TDDE15 - Lab 1

Martin Friberg - marfr370

2021/09/10

Assignment 1

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

Two DAGs represent the same independencies according to the separation criterion (i.e. they are equivalent) if and only if they have the same adjacencies and unshielded colliders. The causal variables influencing the collider are themselves not necessarily associated. If they are not adjacent, the collider is unshielded.

```
library(bnlearn)
library(RBGL)
library(Rgraphviz)
library(gRain)
```

```
data("asia")
nodes = names(asia)
e = empty.graph(nodes)


hc_restart0 = hc(asia, restart=0, score="bde", iss=15)
hc_restart100 = hc(asia, restart=100, score="bde", iss=15)

information <- function(hc1, hc2){
  print(arcs(hc1))
  print(arcs(hc2))
  print('VSTRUCTS')
  #moral = false is the default value and it makes vstructs return unshielded colliders
  print(vstructs(hc1, moral=FALSE))
  print(vstructs(hc2, moral=FALSE))
  print('CPDAG:')
  cpdag(hc1)
  cpdag(hc2)
  all.equal(hc1,hc2)
}

information(hc_restart0, hc_restart100)
```

```
##       from to
##  [1,] "B"  "D"
##  [2,] "L"  "E"
##  [3,] "E"  "X"
##  [4,] "S"  "B"
##  [5,] "T"  "E"
```

```
##  [6,] "E"  "D"
##  [7,] "S"  "L"
##  [8,] "A"  "T"
##  [9,] "A"  "E"
## [10,] "L"  "X"
## [11,] "T"  "X"
## [12,] "T"  "L"
## [13,] "T"  "B"
## [14,] "L"  "D"
##       from to
##  [1,] "B"  "D"
##  [2,] "T"  "E"
##  [3,] "E"  "D"
##  [4,] "T"  "X"
##  [5,] "T"  "L"
##  [6,] "T"  "A"
##  [7,] "T"  "B"
##  [8,] "A"  "E"
##  [9,] "B"  "A"
## [10,] "L"  "E"
## [11,] "E"  "X"
## [12,] "S"  "L"
## [13,] "L"  "X"
## [14,] "S"  "B"
## [15,] "L"  "A"
## [16,] "L"  "D"
## [1] "VSTRUCTS"
##      X   Z   Y
## [1,] "S" "L" "T"
## [2,] "S" "B" "T"
## [3,] "A" "E" "L"
## [4,] "L" "D" "B"
## [5,] "B" "D" "E"
##      X   Z   Y
## [1,] "L" "A" "B"
## [2,] "S" "L" "T"
## [3,] "S" "B" "T"
## [4,] "L" "D" "B"
## [5,] "B" "D" "E"
## [1] "CPDAG:"
```

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

```
library(Rgraphviz)
```

Explain why this happens

Since the hill climbing algorithm uses a randomized search approach and is not asymptotically correct under faithfulness it may get stuck in a local maxima where the solution can not be approved upon by neighboring states since the cost function is higher in those. To attempt to avoid getting stuck in local optima, one could use restarts (i.e. repeated local search) which is what we define in the hc algorithm.

The variable iss is the user-defined imaginary sample size (the higher the less regularization).

Bayesian score favours models that trade off fit of data and model complexity.

