Advanced Machine Learning Lab 1 (Graphical Models)

Omkar Bhutra (omkbh878)

21 September 2019

Task 1:

```r
data("asia")
asia <- as.data.frame(asia)
#default parameter run of HC
hillClimbingResults = hc(asia)
print(hillClimbingResults)
```

```
##
##   Bayesian network learned via Score-based methods
##
##   model:
##    [A][S][T][L|S][B|S][E|T:L][X|E][D|B:E]
##   nodes:                                 8
##   arcs:                                  7
##     undirected arcs:                     0
##     directed arcs:                       7
##   average markov blanket size:           2.25
##   average neighbourhood size:            1.75
##   average branching factor:              0.88
##
##   learning algorithm:                    Hill-Climbing
##   score:                                 BIC (disc.)
##   penalization coefficient:              4.258597
##   tests used in the learning procedure:  77
##   optimized:                             TRUE
```

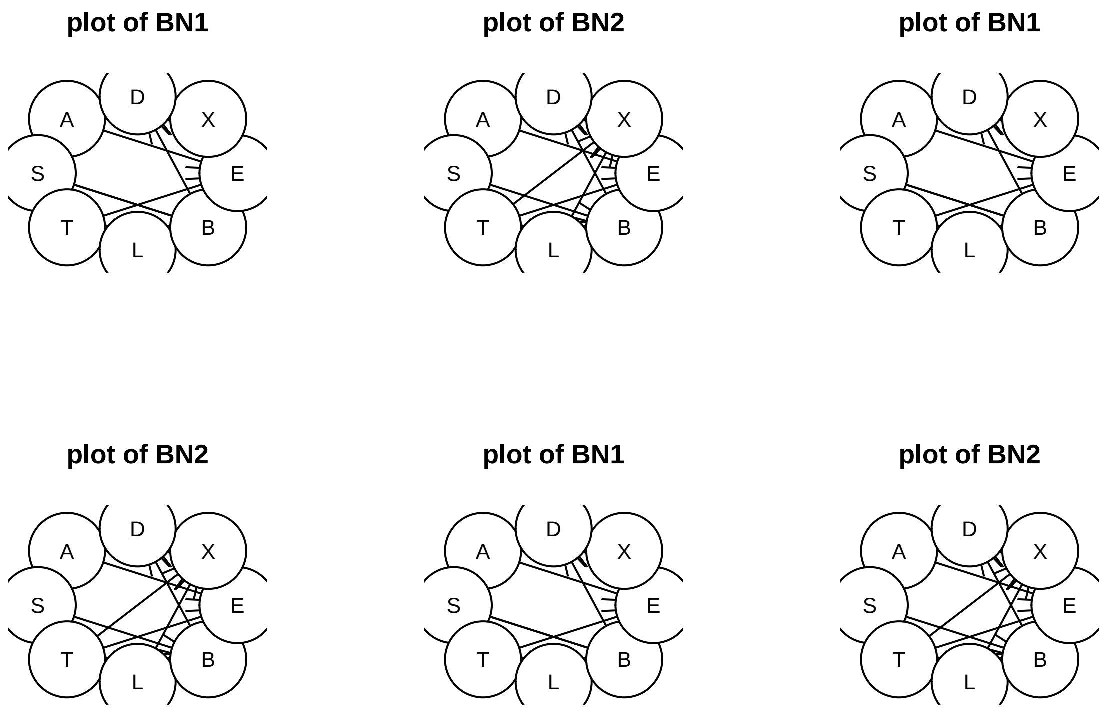
```r
par(mfrow = c(2,3))

hillclimb <- function(x){
hc1 <- hc(x,restart = 15,score = "bde", iss = 3)
hc2 <- hc(x,restart = 10,score = "bde", iss = 5)
hc1dag <- cpdag(hc1)
plot(hc1dag, main = "plot of BN1")
hc1arcs <- vstructs(hc1dag, arcs = TRUE)
# print(hc1arcs)
arcs(hc1)
hc2dag <- cpdag(hc2)
plot(hc2dag, main = "plot of BN2")
hc2arcs <- vstructs(hc2dag, arcs = TRUE)
# print(hc2arcs)
arcs(hc2)
print(all.equal(hc1dag,hc2dag))
}
```

```r
for(i in 1:3){
  hillclimb(asia)
}
```

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

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




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

Non-equivalant solutions are produced using different starting parameters such as number of restarts, scoring algorithm, imaginary sample size etc. Hill climbing algorithm is a deterministic one, and hence generates different results on different input parameters. Hill climbing leads to a local maxima of the objective function which makes two different results possible with the same code. Adding imaginary sample size affects the relationships between the nodes such that we get more edges. Number of restarts increases the possibility of getting the same result.

