Lab1 Bayesian Network

Yifan Ding, Chao Fu, Yunan Dong, Ravinder Reddy

28 September, 2021

Contributions

We disscussed and solved all questions together, Yifan mainly contribute to Q1.1, Chao mainly contribute to Q1.2, Yunan mainly contribute to Q1.3, and Ravinder mainly contributed to Q1.4. We disscussed and wrote Q1.5 together.

Q1.1

1.Default setting

```r
data("asia")
dag1 <-hc(asia)
```

2.Consider a different start structure with $B \longrightarrow L \longrightarrow S$ rather than empty DAG:

```r
init <- model2network("[A][T][B][L|B][S|L][E][X][D]")
dag2 <-hc(asia, start=init)
```

3. $L \longrightarrow S \longrightarrow B$:

```r
init <- model2network("[A][T][B|S][S|L][L][E][X][D]")
dag3 <-hc(asia, start=init)
```

4.AIC Score

```r
dag4 <-hc(asia, score='aic')
```

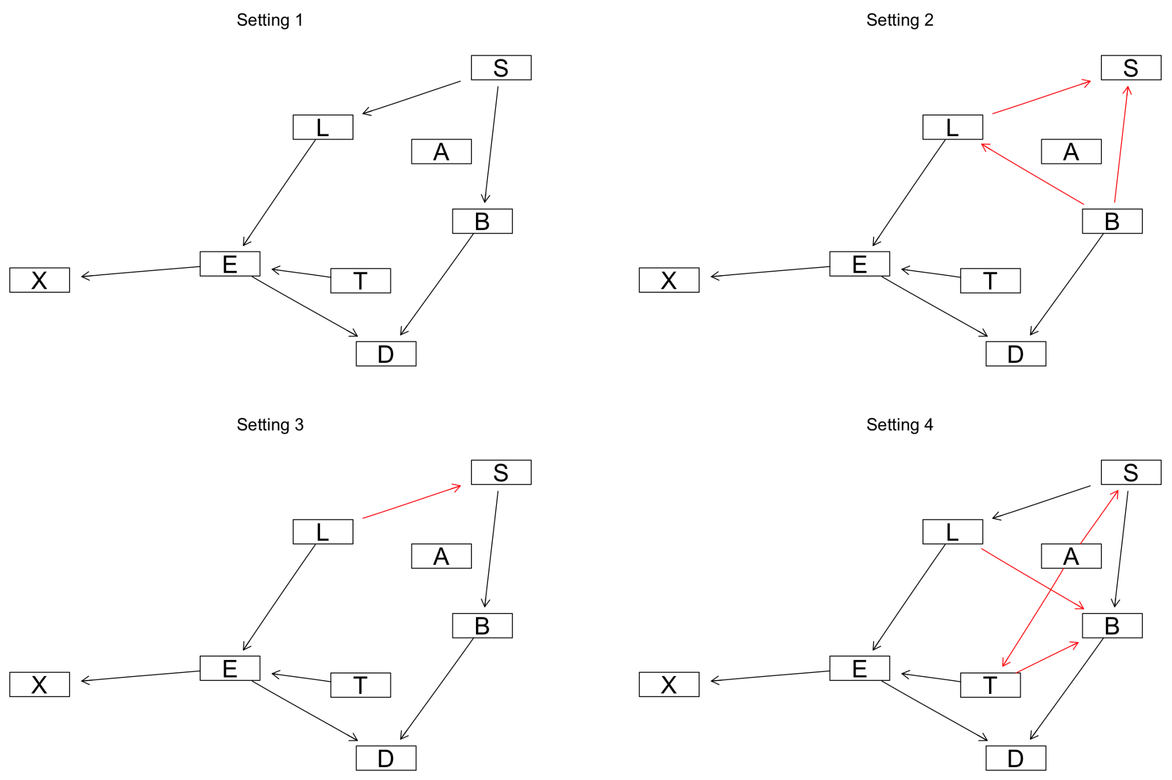
A more general way to compare graphs if they are equivalent by using **cpdag**

```r
all.equal(cpdag(dag1), cpdag(dag2))
```