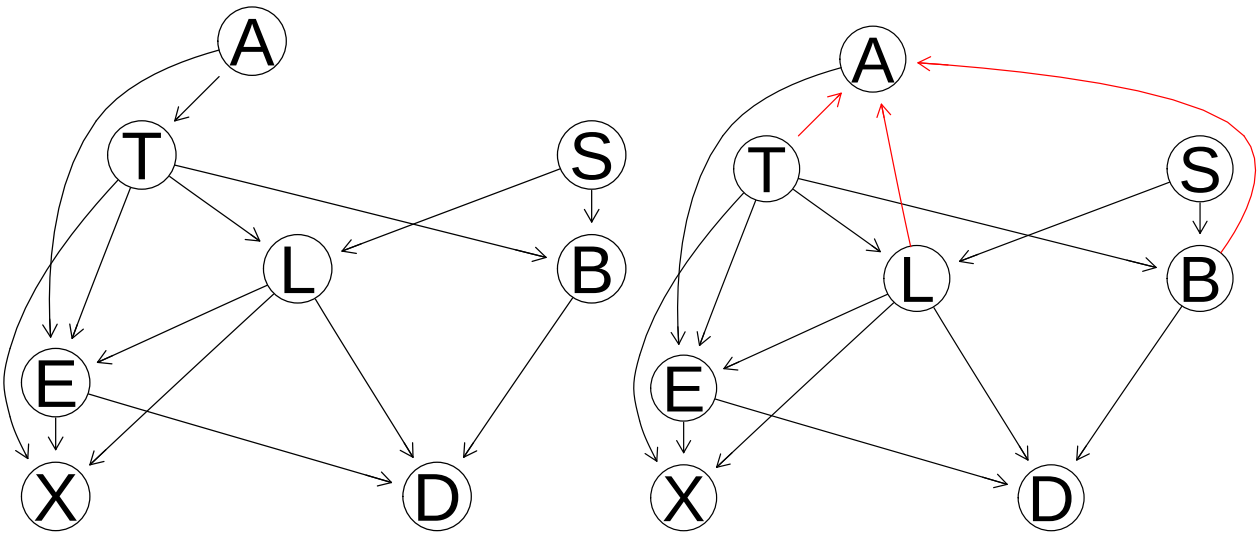


Figure 1: Comparison between DAGs generated by HC for different restarts and iss=15

Assignment 2

Learn a BN from 80 % of the Asia dataset. Learn both the structure and the parameters. Use the BN learned to classify the remaining 20 % of the Asia dataset in two classes: S = yes and S = no. To do so, you have to use exact or approximate inference. Compare your results with those of the true Asia BN.

```
data('asia')
n=dim(asia)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.8))
data=asia[id,]
test = asia[-id,]


information <- function(hc1, hc2){
  print(arcs(hc1))
  print(arcs(hc2))
  print('VSTRUCTS')
  #moral = false is the default value and it makes vstructs return unshielded colliders
  print(vstructs(hc1, moral=FALSE))
  print(vstructs(hc2, moral=FALSE))
  print('CPDAG:')
  cpdag(hc1)
  cpdag(hc2)
  all.equal(hc1,hc2)
}


hc_restart100 = hc(data, restart=100, score="bde", iss=5)

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

information(hc_restart100, true_dag)
```

```
##       from to
##  [1,] "S"  "L"
##  [2,] "T"  "X"
##  [3,] "A"  "E"
##  [4,] "E"  "D"
##  [5,] "S"  "B"
##  [6,] "T"  "E"
##  [7,] "E"  "X"
##  [8,] "L"  "E"
##  [9,] "L"  "X"
## [10,] "B"  "D"
## [11,] "T"  "A"
##      from to
## [1,] "A"  "T"
## [2,] "S"  "L"
## [3,] "S"  "B"
## [4,] "B"  "D"
## [5,] "E"  "D"
## [6,] "T"  "E"
## [7,] "L"  "E"
## [8,] "E"  "X"
## [1] "VSTRUCTS"
##      X   Z   Y
## [1,] "A" "E" "L"
## [2,] "T" "E" "L"
## [3,] "T" "X" "L"
## [4,] "B" "D" "E"
##      X   Z   Y
## [1,] "B" "D" "E"
## [2,] "L" "E" "T"
## [1] "CPDAG:"
## [1] "Different number of directed/undirected arcs"
```