Task 2:

Network structure is trained by the hill climbing algorithm using the BDE score. bn.fit is used to learn the paramets using maximum liklihood estimation. To predict S in the test data , evidence in the network is set to the values of the test data excluding S, using setEvidence. querygrain is used to find the probabilty, maximum of the values is taken to find the misclassification rate.

```r
#setting up training and testing datasets
id <- sample(x = seq(1, dim(asia)[1], 1),
                        size = dim(asia)[1]*0.8,
                        replace = FALSE)
asia.train <- asia[id,]
asia.test <- asia[-id,]

bnprediction = as.numeric()
#fitting a model using Hill Climbing
bnmodel <- hc(asia.train,score = "bde",restart = 50)
bnmodelfit <- bn.fit(bnmodel,asia.train,method = 'mle') #max liklihood estimation
bngrain <- as.grain(bnmodelfit)
comp <- compile(bngrain)

bnmodel_true <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
bnmodelfit_true <- bn.fit(bnmodel_true, asia.train)
bngrain_true <- as.grain(bnmodelfit_true)
comp_true <- compile(bngrain_true)

bnpredict <- function(bntree, data, obs_variables, pred_variable) {
   for (i in 1:dim(data)[1]) {
    x <- NULL
    for (j in obs_variables) {
      x[j] <- if(data[i, j] == "yes") "yes" else "no"
    }
    evidence <- setEvidence(object = bntree,nodes = obs_variables,states=x)
    prob_dist_fit  <- querygrain(object = evidence,nodes = pred_variable)$S
    bnprediction[i] <- if (prob_dist_fit["yes"] >= 0.5) "yes" else "no"
  }
  return(bnprediction)
}

# Predict S from Bayesian Network and test data observations
bnprediction <- bnpredict(bntree = comp,
                                       data = asia.test,
                                       obs_variables = c("A", "T", "L", "B", "E", "X", "D"),
                                       pred_variable = c("S"))
bnprediction_true <- bnpredict(bntree = comp_true,
                                            data = asia.test,
                                            obs_variables = c("A", "T", "L", "B", "E", "X", "D"),
                                            pred_variable = c("S"))

# Calculate confusion matricies
confusion_matrix_fit <- table(bnprediction, asia.test$S)
print(confusion_matrix_fit)
```

```
##
## bnprediction  no yes
##          no  329 136
##          yes 145 390
```

```r
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit))/sum(confusion_matrix_fit)))
```

```
## [1] "Misclassification rate: 0.281"
```