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

```r
all.equal(cpdag(dag1), cpdag(dag3))
```

```
## [1] TRUE
```



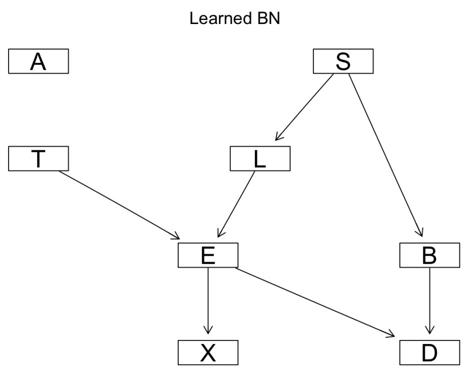
Show that multiple runs of the hill-climbing algorithm can return non-equivalent Bayesian network (BN) structures. Explain why this happens.

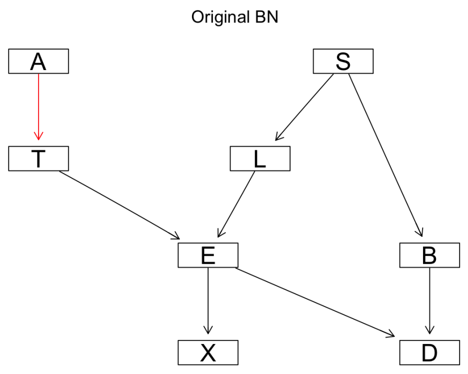
Answer: With different setting of HC, We have:

- 1. $L \longleftarrow S \longrightarrow B$ (Default setting)
- 2. Shielded collider $B \longrightarrow S \longleftarrow L$ with shield $B \longrightarrow L$ (Initial structure $B \longrightarrow L \longrightarrow S$)
- 3. $L \longrightarrow S \longrightarrow B$ (Initial structure $L \longrightarrow S \longrightarrow B$)
- 4. Then got a graph quite unreasonable (Using AIC score)

In case 1,3, they are equivalent but only with different direction of edges(they are equivalent in probabilistic form, which will return the same score). Case 2 has one more edge than 1,3, case 4 generate a graph with even more edges and quite unreasonable. Therefore under some different setting HC generated non-equivalent structures, this is because HC algorithm randomly add, remove and reverse edges. For example, First we initialize the graph with some settings, then calculate The Score(BIC or other scores) for this graph, next, we add an edge, calculate The Score again, if this score is lager than previous score, we keep the change. We again add, remove or change the direction of arrows, calculate The Score, if The Score is lager than previous, save The Score and keep the change, if not, keep previous graph. After many times calculation, when we change any part of the graph, The Score won't change. If we remove an edge(this will produce no-equivalent graph),the score still not change,thus HC stack here in a local minimum.