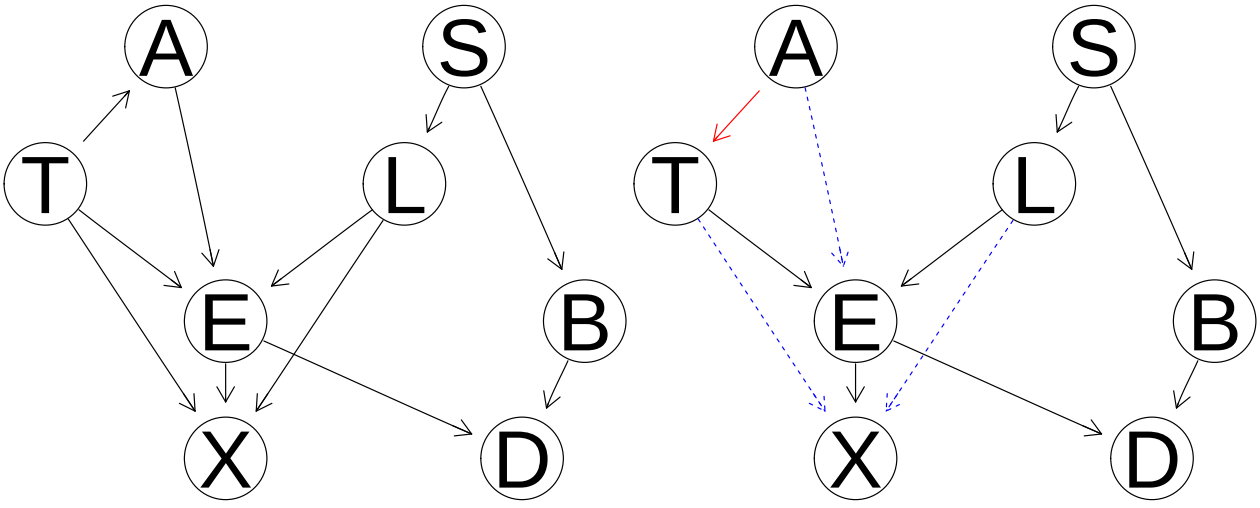
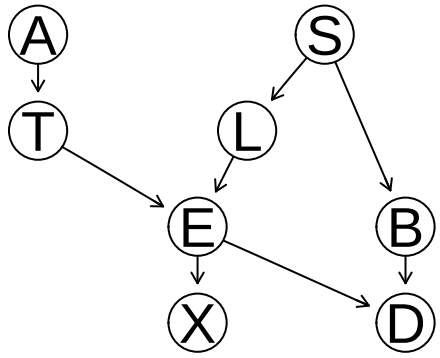


Figure 2: Comparison between DAG generated by hc and the true DAG

```
graphviz.plot(true_dag)
```




```
bn_fitted = bn.fit(hc_restart100, data = data)
true_bn_fitted = bn.fit(true_dag, data= data)

grain_obj = as.grain(bn_fitted)
```

```
## Warning in from.bn.fit.to.grain(x): NaN conditional probabilities in E, replaced
## with a uniform distribution.
```

```
## Warning in from.bn.fit.to.grain(x): NaN conditional probabilities in X, replaced
## with a uniform distribution.
```

```
true_grain_obj = as.grain(true_bn_fitted)

juncTree = compile(grain_obj)
true_juncTree = compile(true_grain_obj)
summary(juncTree)
```

```
## Independence network: Compiled: TRUE Propagated: FALSE
##  Nodes : Named chr [1:8] "A" "S" "T" "L" "B" "E" "X" "D"
##  - attr(*, "names")= chr [1:8] "A" "S" "T" "L" ...
##  Number of cliques:                 5
##  Maximal clique size:               4
##  Maximal state space in cliques:   16
```

