gustav_wahlquist

Gustav Wahlquist

2020-09-11

Libraries used:

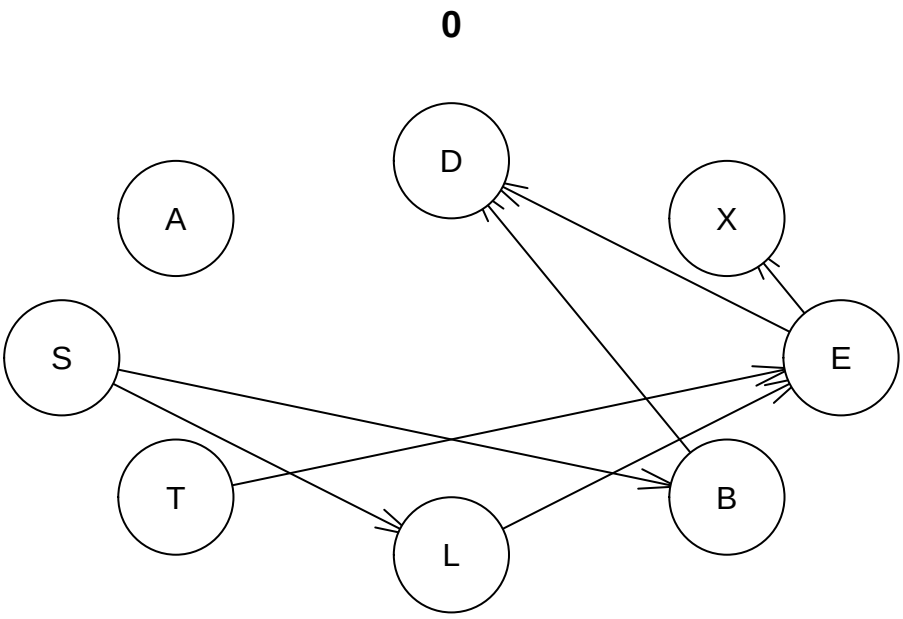
```
library(bnlearn)
library(gRain)
```

Question 1

As shown in the the following plots, multiple runs of the hill-climbing algorithm can lead to non-equivalent Bayesian networks. Running the algorithm for the third time on the network changes direction on the edge between node "S" and "B". This is due to the fact that in the hill-climbing algorithm can get stuck in local optimums and different runs can therefore end up in different solutions depending on starting point.

```
data("asia")
set.seed(1)

# Initial run
model = hc(asia, restart = 1)
plot(model, main= 0)
```

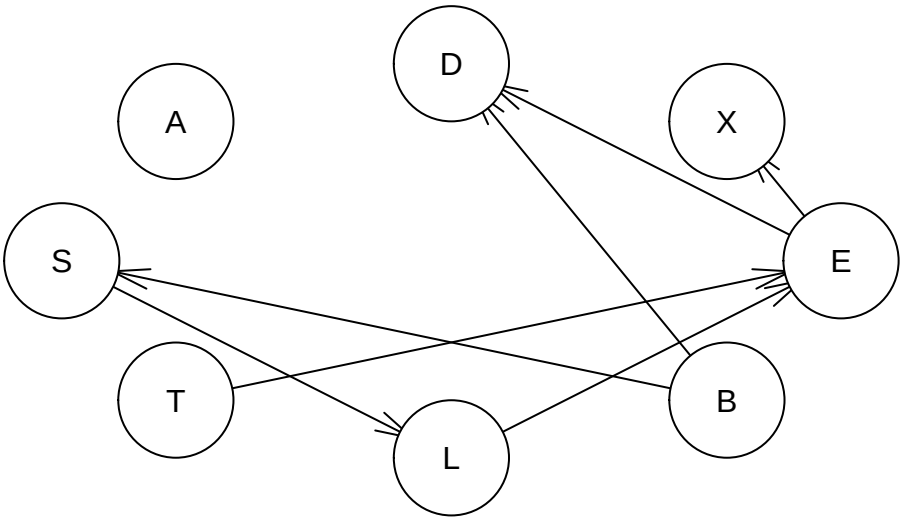



```
# Runs the hc-algorithm 'i' times and if the model changes,
# the network gets plotted.
for (i in 1:10) {
  updated_model = hc(asia, start = model, restart = 1)


  if (all.equal(updated_model, model) != TRUE) {
    plot(updated_model, main = i)
  }

  model = updated_model
}
```