Q1.2





```
#asia<-apply(asia, 2, as.character)

#train and test split
n <- dim(asia)[1]
set.seed(1010)
id <- sample(1:n, floor(n*0.8))
train <- asia[id,]

id1 <- setdiff(1:n, id)
set.seed(1010)
#id2 <- sample(id1, floor(n*0.2))
test <- asia[id1,]
```

```
# Create structure
structure <- hc(train)
fit <- bn.fit(x = structure, data = train)  # training data , structure learning
fit_grain <- as.grain(fit)
```

```
compiled_grain <- compile(fit_grain)  # necessary syntax to use needed proporties

# Manipulating data, for the function querygrain needs the data to be in character form
test2 <- test
test <- apply(test, 2, as.character)

# nodes to S: "A", "T", "L", "B", "E", "X", "D"
nodes_2_S<-colnames(asia)[-2]
```

```
compiled_obj = compiled_grain
nodes_from = nodes_2_S  # nodes towards s
node_to = "S"  # objective variable
new_data = test[, -2]  # data

# my prediction function
my_pred<-function(compiled_obj,nodes_from, node_to,new_data){

  predictions <- c()

  for (i in 1:nrow(new_data)){
    evidence <- setEvidence(object = compiled_obj,
                            nodes = nodes_from,  # node names
```

```r
                              states = new_data[i,] )   # data

  posterior <- unlist(querygrain(object = evidence, nodes=node_to))

  if (posterior[1] > 0.5) {
    predictions[i] <- "No"}
  else {
    predictions[i] <- "Yes"  }

  }

return(predictions)
}

res=my_pred(compiled_obj,nodes_from, node_to,new_data)
confusion_matrix <- table(test2$S, res)
print('Confusion Matrix of Learned Graph')
```

```
## [1] "Confusion Matrix of Learned Graph"
```

```r
confusion_matrix
```

```
##      res
##         No Yes
##   no  339 149
##   yes 130 382
```

```r
sum(confusion_matrix[c(1,4)])/sum(confusion_matrix)
```

```
## [1] 0.721
```

The True Graph

```r
# True Bayesian Network
true_dag <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
fit_true <- bn.fit(x = true_dag, data = train)%>%as.grain()
```

```r
# parameters involved

compiled_obj = compile(fit_true)  # true dag model
nodes_from = nodes_2_S
node_to = "S"
new_data = test[, -2]

res=my_pred(compiled_obj,nodes_from, node_to,new_data)
confusion_matrix <- table(test2$S, res)
print('Confusion matrix of True Graph')
```

```
## [1] "Confusion matrix of True Graph"
```

```r
confusion_matrix
```

```
##      res
##         No Yes
##   no  339 149
##   yes 130 382
```

```r
sum(confusion_matrix[c(1,4)])/sum(confusion_matrix)
```

```
## [1] 0.721
```

Q1.3

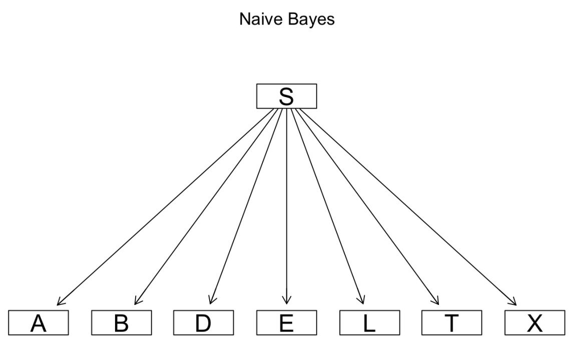
```r
markov_blanket <- mb(x = fit, node = "S")

# parameters involved
compiled_obj = compiled_grain  # markov obj
nodes_from = markov_blanket   # cliques toward S
node_to = "S"
new_data = test[, markov_blanket]

# predict
res=my_pred(compiled_obj,nodes_from, node_to,new_data)
# res
confusion_matrix <- table(test2$S, res)
confusion_matrix
```

```
##      res
##        No Yes
##   no  339 149
##   yes 130 382
```

```r
sum(confusion_matrix[c(1,4)])/sum(confusion_matrix)
```

```
## [1] 0.721
```

Q1.4



```r
# Naive Bayes:
naive_bayes = model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")
# building  object
naive_bayes <- bn.fit(x = naive_bayes, data = train) %>%as.grain() %>%compile()


# parameters involved
compiled_obj = naive_bayes
nodes_from = nodes_2_S
node_to = "S"
new_data = test[, -2]
```

