Lab1 Group Report

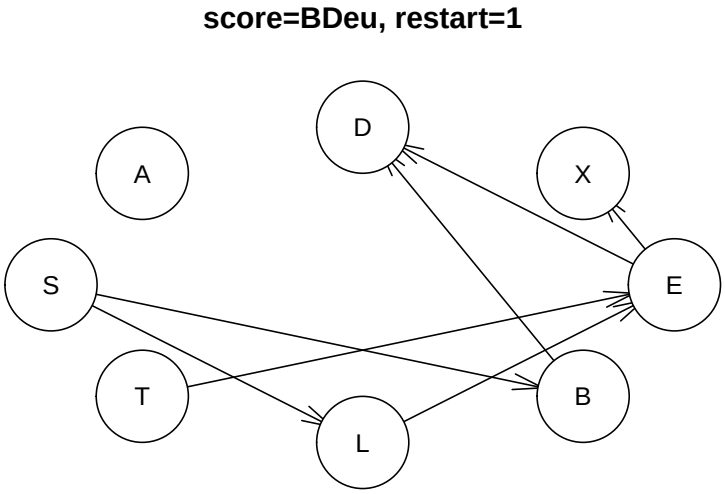
Zuxiang Li,Jooyoung Lee,Bayu Brahmantio,Weng Hang Wong

2020/9/13

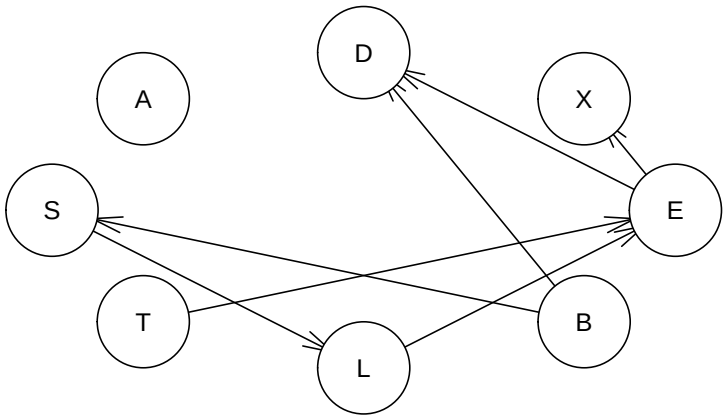
Question 1.

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 included in the bnlearn package. To load the data, run *data("asia")*.

Hint: Check the function hc in the bnlearn package. Note that you can specify the initial structure, the number of random restarts, the score, and the equivalent sample size (a.k.a imaginary sample size) in the BDeu score. You may want to use these options to answer the question. You may also want to use the functions *plot, arcs, vstructs, cpdag and all.equal.*




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




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

After a few trials, I use the score of BDe with different restart value of 1 and 100. From the below plot, the orientation between S<-L, S->L is different between restart = 1 and restart = 100.

Since HC algorithm is belongs to the family of local search, the restarting point is an arbitrary point that the HC algorithm starts to search(climb), when it reach to a local maxima then the algorithm will stop. So the HC algorithm is not so accurate sometimes because it will get stuck in the local maxima.

Using function *all.equal* can compare the two BN structure, so the two graph is not equivalent with different arces.

```
all.equal(hc_1, hc_2)
```

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

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]").

Hint: You already know the Lauritzen-Spiegelhalter algorithm for inference in BNs, which is an exact algorithm. There are also approximate algorithms for when the exact ones are too demanding computationally. For exact inference, you may need the functions bn.fit and as.grain from the bnlearn package, and the functions compile, setFinding and querygrain from the package gRain. For approximate inference, you may need the functions prop.table, table and cpdist from the bnlearn package

```
## Confusion Matrix:
##
##      pred_state
##        no yes
##   no  329 168
##   yes 124 379
##
##  Confusion Matrix in Percentage:
##
##      pred_state
##          no   yes
##   no  0.329 0.168
##   yes 0.124 0.379
##
##  Accuracy of the Trained BN:  0.708
## Confusion Matrix with the True BN:
##
##      true_pred_state
##        no yes
##   no  329 168
##   yes 124 379
##
##  Confusion Matrix in Percentage with the True BN:
##
##      true_pred_state
##          no   yes
##   no  0.329 0.168
##   yes 0.124 0.379
##
```

```
##  Accuracy of the True BN:  0.708
```

The fitted HC algorithm based BN showed the same result to the true Asia BN, considering returned confusion matrix and calculated accuracy using the test data.

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.

