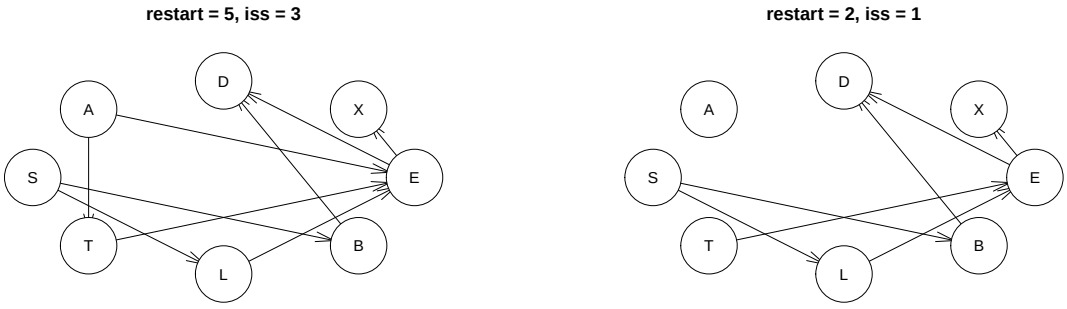
TDDE15 Lab 1

William Bergekrans

2021-09

Question 1

Show that multiple runs of the hill-climbing algorithm can return non-equivalent Bayesian network structures. The definition of equivalent DAGs is that they need to have the same adjacencies and unshielded colliders.



Looking at the two different networks we can see that they do not have the same adjacencies which they need to have in order for the two graphs to be equivalent. In the following tables all unshielded colliders are presented for the two graphs.

Table 1: restart = 5, iss = 3

X	Z	Y
A	E	L
T	E	L
B	D	E

Table 2: restart = 2, iss = 1

X	Z	Y
T	E	L
B	D	E

As we can see from the two tables the two graphs does not have the same unshielded colliders and therefore does not satisfy the equivalence criteria.

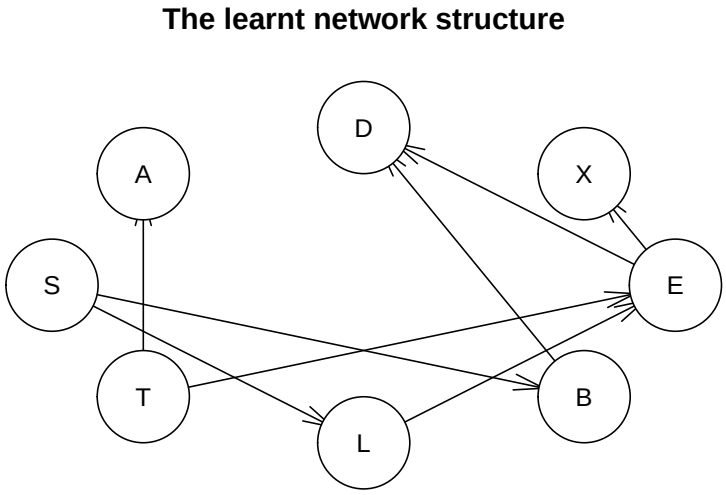
The reason different networks are returned is because the represent different local maximums (or a global maximum if we are lucky). If there are more restarts the algorithm should be more likely to find a better

solution. The iss score is a weight to the prior compared to the sample and impacts the smoothness of the posterior distribution. A higher iss value will lead to more arcs in the graph and will therefore likely increase the number of colliders in the graph, which could lead to non-equivalent graphs.

```
# Code for question 1
data("asia") # load the Asia data set.

# Get bn structure using hill-climbing with standard settings.
bn <- hc(asia, score = "bde", restart = 5, iss = 3)
plot(bn, main="restart = 5, iss = 3")

bn2 <- hc(asia, score = "bde", restart = 2, iss = 1)
plot(bn2, main="restart = 2, iss = 1")

# vstructs(bn)
# vstructs(bn2)
```

Question 2

First the original dataset is divided into a training (80%) and test (20%) set. Then the structure is learnt using the hill-climbing algorithm used in qustion 1. The BDeu-score is used together with 30 restarts and the iss set to 2. The resulting plot is:



The network is almost identical to the correct network found at https://www.bnlearn.com/documentation/networks/. The difference is that the edge between A and T have the wrong direction. However, this does not affect any colliders and or adjacencies and the two graphs are equivalent. As it is not the complete asian graph casual inference cannot be used. However we can still use probabilistic inference on the network.