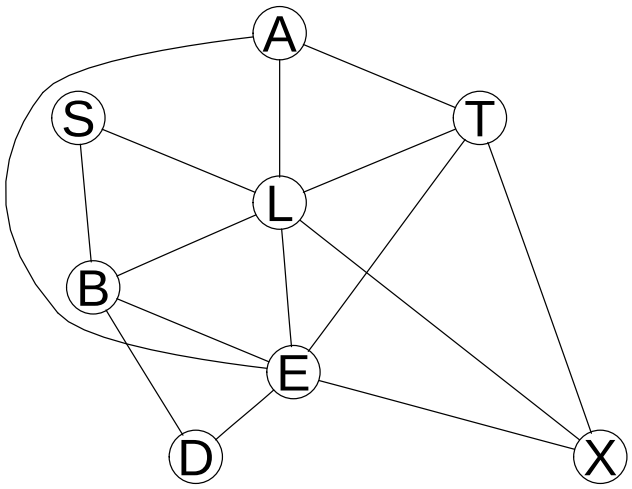


Figure 3: The junction tree created from the DAG through the junction tree algorithm. The junction tree algorithm turns the graph into a tree of clusters where each cluster is connected to a factor from the VE process.

```
predictfunc <- function(testdata, tree, xvars){
  prediction = c()
  for (i in 1:dim(testdata)[1]){
    xvals <- testdata[i, xvars]
    xvals <- as.vector(unlist(xvals, use.names = FALSE))
    states <- setEvidence(tree, nodes = xvars, states = xvals)
    probs <- querygrain(states, nodes="S")
    prediction[i] <- names(which.max(probs$S))

  }
  return(prediction)
}
xvars <- colnames(test[, -which(names(test) == "S")])
pred <- predictfunc(test, juncTree, xvars)
asia_Pred <- predictfunc(test, true_juncTree, xvars)
true_s <- test$S
table(true_s, pred)
```

```
##       pred
## true_s  no yes
##    no  337 176
##    yes 121 366
```

```
table(true_s, asia_Pred)
```

```
##       asia_Pred
## true_s  no yes
##    no  337 176
##    yes 121 366
```

```
table(pred, asia_Pred)
```

```
##      asia_Pred
## pred   no yes
##   no  458   0
```

```
##   yes   0 542
```

As can be seen from the confusion matrices, predicting S based on exact inference from the DAG generated through the HC algorithm and from the true DAG generates the exact same results. This is due to the fact that the markov blanket of S is the same for both of the DAGs. More on this in assignment 3.

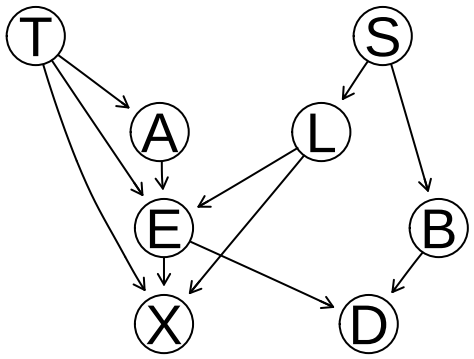
Assignment 3

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.

A subset that contains all the useful information is called a Markov blanket. Since S has no children and therefore no parents of it's children, the Markov blanket only consists of S's parents themselves. In an undirected graph (markov network) the markov blanket is all of the variables that are connected to the variable in question via an edge.

All of the variables not in the markov blanket of S are independent of S and does therefore not influence S in any way. The forementioned fact leads to that S can be as well predicted through only the Markov Blanket as through the whole DAG.

```
graphviz.plot(bn_fitted)
```




```
mb_nodes = mb(x=bn_fitted, node='S')
xvars <- colnames(test[, mb_nodes])
xvars
```

```
## [1] "L" "B"
```

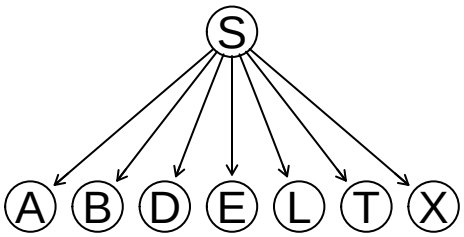
```
mb_pred <- predictfunc(test, juncTree, xvars)
table(true_s, pred)
```

```
##       pred
## true_s  no yes
##    no  337 176
##    yes 121 366
```

Assignment 4

Classification through naïve bayes is done by applying Bayes rule to compute the probability of C given the particular instance of variables x1,…,xn, and from that predicting the class with the highest posterior probability. This computation can be done by making the independence assumption that all the attributes xi are conditionally independent given the value of the class C. For the Naive Bayesian Network all features are considered as attributes and are independent given the class.

```
naive_dag = model2network("[S][A|S][B|S][T|S][L|S][E|S][X|S][D|S]")
graphviz.plot(naive_dag, main="Naïve Bayes Classifier")
```




```
naive_fitted = bn.fit(naive_dag, data = data)
naiveGrain_obj = as.grain(naive_fitted)
naiveJuncTree = compile(naiveGrain_obj)
xvars <- colnames(test[, -which(names(test) == "S")])
naive_prediction <- predictfunc(test, naiveJuncTree, xvars)
table(true_s, naive_prediction)
```

```
##       naive_prediction
## true_s  no yes
##    no  359 154
##    yes 180 307
```

The resulting confusion matrix from the Naïve Bayes Classifier is a bit worse off than the confusion matrices where the Markov Blanket could be used. The Naïve Bayes Classifier is most often used when our data set is small, i.e we have a small amount of observations since no good probabilistic inference can be done in that case. In our case with the Asia data set that contains 5000 observations, reasonably good probabilistic inference is attained, which trumphs the Naïve Bayes Classifier.

Assignment 5

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

The explanation to this assignment is described throughout assignments 2-4.