Hint: You may want to use the function mb from the bnlearn package.

```
##  Markov blanket:  L B

##
##  Confusion Matrix with Markov Blanket:
##

##      q3_pred_state
##        no yes
##   no  329 168
##   yes 124 379

##
##  Confusion Matrix in Percentage with Markov Blanket:
##

##      q3_pred_state
##          no   yes
##   no  0.329 0.168
##   yes 0.124 0.379

##
##  Accuracy of the BN with Markov Blanket:  0.708

##  Confusion Matrix with Markov Blanket (based on true BN):
##

##      q31_pred_state
##        no yes
##   no  329 168
##   yes 124 379

##
##  Confusion Matrix in Percentage with Markov Blanket (based on true BN):
##

##      q31_pred_state
##          no   yes
##   no  0.329 0.168
##   yes 0.124 0.379

##
##  Accuracy of the Model with Markov Blanket (based on true BN):  0.708
```

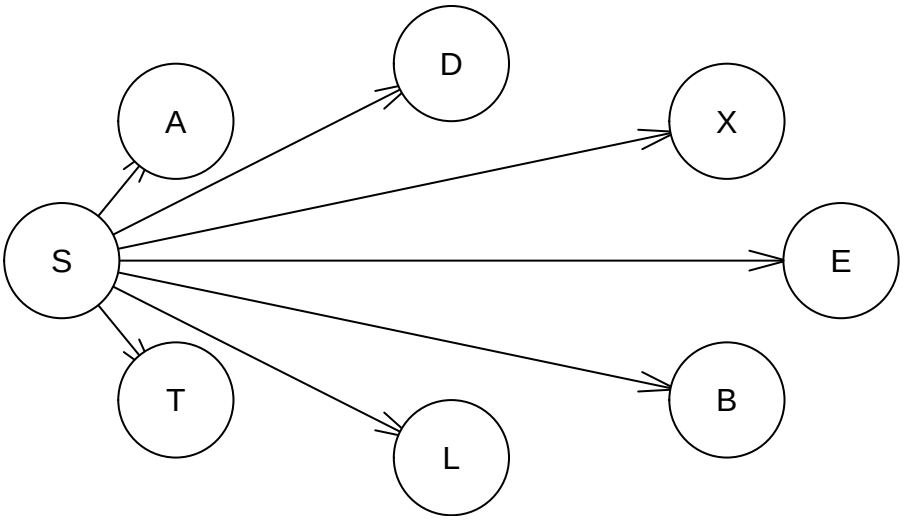
The accuracy of the BNs using Markov blanket are the same to the accuracy of the BNs in Question 2 above.

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.

Hint: Check http://www.bnlearn.com/examples/dag/ to see how to create a BN by hand.

```
## Warning in 1:dim(covariates[1]): numerical expression has 2 elements: only the
## first used
```




```
##
##  Confusion matrix for nb_pred
##
##       no yes
##   yes 142 323
##   no  355 180
##
##  Confusion Matrix in Percentage with naive bayes
##
##          no   yes
##   yes 0.142 0.323
##   no  0.355 0.180
##
```

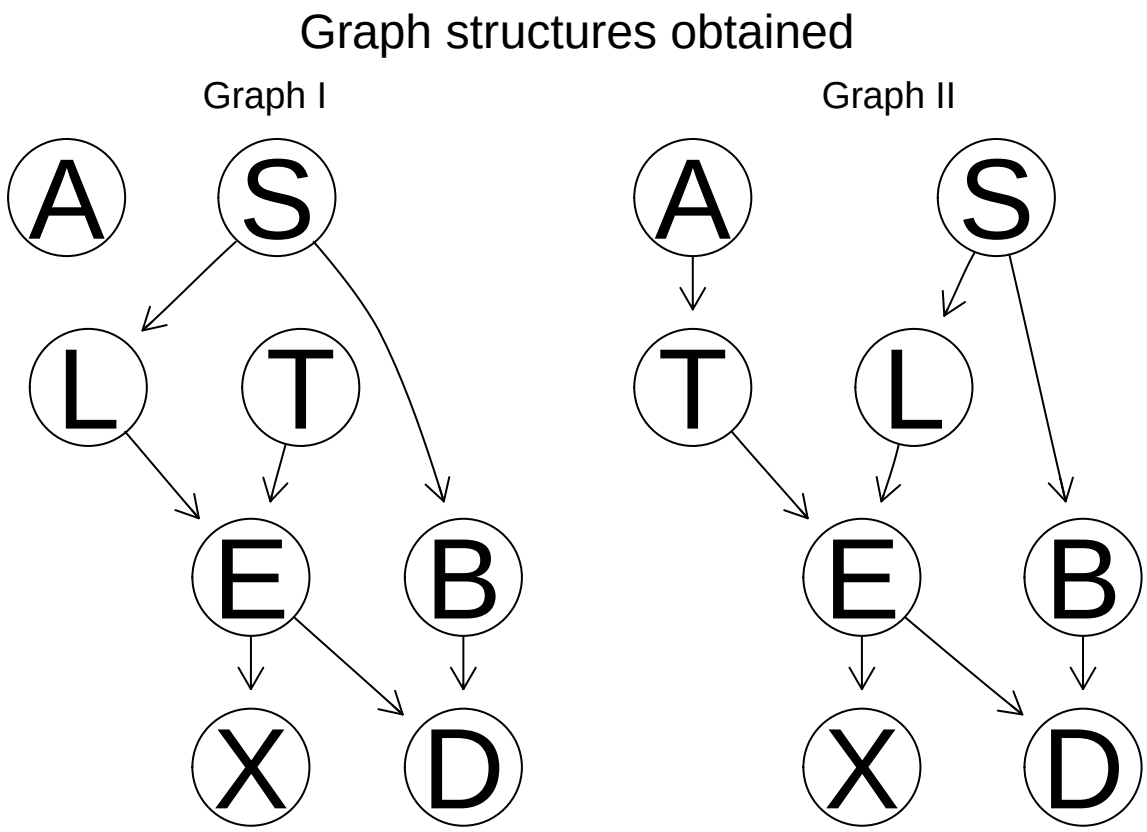
```
##  Accuracy using naive bayes: 0.678
```

