## 0.990377406 0.009622594
##
##   Parameters of node L (multinomial distribution)
##
## Conditional probability table:
##
##      S
## L             no        yes
##   no  0.98376542 0.87875403
##   yes 0.01623458 0.12124597
##
##   Parameters of node B (multinomial distribution)
##
## Conditional probability table:
##
##      S
## B            no       yes
```

```
##   no  0.7063932 0.2805907
##   yes 0.2936068 0.7194093
##
##   Parameters of node E (multinomial distribution)
##
## Conditional probability table:
##
## , , L = no
##
##      T
## E               no          yes
##   no  9.999661e-01 3.546099e-03
##   yes 3.388223e-05 9.964539e-01
##
## , , L = yes
##
##      T
## E               no          yes
##   no  4.574565e-04 3.846154e-02
##   yes 9.995425e-01 9.615385e-01
##
##
##   Parameters of node X (multinomial distribution)
##
## Conditional probability table:
##
##      E
## X              no         yes
##   no  0.956565930 0.004012841
##   yes 0.043434070 0.995987159
##
##   Parameters of node D (multinomial distribution)
##
## Conditional probability table:
##
## , , E = no
##
##      B
## D             no        yes
##   no  0.90292643 0.21424698
##   yes 0.09707357 0.78575302
##
## , , E = yes
##
##      B
## D             no        yes
##   no  0.26876268 0.16002656
##   yes 0.73123732 0.83997344
```

```
grain_bn <- as.grain(fitted_bn1) # convert the network to grain object
compiled_grain_bn <- compile(grain_bn) # construct the junction tree
```

```
preds1 <- vector("character", length = nrow(test_asia))

for (i in 1:nrow(test_asia)){
```

```
  evidence_test <- test_asia[i, -2]

  # Conditioning the network on specific values, all variables expect for S
  compiled_grain_bn_evidence <- setEvidence(compiled_grain_bn,
                                            nodes = names(evidence_test),
                                            states = as.vector(unlist(evidence_test)))

  # Get the probability of each state of node S by taking into account the evidence set in the network
  query_result1 <- querygrain(compiled_grain_bn_evidence, nodes = "S", type = "marginal")$S

  if (query_result1['yes'] > query_result1['no']){
    preds1[i] <- "yes"
  }
  else{
    preds1[i] <- "no"
  }
}

cat("Confusion matrix:\n")
```

```
## Confusion matrix:
```

```
confusion_matrix1 <- table(preds1, test_asia$S)
confusion_matrix1
```

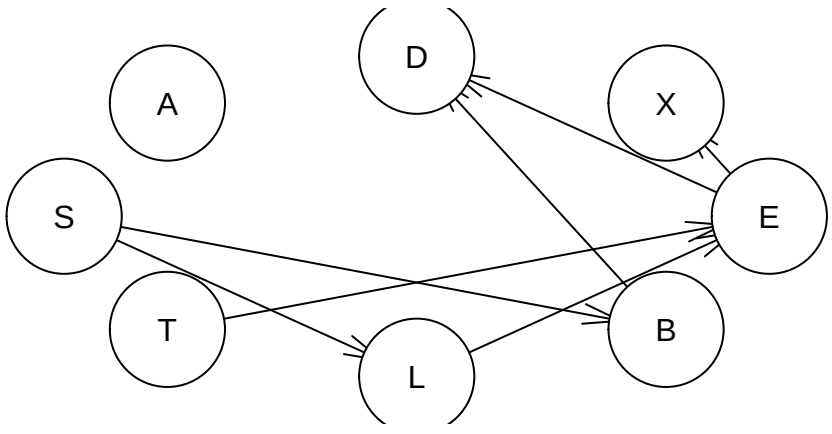
```
##
## preds1  no yes
##    no  337 133
##    yes 162 368
```