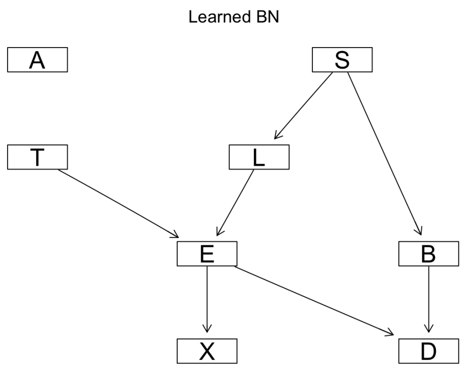
```r
# predict
res=my_pred(compiled_obj,nodes_from, node_to,new_data)
# res
confusion_matrix <- table(test2$S, res)
confusion_matrix
```

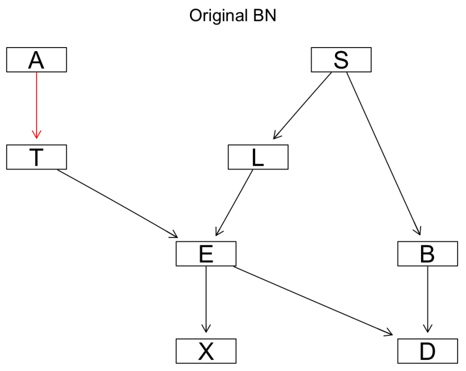
```
##      res
##        No Yes
##   no  368 120
##   yes 185 327
```

```r
sum(confusion_matrix[c(1,4)])/sum(confusion_matrix)
```

```
## [1] 0.695
```

Q1.5





We got the same results for Q1.2 Generated Graph and Q1.3 Markov Blanket, but different results for Q1.4 Naive Bayes. The explaination as follow:

First we consider the conditional distribution of S in Asia Network, and try see this from a variable elimination point of view:

$$P(S|B,L,E,D,X,T,A) = \frac{P(S)P(B|S)P(L|S)P(D|B,E)P(E|T,L)P(T|A)P(A)}{\sum_S P(S)P(B|S)P(L|S)P(D|B,E)P(E|T,L)P(T|A)P(A)}$$
$$= \frac{P(S)P(B|S)P(L|S)}{\sum_S P(S)P(B|S)P(L|S)} = \frac{P(S,B,L)}{P(B,L)} = P(S|B,L)$$

Note $P(S|B,L)$ is the posterior of Markov blanket S with B,L. Regardless the structure of $P(E,D,X,T,A|B,L)$, the partial structure $L \longleftarrow S \longrightarrow B$ is the same in Q1.2 and Q1.3. Above formula hold for all 3 situations, the confusion matrix thus should be the same.

We can also simply see it from a graphical point of view (conditional independent):

$$S \perp E,D,X,T,A|B,L$$

Thus for both two graph:

$$P(S|B,L,E,D,X,T,A) = P(S|B,L)$$

But in Q1.4 the Naive Bayes model is:

$$P(S|B,L,E,D,X,T,A) = \frac{P(S)P(B,L,E,D,X,T,A|S)}{\sum_S P(S)P(B,L,E,D,X,T,A|S)} = \frac{P(S)P(B,L,E,D,X,T,A|S)}{P(B,L,E,D,X,T,A)}$$

in which above formula cannot be further simplified like previous Bayesian network. Then this can returns a different result. So far, we only know Naive Bayes has lower accuracy from above experiment, but why Naive Bayes has lower accuracy comparing with Bayesian Network? Regard this, we need to compare $P(S|B,L,E,D,X,T,A)$ with $P(S|B,L)$, Bayesian network could find conditional independent in variables, such as $S \perp E,D,X,T,A|B,L$. This means no matter what E,D,X,T,A are, once we have B,L we have all the information to predict the posterior of S, but Naive Bayes insist to add E,D,X,T,A to calculate the posterior regardless the dependencies. This actually included noises(E,D,X,T,A are conditional independent with S, so they are noise) into calculations, therefore lead to worse results.