Question 5

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

The results for question 2 and 3 are the same. This is because in all of the 3 cases (graph with learned structure, the true graph, and markov blanket), the graph structure used are similar.

```
#================================== Q5 ==================================
# plot graphs
par(mfrow=c(1,2), oma = c(0, 0, 2, 0))
graphviz.plot(model, main = "Graph I")
graphviz.plot(true_model, main = "Graph II")
mtext("Graph structures obtained", outer = TRUE, cex = 1.5)
```

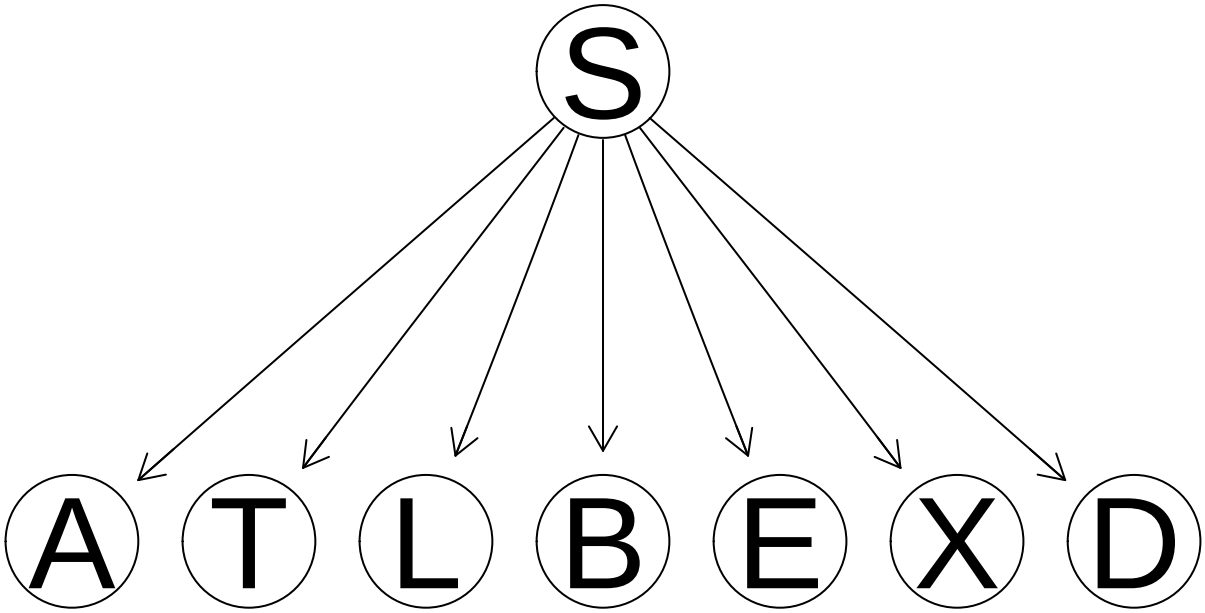


The structure of Graph I is learned through Hill-Climbing algorithm while Graph II is the true structure. By the property of Markov blanket, the conditional distribution of S given all remaining variables is dependent only on the variables in the Markov blanket. Because the Markov blanket of node S in both cases comprised only of nodes L and B, we can see that both of the results in the Question 2 are the same as the result given only variables in the Markov blanket (Question 3).