3



Question 2

```
# Structuring data
train_set = asia[1:4000,]
test_set = asia[4001:5000,]
test_labels = test_set["S"]
test_set = subset(test_set, select = -c(S) )
test_columns = colnames(test_set)

bn_structure_learning <- function(dataset, restarts) {

  # Structure learning
  model_structure = hc(dataset, restart = restarts)
  #plot(model_structure)

  return(model_structure)
}

bn_parameter_learning <- function(bn_structure, training_set) {

  # Parameter learning
  model = bn.fit(bn_structure, training_set)

  # Converting model to grain objects (avoiding NaN-values?)
```

```
  grained_model = as.grain(model)

  # Compiling model to an BN?
  compiled_model = compile(grained_model)

  return(compiled_model)
}

bn_precdict <- function(bn_model, testing_set, columns) {

  predictions = c()

  # Predictions
  for (i in 1:dim(testing_set)[1]) {

    # Collecting ith obs states in a vector
    states = NULL
    for (j in columns) {
      if (testing_set[i,j] == "yes") {
        states = c(states, "yes")
      }  else {
        states = c(states, "no")
      }
    }

    # Prediction of var "S" on the ith obs
    evidence = setEvidence(bn_model, nodes = columns, states = states)

    probabilities = querygrain(evidence, c("S"))

    if(probabilities$S["no"] >= 0.5) {
      predictions[i] = "no"
    } else {
      predictions[i] = "yes"
    }
  }
  return(predictions)
}

# Learning a BN from dataset
bn_structure = bn_structure_learning(asia, 5)
bn = bn_parameter_learning(bn_structure, train_set)
preds = bn_precdict(bn, test_set, test_columns)

# Plotting result
confusion_matrix = table(preds, test_labels$S, dnn=c("Predictions", "True labels"))
#plot(confusion_matrix)
confusion_matrix
```

```
##            True labels
## Predictions  no yes
##         no  358 120
##         yes 147 375
```

```
# True BN network for asia dataset
true_asia_bn_struct = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
true_asia_bn = bn_parameter_learning(true_asia_bn_struct, train_set)
preds_asia = bn_precdict(true_asia_bn, test_set, test_columns)

# Plotting result, true BN
confusion_matrix_true_bn = table(preds_asia, test_labels$S, dnn=c("Predictions", "True labels"))
#plot(confusion_matrix_true_bn)
confusion_matrix_true_bn
```

```
##             True labels
## Predictions  no yes
##         no  358 120
##         yes 147 375
```

As seen by looking at the two confusion matrices (the first one for the network with learned structure and the second one for the given true-structure network), there is no difference in the result between the two different networks.

Question 3

```
# Learning a BN from dataset
bn_structure = bn_structure_learning(asia, 5)
bn = bn_parameter_learning(bn_structure, train_set)
# Here, only the nodes in the markov blanket of "S" are to be used when predicting "S"
preds = bn_precdict(bn, test_set, mb(bn_structure, c("S")))

# Plotting result
confusion_matrix_mb = table(preds, test_labels$S, dnn=c("Predictions", "True labels"))
#plot(confusion_matrix_mb)
confusion_matrix_mb
```

```
##             True labels
## Predictions  no yes
##         no  358 120
##         yes 147 375
```