Apendix

```r
knitr::opts_chunk$set(echo = TRUE)
library(bnlearn)
library(gRain)
data("asia")
dag1 <-hc(asia)
init <- model2network("[A][T][B][L|B][S|L][E][X][D]")
dag2 <-hc(asia, start=init)
init <- model2network("[A][T][B|S][S|L][L][E][X][D]")
dag3 <-hc(asia, start=init)
dag4 <-hc(asia, score='aic')
all.equal(cpdag(dag1), cpdag(dag2))
all.equal(cpdag(dag1), cpdag(dag3))
graphviz.compare(dag1, dag2, dag3, dag4, shape = "rectangle", layout = "fdp",
  main = c("Setting 1", "Setting 2", "Setting 3", "Setting 4"))
#asia<-apply(asia, 2, as.character)

#train and test split
n <- dim(asia)[1]
set.seed(1010)
id <- sample(1:n, floor(n*0.8))
train <- asia[id,]

id1 <- setdiff(1:n, id)
set.seed(1010)
#id2 <- sample(id1, floor(n*0.2))
test <- asia[id1,]

# Create structure
structure <- hc(train)
fit <- bn.fit(x = structure, data = train)  # training data , structure learning
fit_grain <- as.grain(fit)

compiled_grain <- compile(fit_grain)  # necessary syntax to use needed proporties

# Manipulating data, for the function querygrain needs the data to be in character form
test2 <- test
test <- apply(test, 2, as.character)

# nodes to S: "A", "T", "L", "B", "E", "X", "D"
```

```r
nodes_2_S<-colnames(asia)[-2]


compiled_obj = compiled_grain
nodes_from = nodes_2_S  # nodes towards s
node_to = "S"  # objective variable
new_data = test[, -2]  # data

# my prediction function
my_pred<-function(compiled_obj,nodes_from, node_to,new_data){

  predictions <- c()

  for (i in 1:nrow(new_data)){
    evidence <- setEvidence(object = compiled_obj,
                            nodes = nodes_from,  # node names
                            states = new_data[i,] )   # data

  posterior <- unlist(querygrain(object = evidence, nodes=node_to))

  if (posterior[1] > 0.5) {
    predictions[i] <- "No"}
  else {
    predictions[i] <- "Yes"  }

  }

return(predictions)
}

res=my_pred(compiled_obj,nodes_from, node_to,new_data)
confusion_matrix <- table(test2$S, res)
print('Confusion Matrix of Learned Graph')
confusion_matrix
sum(confusion_matrix[c(1,4)])/sum(confusion_matrix)
# True Bayesian Network
true_dag <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
fit_true <- bn.fit(x = true_dag, data = train)%>%as.grain()

# parameters involved

compiled_obj = compile(fit_true)  # true dag model
nodes_from = nodes_2_S
node_to = "S"
new_data = test[, -2]

res=my_pred(compiled_obj,nodes_from, node_to,new_data)
confusion_matrix <- table(test2$S, res)
print('Confusion matrix of True Graph')
confusion_matrix
sum(confusion_matrix[c(1,4)])/sum(confusion_matrix)

markov_blanket <- mb(x = fit, node = "S")
```

```r
# parameters involved
compiled_obj = compiled_grain  # markov obj
nodes_from = markov_blanket   # cliques toward S
node_to = "S"
new_data = test[, markov_blanket]

# predict
res=my_pred(compiled_obj,nodes_from, node_to,new_data)
# res
confusion_matrix <- table(test2$S, res)
confusion_matrix
sum(confusion_matrix[c(1,4)])/sum(confusion_matrix)
# Naive Bayes:
naive_bayes = model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")
# building  object
naive_bayes <- bn.fit(x = naive_bayes, data = train) %>%as.grain() %>%compile()


# parameters involved
compiled_obj = naive_bayes
nodes_from = nodes_2_S
node_to = "S"
new_data = test[, -2]

# predict
res=my_pred(compiled_obj,nodes_from, node_to,new_data)
# res
confusion_matrix <- table(test2$S, res)
confusion_matrix
sum(confusion_matrix[c(1,4)])/sum(confusion_matrix)
```