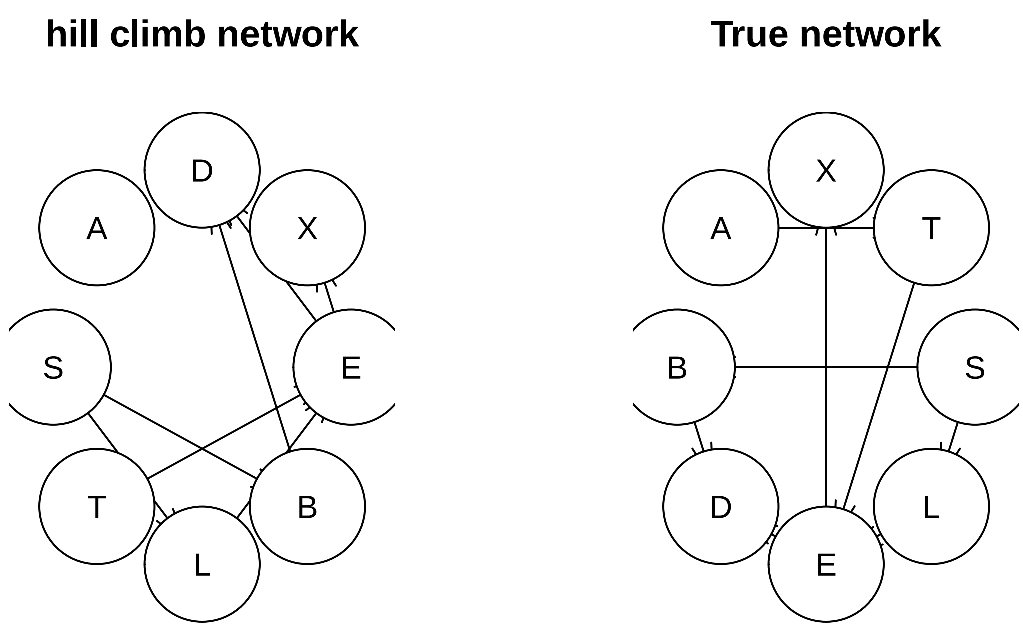
```r
confusion_matrix_true <- table(bnprediction_true, asia.test$S)
print(confusion_matrix_true)
```

```
##
## bnprediction_true  no yes
##               no  329 136
##               yes 145 390
```

```r
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_true))/sum(confusion_matrix_true)))
```

```
## [1] "Misclassification rate: 0.281"
```

```r
par(mfrow=c(1,2))
plot(bnmodel)
title("hill climb network")
plot(bnmodel_true)
title("True network")
```




```r
par(mfrow=c(1,1))
```

The Misclassification rate remain the same for both the hill climb network and the true network. However, The state of "Different number of directed/undirected arcs" persists.

Task 3:

Using the markov blanket, we predict the results again, it is expected to output the same result.

```
markov_blanket = mb(bnmodel, c("S"))
markov_blanket_true = mb(bnmodel_true, c("S"))
bnpredict_mb <- bnpredict(comp, asia.test, markov_blanket, c("S"))
bnpredict_mb_true <- bnpredict(comp_true, asia.test, markov_blanket_true, c("S"))
confusion_matrix_fit <- table(bnpredict_mb, asia.test$S)
confusion_matrix_fit_true <- table(bnpredict_mb_true, asia.test$S)
print(confusion_matrix_fit)
```

```
##
## bnpredict_mb  no yes
##          no  329 136
##          yes 145 390
```

```
print(confusion_matrix_fit_true)
```

```
##
## bnpredict_mb_true  no yes
##               no  329 136
##               yes 145 390
```

```
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit))/sum(confusion_matrix_fit)))
```

```
## [1] "Misclassification rate: 0.281"
```

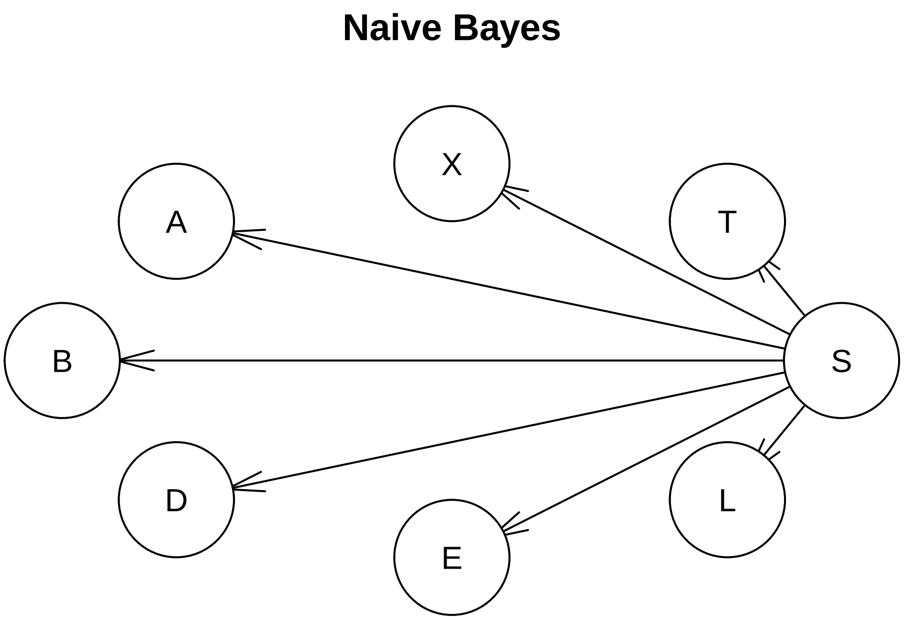
```
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit_true))/sum(confusion_matrix_fit_
```

```
## [1] "Misclassification rate: 0.281"
```