Question 4

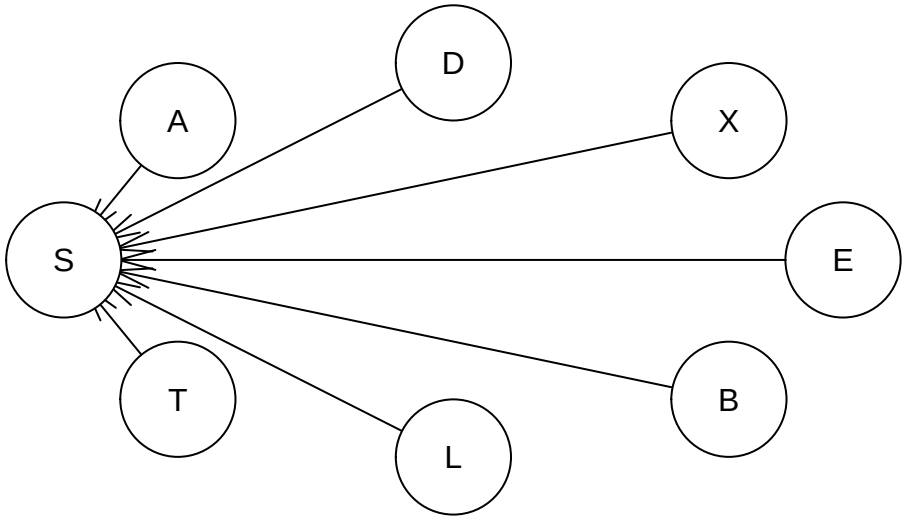
```
# This function is changed from using the HC-algorithm to compute the structure of the network to only
bn_structure_learning <- function() {

  model_structure = empty.graph(c("A", "S", "T", "L", "B", "E", "X", "D"))
  arc.set = matrix(c("A", "S", "T", "S", "L", "S", "B", "S", "E", "S", "X", "S", "D", "S"), ncol = 2,
                   byrow = TRUE, dimnames = list(NULL, c("from", "to")))
  arcs(model_structure) = arc.set
  #plot(bn_nodes)

  return(model_structure)
```

```
}

# Learning a BN from dataset
bn_naive_structure = bn_structure_learning()
plot(bn_naive_structure)
```




```
bn = bn_parameter_learning(bn_naive_structure, train_set)
preds = bn_precdict(bn, test_set, test_columns)

# Plotting result
confusion_matrix = table(preds, test_labels$S, dnn=c("Predictions", "True labels"))
#plot(confusion_matrix)
confusion_matrix
```

```
##            True labels
## Predictions  no yes
##         no  358 122
##         yes 147 373
```

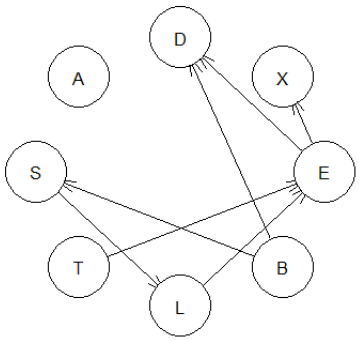
```
# True BN network for asia dataset
true_asia_bn_struct = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
true_asia_bn = bn_parameter_learning(true_asia_bn_struct, train_set)
preds_asia = bn_precdict(true_asia_bn, test_set, test_columns)
```

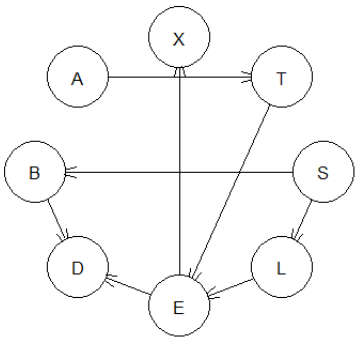
```r
# Plotting result, true BN
confusion_matrix_true_bn = table(preds_asia, test_labels$S, dnn=c("Predictions", "True labels"))
#plot(confusion_matrix_true_bn)
confusion_matrix_true_bn
```

```
##            True labels
## Predictions  no yes
##         no  358 120
##         yes 147 375
```

Question 5

The two networks in question 2 are quite differently structured but they both produces the same results since the edges connected to the "S"-node are the same in both networks and the data in the "asia"-set is complete. Since the data is complete, the prediction on "S" is independent from nodes that are not directly connected to "S" with an edge.





The result in the confusion matrix in question 3 is the same as the two in question 2. This is due to the same reason as mentioned above, the prediction on "S" is only dependent on the nodes directly connected to "S" and since the data is complete, it is enough to only look at the markov blanket of "S".

In question 4, the result is slightly worse than the previous predictions. This is due to the fact that when the network is structured to resemble a naive bayes classifier, the prediction/classification on the node "S" is done by taking into account the values of all other variables. This leads to worse performance since variables/nodes that are not directly connected to the "S"-node influences the prediction, even though the data that is directly connected is known.