Question 4 gives a different result because the graph used is also different and therefore different markov blanket.

```
# plot graphs
par(mfrow=c(1,1), oma = c(0, 0, 2, 0))
graphviz.plot(nb_dag)
mtext("Graph structure of Naive Bayes Classifier", outer = TRUE, cex = 1.5)
```

Graph structure of Naive Bayes Classifier



In the graph above, all variables besides S are direct descendants of S. Therefore, the conditional distribution of S given all remaining variables is dependent on all of said remaining variables.

References

Højsgaard, S. Graphical Independence Networks with the `gRain` Package for R. Journal of Statistical Software 46, 2012.

Scutari, S. Learning Bayesian Networks with the `bnlearn` R Package. Journal of Statistical Software 35, 2010.

Bishop, C. M. Pattern Recognition and Machine Learning. New York: Springer, 2006, pp. 382-383.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
library(bnlearn)
library(gRain)
data("asia")
data=asia
# 1. ################################################################
set.seed(12345)
## BN with score bayesian dirichet equivilent, restart=10
hc_1 = hc(asia, score="bde", restart=1)
plot(hc_1, main="score=BDeu, restart=1")
arcs(hc_1)

# Bde, restart=100
hc_2 = hc(asia, score="bde", restart=100)
plot(hc_2, main="score=BDeu, restart=100")
arcs(hc_2)

#plot(hc(asia, score="bic", restart=50))
# imaginary sample size iss : BDe BGe, which is typically set to a very small value to reduce the relat
all.equal(hc_1, hc_2)
################################### Question 2 ######################################

# Dividing data into training and test dataset
n = dim(data)[1]
id = sample(1:n, floor(n*0.8))
training = data[id,]
test = data[-id,]
# not giving specific setting except the score for the initial BN
model = hc(data, score = 'bde')
fitted_model = bn.fit(model, training)
fitted_model_grain = as.grain(fitted_model) # since it returned Compiled: TRUE for summary(fitted_model


# produce prediction vector on S with test data using trained BN
# condition means it is observed
condition_node = colnames(data)[colnames(data)!="S"]
condition_df = test[,colnames(data)!="S"]
pred_state = c()

for (i in 1:dim(test)[1]){
        condition_state = c()

        for (j in 1:dim(condition_df)[2]){
                if (condition_df[i,j] == "yes") {
                        condition_state = append(condition_state, "yes")
                } else {
                                condition_state = append(condition_state, "no")
                        }
        }

        findings = setFinding(fitted_model_grain, nodes = condition_node, states = condition_state)
```

```
        probs = querygrain(findings, nodes = "S", type = "marginal")
        if (probs$S[1] > probs$S[2]) {
                pred_state = append(pred_state, names(probs$S[1]))
        } else {
                pred_state = append(pred_state, names(probs$S[2]))
        }
}
cat("Confusion Matrix: ", "\n", "\n")
tb = table(test[,colnames(data)=="S"], pred_state)
tb

cat("\n", "Confusion Matrix in Percentage:", "\n", "\n")
pct = prop.table(tb)
pct

cat("\n", "Accuracy of the Trained BN: ", sum(diag(pct)))

# true BN given
true_model = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
dag = bn.fit(true_model, training)
dag = as.grain(dag)

#produce prediction vector on S with test data using the true BN
true_pred_state = c()

for (i in 1:dim(test)[1]){

        condition_state = c()

        for (j in 1:dim(condition_df)[2]){
                if (condition_df[i,j] == "yes") {
                        condition_state = append(condition_state, "yes")
                } else {
                                condition_state = append(condition_state, "no")
                        }
        }

        findings = setFinding(dag, nodes = condition_node, states = condition_state)
        probs = querygrain(findings, nodes = "S", type = "marginal")

        if (probs$S[1] > probs$S[2]) {
                true_pred_state = append(true_pred_state, names(probs$S[1]))
        } else {
                true_pred_state = append(true_pred_state, names(probs$S[2]))
        }
}

cat("Confusion Matrix with the True BN: ", "\n", "\n")
true_tb = table(test[,colnames(data)=="S"], true_pred_state)
true_tb

cat("\n", "Confusion Matrix in Percentage with the True BN:", "\n", "\n")
true_pct = prop.table(true_tb)
```

```
true_pct

cat("\n", "Accuracy of the True BN: ", sum(diag(true_pct)))

################################## Question 3 ###################################