Task 4:

The naiveBayes structure implies that all variables are independant of S, this implies that there is a diffrent true structure used.

```
naiveBayes = model2network("[S][A|S][B|S][D|S][E|S][X|S][L|S][T|S]")
plot(naiveBayes, main = "Naive Bayes")
```




```
naiveBayes.fit <- bn.fit(naiveBayes, asia.train)
result_naive <- bnpredict(compile(as.grain(naiveBayes.fit)), asia.test, c("A", "T", "L", "B", "E", "X",
# Calculate confusion matricies
confusion_matrix_naive_bayes <- table(result_naive, asia.test$S)
print(confusion_matrix_naive_bayes)
```

```
##
## result_naive  no yes
##          no  349 208
##          yes 125 318
```

```
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_naive_bayes))/sum(confusion_matrix_na
```

```
## [1] "Misclassification rate: 0.333"
```

Task 5:

In Task 2, Same results are observed for the trained structure and the true structure. Same results are also obtained in Task 3 as before, this is expected due to the reduction of the node from the rest of the structure. The reduced number of predictors still provides the same misclassification rate although some information loss is expected. In Task 4, Naive bayes is not a predictor for the data as the structure used is different as the variables are inferred to be independant of S, but all are assumed to have an effect on S.

```r
knitr::opts_chunk$set(echo = TRUE)
library("dplyr")
library("ggplot2")
library("gRain")
library("bnlearn")
data("asia")
asia <- as.data.frame(asia)
#default parameter run of HC
hillClimbingResults = hc(asia)
print(hillClimbingResults)
par(mfrow = c(2,3))

hillclimb <- function(x){
hc1 <- hc(x,restart = 15,score = "bde", iss = 3)
hc2 <- hc(x,restart = 10,score = "bde", iss = 5)
hc1dag <- cpdag(hc1)
plot(hc1dag, main = "plot of BN1")
hc1arcs <- vstructs(hc1dag, arcs = TRUE)
# print(hc1arcs)
arcs(hc1)
hc2dag <- cpdag(hc2)
plot(hc2dag, main = "plot of BN2")
hc2arcs <- vstructs(hc2dag, arcs = TRUE)
# print(hc2arcs)
arcs(hc2)
print(all.equal(hc1dag,hc2dag))
}
for(i in 1:3){
  hillclimb(asia)
}
# Question 1: PGMs

# Learn a BN from the Asia dataset, find a separation statement (e.g. B _|_ E | S, T) and, then, check
# it corresponds to a statistical independence.
library(bnlearn)
library(gRain)
set.seed(123)
data("asia")
hc3<-hc(asia,restart=10,score="bde",iss=10)
plot(hc3)
hc4<-bn.fit(hc3,asia,method="bayes")
hc5<-as.grain(hc4)
hc6<-compile(hc5)
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("yes","yes","yes"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("yes","yes","no"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("yes","no","yes"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("yes","no","no"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("no","yes","yes"))
querygrain(hc7,c("B"))
```

```
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("no","yes","no"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("no","no","yes"))
querygrain(hc7,c("B"))
hc7<-setFinding(hc6,nodes=c("S","T","E"),states=c("no","no","no"))
querygrain(hc7,c("B"))

# Sample DAGs at random and, then, check which of them coincide with their CPDAGs, i.e. the CPDAGs have
# undirected edge (recall that a CPDAG has an undirected edge if and only if there are two DAGs in the
# equivalence class that differ in the direction of that edge).

#  The exact ratio according to the literature is 11.2
library(bnlearn)
set.seed(123)
ss<-50000
x<-random.graph(c("A","B","C","D","E"),num=ss,method="melancon",every=50,burn.in=30000)