Next, the parameters are learnt. For this purpose the function bn.learn() is used for exact inference.

The confusion matrix for predictions on the test set using the learnt Bayesian network is as follows:

Table 3: Confusion errors, trainied DAG structure

Predicted S	True S	Frequency
no	no	341
yes	no	155
no	yes	120
yes	yes	384

As the table shows it is quite common with wrong predictions, and the faults appear to be both false-positives and true-negatives. After creating the true dag structure and training the parameters using the same method as for the first structure, the following predictions on the test set are calculated:

Table 4: Confusion errors, true DAG structure

Predicted S	True S	Frequency
no	no	341
yes	no	155
no	yes	120
yes	yes	384

The tables are identical which in this case isn't entirely unexpected as the first DAG that was used is almost identical to the true DAG. The only difference between the two graphs is the link between A and T. The connection between A and T is very weak which probably explains why this fault in the graph doesn't affect the prediction results. If there was a larger test set there should be a small difference as the two models are different even if it doesn't show in this test.

```
# Code for question 2
# Divide the data into a training and test set.
set.seed(23457) # For reproducibility
n = dim(asia)[1]
id = sample(1:n, floor(n*0.8))
train = asia[id,]
test = asia[-id,]
rm(id, n)

# Learn the structure of the network.
bn <- hc(train, score = "bde", restart = 30, iss = 2)
plot(bn, main = "The learnt network structure")

# Learn the parameters
fitted <- bn.fit(bn, train, method = "bayes", iss = 2)
pred <- integer(dim(test)[1])
# For every test case, calculate the conditional probability for S given
# the other variables.
for (i in 1:dim(test)[1]) {
  # Conditional probability queries
  alpha <- cpquery(fitted, event = (S == "yes"),
    evidence = as.list(test[i, -2]), method = "lw")

  if (alpha > 0.5) {
    pred[i] <- "yes"
  } else {
```

```
    pred[i] <- "no"
  }
}

pred <- as.factor(pred)

confMatrix <- as.data.frame(table(pred, test$S))
colnames(confMatrix) <- c("Predicted S", "True S", "Frequency")

# True network
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
true_fit <- bn.fit(dag, train, method = "bayes", iss = 2)
true_pred <- integer(dim(test)[1])
# For every test case, calculate the conditional probability for S given
# the other variables.
for (i in 1:dim(test)[1]) {
  # Conditional probability queries
  alpha <- cpquery(true_fit, event = (S == "yes"),
    evidence = as.list(test[i, -2]), method = "lw")

  if (alpha > 0.5) {
    true_pred[i] <- "yes"
  } else {
    true_pred[i] <- "no"
  }
}

true_pred <- as.factor(true_pred)

true_confMatrix <- as.data.frame(table(true_pred, test$S))
colnames(true_confMatrix) <- c("Predicted S", "True S", "Frequency")
```

Question 3

Now S is classified using its Markov blanket. The confusion matrix for the test set using this method is:

Table 5: Confusion errors, Markov Blanket of S

Predicted S	True S	Frequency
no	no	341
yes	no	155
no	yes	120
yes	yes	384

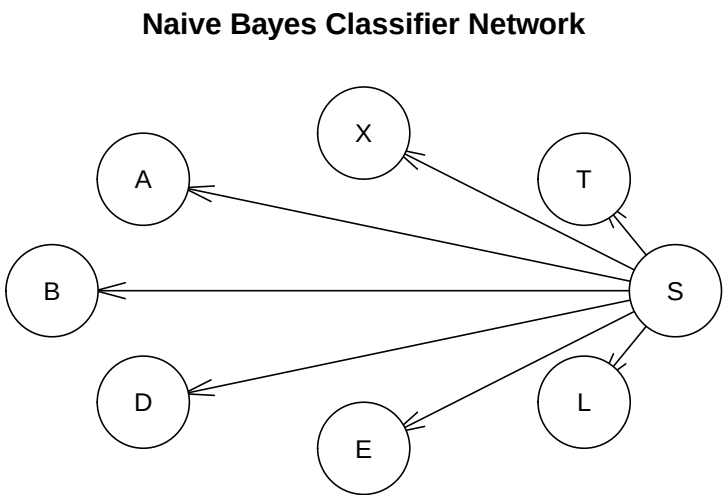
```
# Question 3
# Get the markov blanket for S
blanket <- mb(fitted, "S")
# mb is L and B.
# Use only L and B for predicting on

mb_pred <- integer(dim(test)[1])
# For every test case, calculate the conditional probability for S given
```

```r
# the other variables.
for (i in 1:dim(test)[1]) {
  # Conditional probability queries
  alpha <- cpquery(fitted, event = (S == "yes"),
    evidence = as.list(test[i, 4:5]), method = "lw")

  if (alpha > 0.5) {
    mb_pred[i] <- "yes"
  } else {
    mb_pred[i] <- "no"
  }
}

mb_pred <- as.factor(mb_pred)

mb_confMatrix <- as.data.frame(table(mb_pred, test$S))
colnames(mb_confMatrix) <- c("Predicted S", "True S", "Frequency")

kbl(mb_confMatrix, caption = "Confusion errors, Markov Blanket of S") %>%
  kable_styling(latex_options = "HOLD_position")
```

Question 4

Instead of using a normal Bayesian network a naive Bayes classifier shall be used. Similar to question 2, the model will be trained on 80% of the asia data set and tested on the remaining observations. Both the structure and parameters should be learnt. The network is created manually using the model2network function. The naive bayes classifier assumes that all features are independent each other given the target, which in this case is S.

The network for looks as follows:



The confusion matrix after predicting on the test set using the naive Bayesian classifier is:

Table 6: Confusion errors, naive Bayesian Classifier

Predicted S	True S	Frequency
no	no	365
yes	no	131
no	yes	173
yes	yes	331

Compared to the previous results (Q2-Q3) the confusion matrix is different. Interestingly it performs better for when S is "no" and worse when S is "yes". The naive bayes classifier had a total of 304 wrong predictions while the complete bayesian network and variable elimination resulted in 275 wrong predictions. There is a small difference and the simplicity of the naive classifier comes at a cost, it is hard to tell if it is worth it or not without a more concrete use case and information.

```
# Code for question 4
# Set the structure.
naive <- model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")

# Train the parameters
naive_fit <- bn.fit(naive, train, method = "bayes", iss=2)

# Predict on the test set
naive_pred <- integer(dim(test)[1])
# For every test case, calculate the conditional probability for S given
# the other variables.
for (i in 1:dim(test)[1]) {
  # Conditional probability queries
  alpha <- cpquery(naive_fit, event = (S == "yes"),
    evidence = as.list(test[i, -2]), method = "lw")

  if (alpha > 0.5) {
    naive_pred[i] <- "yes"
  } else {
    naive_pred[i] <- "no"
  }
}

naive_pred <- as.factor(naive_pred)

naive_confMatrix <- as.data.frame(table(naive_pred, test$S))
colnames(naive_confMatrix) <- c("Predicted S", "True S", "Frequency")
```

Question 5

As seen earlier the prediction results in question 2 and 3 where identical. This is because even though the first network was not the true one it had the same markov blanket as the true network. When we actually only predicted using the markov blanket the results did not differ either. This is because that in Bayesian network a variable is independent with the other given its markov blanket. So the first two predictions were actually using redundant information compared to when only the market blanket was used.

The network used in question 2 and 3 were complete bayesian networks for the given problem. In question 4 a naive bayes classifier was created which comes with a simplified network structure that doesn't capture the true dependecies. Because it is a simplified version it is expected to perform worse, which it did.