# find Markov blanket of S
M_blanket = as.vector(mb(fitted_model, "S"))

cat(" Markov blanket: ", M_blanket, "\n")
indicies = c()
for (i in 1:length(M_blanket)){
        indicies = append(indicies, which(colnames(data)==M_blanket[i]))
}

# dataframe of test dataset consisting only Markov blanket of S
q3_test = test[,indicies]

# produce prediction vector on S with markov blanket test data using trained BN

q3_pred_state = c()

for (i in 1:dim(test)[1]){
        condition_state = c()

        for (j in 1:dim(q3_test)[2]){
                if (q3_test[i,j] == "yes") {
                        condition_state = append(condition_state, "yes")
                } else {
                                condition_state = append(condition_state, "no")
                        }
        }

        findings = setFinding(fitted_model_grain, nodes = M_blanket, states = condition_state)
        probs = querygrain(findings, nodes = "S", type = "marginal")
        if (probs$S[1] > probs$S[2]) {
                q3_pred_state = append(q3_pred_state, names(probs$S[1]))
        } else {
                q3_pred_state = append(q3_pred_state, names(probs$S[2]))
        }
}

cat("\n","Confusion Matrix with Markov Blanket: ", "\n", "\n")
q3_tb = table(test[,colnames(data)=="S"], q3_pred_state)
q3_tb

cat("\n", "Confusion Matrix in Percentage with Markov Blanket:", "\n", "\n")
q3_pct = prop.table(q3_tb)
q3_pct

cat("\n", "Accuracy of the BN with Markov Blanket: ", sum(diag(q3_pct)))

q31_pred_state = c()
```

```
for (i in 1:dim(test)[1]){
        condition_state = c()

        for (j in 1:dim(q3_test)[2]){
                if (q3_test[i,j] == "yes") {
                        condition_state = append(condition_state, "yes")
                } else {
                                condition_state = append(condition_state, "no")
                        }
        }

        findings = setFinding(dag, nodes = M_blanket, states = condition_state)
        probs = querygrain(findings, nodes = "S", type = "marginal")
        if (probs$S[1] > probs$S[2]) {
                q31_pred_state = append(q31_pred_state, names(probs$S[1]))
        } else {
                q31_pred_state = append(q31_pred_state, names(probs$S[2]))
        }
}

cat("\n", "\n")
cat(" Confusion Matrix with Markov Blanket (based on true BN): ", "\n", "\n")
q31_tb = table(test[,colnames(data)=="S"], q31_pred_state)
q31_tb

cat("\n", "Confusion Matrix in Percentage with Markov Blanket (based on true BN):", "\n", "\n")
q31_pct = prop.table(q31_tb)
q31_pct

cat("\n", "Accuracy of the Model with Markov Blanket (based on true BN): ", sum(diag(q31_pct)))

n=length(colnames(training))
adj=matrix(0L,ncol=n,nrow=n,dimnames = list(colnames(training),colnames(training)))
adj["S",]=1
adj["S","S"]=0
nb_dag=empty.graph(colnames(training))
amat(nb_dag)=adj

nb_res=bn.fit(nb_dag,training)

a=as.grain(nb_res)
CPT=compileCPT(a$cptlist)
nb_chest.bn=grain(CPT)


covariates=test[,-2]


# nb prediction
nb_pred=data.frame(res=c())
for(i in c(1:dim(covariates[1]))){
  state=as.vector(as.matrix(covariates[i,]))
  finding=setFinding(nb_chest.bn,c("A","T","L","B","E","X","D"),states = state)
```

```
  query=querygrain(finding,c("S"))
  ans=rownames(query$S)[which.max(query$S)]
  nb_pred=rbind(nb_pred,data.frame(res=ans))
}

plot(nb_dag)
cat("\n Confusion matrix for nb_pred")
nb_cft=table(nb_pred$res,test$S)
print(nb_cft)

nb_ptb=prop.table(nb_cft)
cat("\n Confusion Matrix in Percentage with naive bayes")

print(nb_ptb)
nb_acc=1-sum(diag(nb_ptb))
cat("\n Accuracy using naive bayes:",nb_acc)
#=================================== Q5 ====================================
# plot graphs
par(mfrow=c(1,2), oma = c(0, 0, 2, 0))
graphviz.plot(model, main = "Graph I")
graphviz.plot(true_model, main = "Graph II")
mtext("Graph structures obtained", outer = TRUE, cex = 1.5)
# plot graphs
par(mfrow=c(1,1), oma = c(0, 0, 2, 0))
graphviz.plot(nb_dag)
mtext("Graph structure of Naive Bayes Classifier", outer = TRUE, cex = 1.5)
```