y<-unique(x)
z<-lapply(y,cpdag)

r=0
for(i in 1:length(y)) {
  if(all.equal(y[[i]],z[[i]])==TRUE)
    r<-r+1
}
length(y)/r
#setting up training and testing datasets
id <- sample(x = seq(1, dim(asia)[1], 1),
                        size = dim(asia)[1]*0.8,
                        replace = FALSE)
asia.train <- asia[id,]
asia.test <- asia[-id,]

bnprediction = as.numeric()
#fitting a model using Hill Climbing
bnmodel <- hc(asia.train,score = "bde",restart = 50)
bnmodelfit <- bn.fit(bnmodel,asia.train,method = 'mle') #max liklihood estimation
bngrain <- as.grain(bnmodelfit)
comp <- compile(bngrain)

bnmodel_true <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
bnmodelfit_true <- bn.fit(bnmodel_true, asia.train)
bngrain_true <- as.grain(bnmodelfit_true)
comp_true <- compile(bngrain_true)

bnpredict <- function(bntree, data, obs_variables, pred_variable) {
   for (i in 1:dim(data)[1]) {
    x <- NULL
    for (j in obs_variables) {
      x[j] <- if(data[i, j] == "yes") "yes" else "no"
    }
    evidence <- setEvidence(object = bntree,nodes = obs_variables,states=x)
    prob_dist_fit  <- querygrain(object = evidence,nodes = pred_variable)$S
```

```r
    bnprediction[i] <- if (prob_dist_fit["yes"] >= 0.5) "yes" else "no"
  }
  return(bnprediction)
}

# Predict S from Bayesian Network and test data observations
bnprediction <- bnpredict(bntree = comp,
                                            data = asia.test,
                                            obs_variables = c("A", "T", "L", "B", "E", "X", "D"),
                                            pred_variable = c("S"))
bnprediction_true <- bnpredict(bntree = comp_true,
                                                 data = asia.test,
                                                 obs_variables = c("A", "T", "L", "B", "E", "X", "D"),
                                                 pred_variable = c("S"))

# Calculate confusion matricies
confusion_matrix_fit <- table(bnprediction, asia.test$S)
print(confusion_matrix_fit)
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit))/sum(confusion_matrix_fit)))

confusion_matrix_true <- table(bnprediction_true, asia.test$S)
print(confusion_matrix_true)
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_true))/sum(confusion_matrix_true)))

par(mfrow=c(1,2))
plot(bnmodel)
title("hill climb network")
plot(bnmodel_true)
title("True network")
par(mfrow=c(1,1))
markov_blanket = mb(bnmodel, c("S"))
markov_blanket_true = mb(bnmodel_true, c("S"))
bnpredict_mb <- bnpredict(comp, asia.test, markov_blanket, c("S"))
bnpredict_mb_true <- bnpredict(comp_true, asia.test, markov_blanket_true, c("S"))
confusion_matrix_fit <- table(bnpredict_mb, asia.test$S)
confusion_matrix_fit_true <- table(bnpredict_mb_true, asia.test$S)
print(confusion_matrix_fit)
print(confusion_matrix_fit_true)
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit))/sum(confusion_matrix_fit)))
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_fit_true))/sum(confusion_matrix_fit_
naiveBayes = model2network("[S][A|S][B|S][D|S][E|S][X|S][L|S][T|S]")
plot(naiveBayes, main = "Naive Bayes")

naiveBayes.fit <- bn.fit(naiveBayes, asia.train)
result_naive <- bnpredict(compile(as.grain(naiveBayes.fit)), asia.test, c("A", "T", "L", "B", "E", "X",
# Calculate confusion matricies
confusion_matrix_naive_bayes <- table(result_naive, asia.test$S)
print(confusion_matrix_naive_bayes)
print(paste("Misclassification rate:", 1-sum(diag(confusion_matrix_naive_bayes))/sum(confusion_matrix_na
```