```
accuracy1 <- sum(diag(confusion_matrix1)) / sum(confusion_matrix1)
cat("Accuracy:", accuracy1)
```

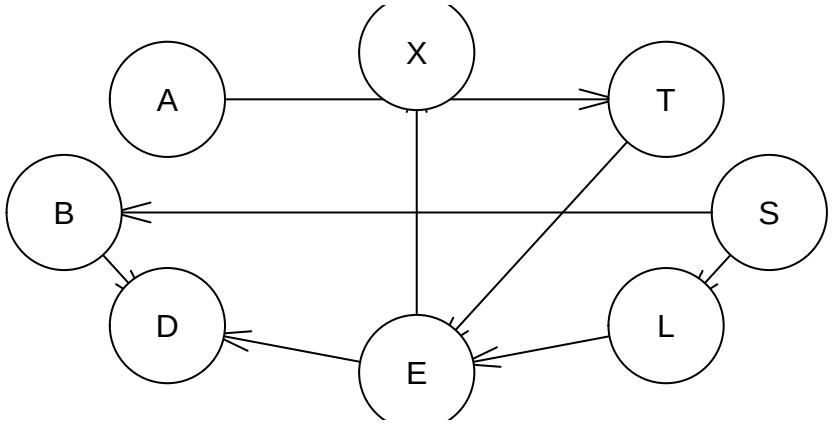
```
## Accuracy: 0.705
```

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

plot(bn1, main = "Fitted BN")
```



```r
plot(dag, main = "True BN") # The best
```




```r
# Run the algorithm on the optimal tree
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

fitted_bn_best <- bn.fit(dag, data = train_asia, method = 'bayes')
grain_bn_best <- as.grain(fitted_bn_best) # convert the network to grain object
compiled_grain_best <- compile(grain_bn_best) # construct the junction tree

preds_best_bn <- vector("character", length = nrow(test_asia))

for (i in 1:nrow(test_asia)){
  evidence_test <- test_asia[i, -2]

  # Conditioning the network on specific values, all variables expect for S
  compiled_grain_best_evidence <- setEvidence(compiled_grain_best,
                                              nodes = names(evidence_test),
                                              states = as.vector(unlist(evidence_test)))

  # Get the probability of each state of node S by taking into account the evidence set in the network
  query_result_best <- querygrain(compiled_grain_best_evidence, nodes = "S", type = "marginal")$S

  if (query_result_best['yes'] > query_result_best['no']){
    preds_best_bn[i] <- "yes"
  }
  else{
    preds_best_bn[i] <- "no"
  }
}

cat("Confusion matrix:\n")
```

```
## Confusion matrix:
```

```r
confusion_matrix_best_bn <- table(preds_best_bn, test_asia$S)
confusion_matrix_best_bn
```

```
##
```

```
## preds_best_bn  no yes
##           no  337 133
##           yes 162 368
```

```
accuracy_best_bn <- sum(diag(confusion_matrix_best_bn)) / sum(confusion_matrix_best_bn)
cat("Accuracy:", accuracy_best_bn)
```

```
## Accuracy: 0.705
```

The BN I fitted and optimal BN gives the same confusion matrix.

Question 3

In the previous exercise, you classified the variable S given observations for all the rest of the variables. Now, you are asked to classify S given observations only for the so-called Markov blanket of S, i.e. its parents plus its children plus the parents of its children minus S itself. Report again the confusion matrix.

Answer:

```
markov_blanket_S <- mb(fitted_bn1, "S") # get markov blanket of S
cat("Markov blanket of S:", markov_blanket_S)
```

```
## Markov blanket of S: L B
```

```
test_asia_mb <- test_asia[, markov_blanket_S]
preds_mb <- vector("character", length = nrow(test_asia))

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

  # Conditioning the network on specific values, all variables expect for S
  compiled_grain_evidence_mb <- setEvidence(compiled_grain_bn,
                                            nodes = names(test_asia_mb),
                                            states = as.character(unlist(test_asia_mb[i, ])))

  # Gives the probability of each state of node S, taking into account the evidence set in the network
  query_result_mb <- querygrain(compiled_grain_evidence_mb, nodes = "S", type = "marginal")$S

  if (query_result_mb['yes'] > query_result_mb['no']){
    preds_mb[i] <- "yes"
  }
  else{
    preds_mb[i] <- "no"
  }
}

cat("Confusion matrix:\n")
```

```
## Confusion matrix:
```

```
confusion_matrix_mb <- table(preds_mb, test_asia$S)
confusion_matrix_mb
```

```
##
## preds_mb  no yes
##      no  337 133
##      yes 162 368
```

```
accuracy_mb <- sum(diag(confusion_matrix_mb)) / sum(confusion_matrix_mb)
cat("Accuracy:", accuracy_mb)
```

```
## Accuracy: 0.705
```

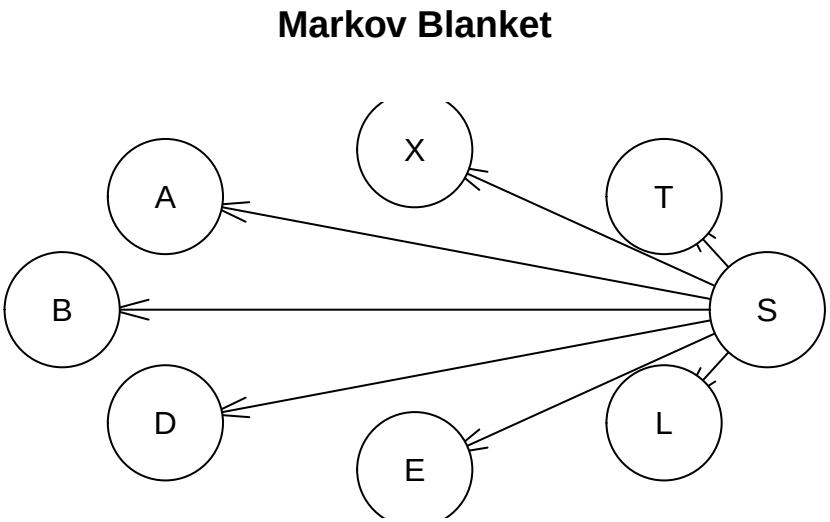
Question 4:

Repeat the exercise (2) using a naive Bayes classifier, i.e. the predictive variables are independent given the class variable. See p. 380 in Bishop's book or Wikipedia for more information on the naive Bayes classifier. Model the naive Bayes classifier as a BN. You have to create the BN by hand, i.e. you are not allowed to use the function naive.bayes from the bnlearn package.

Answer:

It assumes that predictors in a Naïve Bayes model are conditionally independent, or unrelated to any of the other feature in the model.

```
dag_naive_bayes <- model2network("[A|S][T|S][L|S][B|S][E|S][X|S][D|S][S]")
plot(dag_naive_bayes, main = "Markov Blanket")
```




```
fitted_naive_bayes <- bn.fit(dag_naive_bayes, data = train_asia, method = 'bayes')
fitted_naive_bayes # Shows the parameters of the model
```

```
##
##   Bayesian network parameters
##
##   Parameters of node A (multinomial distribution)
##
## Conditional probability table:
##
##      S
## A              no         yes
##   no  0.992323181 0.988458675
##   yes 0.007676819 0.011541325
##
##   Parameters of node B (multinomial distribution)
##
## Conditional probability table:
##
##      S
## B            no       yes
```

```
##   no  0.7063932 0.2805907
##   yes 0.2936068 0.7194093
##
##   Parameters of node D (multinomial distribution)
##
## Conditional probability table:
##
##      S
## D            no       yes
##   no  0.6872640 0.3768925
##   yes 0.3127360 0.6231075
##
##   Parameters of node E (multinomial distribution)
##
## Conditional probability table:
##
##      S
## E            no        yes
##   no  0.97420086 0.87081162
##   yes 0.02579914 0.12918838
##
##   Parameters of node L (multinomial distribution)
##
## Conditional probability table:
##
##      S
## L            no        yes
##   no  0.98376542 0.87875403
##   yes 0.01623458 0.12124597
##
##   Parameters of node S (multinomial distribution)
##
## Conditional probability table:
##         no       yes
## 0.4965009 0.5034991
##
##   Parameters of node T (multinomial distribution)
##
## Conditional probability table:
##
##      S
## T            no        yes
##   no  0.98980619 0.99094068
##   yes 0.01019381 0.00905932
##
##   Parameters of node X (multinomial distribution)
##
## Conditional probability table:
##
##      S
## X            no        yes
##   no  0.93342562 0.83209233
##   yes 0.06657438 0.16790767
```

```
grain_naive_bayes <- as.grain(fitted_naive_bayes) # convert the network to grain object
compiled_grain_naive_bayes <- compile(grain_naive_bayes) #construct the junction tree
```

```
preds_naive_bayes <- vector("character", length = nrow(test_asia))

for (i in 1:nrow(test_asia)){
  evidence_test <- test_asia[i, -2]

  # Conditioning the network on specific values, all variables expect for S
  compiled_grain_evidence_nb <- setEvidence(compiled_grain_naive_bayes,
                                            nodes = names(evidence_test),
                                            states = as.vector(unlist(evidence_test)))

  # Gives the probability of each state of node S, taking into account the evidence set in the network
  query_result_nb <- querygrain(compiled_grain_evidence_nb, nodes = "S", type = "marginal")$S

  if (query_result_nb['yes'] > query_result_nb['no']){
    preds_naive_bayes[i] <- "yes"
  }
  else{
    preds_naive_bayes[i] <- "no"
  }
}

cat("Confusion matrix:\n")
```

```
## Confusion matrix:
```

```
confusion_matrix_naive_bayes <- table(preds_naive_bayes, test_asia$S)
confusion_matrix_naive_bayes
```

```
##
## preds_naive_bayes  no yes
##               no  374 190
##               yes 125 311
```

```
accuracy_naive_bayes <- sum(diag(confusion_matrix_naive_bayes)) / sum(confusion_matrix_naive_bayes)
cat("Accuracy:", accuracy_naive_bayes)
```

```
## Accuracy: 0.685
```

Question 5:

Explain why you obtain the same or different results in the exercises (2-4)

Answer: Part 2 and part 3 give the same result because markov blanket take into account only the close neighborhood. However, if the graphs was more complex, the results might be different. Naive Bayes classifier give lower accuracy compared to part 2 and part 3 because it assumes that any of the features are independent of each other which is not the case.