Lab1 Group Report

2019-09-21

Dependencies

```
library(bnlearn)
library(gRain)
library(caret)
library(e1071)
data(asia)
```

Contributions

1. omkbh878
2. henan076
3. rubmu773
4. musab250
5. obaur539

Exercise 1

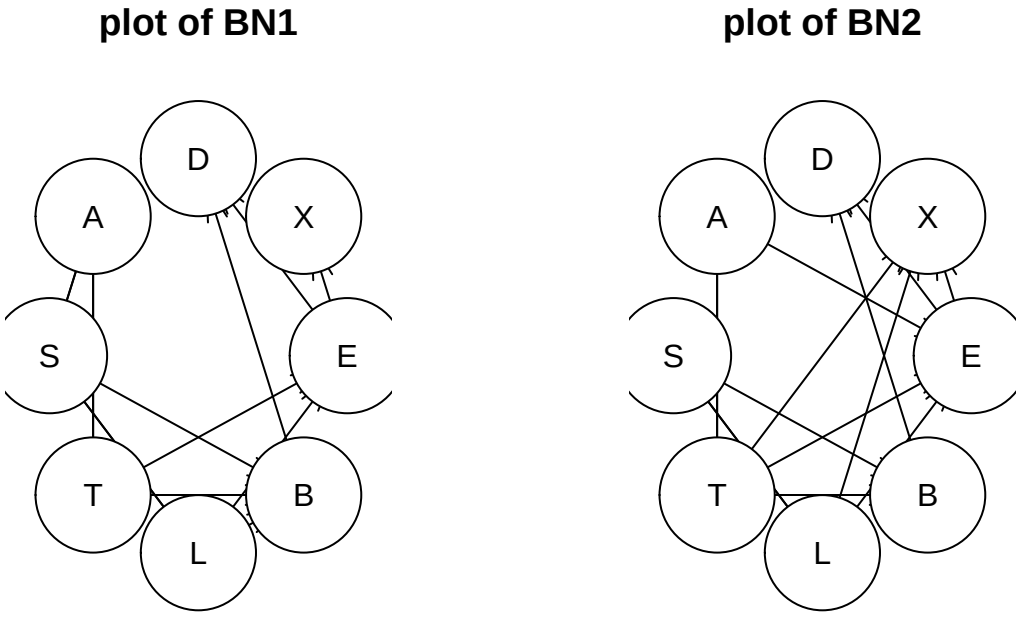
```
asia <- as.data.frame(asia)
#default parameter run of HC
hillClimbingResults = hc(asia)
print(hillClimbingResults)
```

```
##
##   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
```

```
par(mfrow = c(1,2))
hillclimb <- function(x){
  hc1 <- hc(x,restart = 5,score = "aic", iss = 3)
  hc2 <- hc(x,restart = 50,score = "bde", iss = 5)
  hc1dag <- cpdag(hc1)
  plot(hc1dag, main = "plot of BN1")
  hc1arcs <- vstructs(hc1dag, arcs = TRUE)
```

```
  arcs(hc1)
  hc2dag <- cpdag(hc2)
  plot(hc2dag, main = "plot of BN2")
  hc2arcs <- vstructs(hc2dag, arcs = TRUE)
  arcs(hc2)
  print(all.equal(hc1,hc2))
}

hillclimb(asia)
```




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

As we can see in results above, the runs result in different graph structures.

Exercise 2

The network structure is learned using the hill climbing algorithm as before, using BIC score. The parameters of the network are learned using `bn.fit` which learns the parameters using Maximum Likelihood parameter estimation. To predict S in the test data we set the hard evidence in the network to the values of the test data excluding S using `setEvidence`. Then we query the probability of S, from the network with hard evidence applied, using `querygrain`. We pick the classification with highest probability and compare to the true value in the test data using a confusion matrix.

```
data("asia")
sample_size <- floor(0.8 * nrow(asia))
train_ix <- sample(nrow(asia), size = sample_size)
train.asia <- asia[train_ix,]
```

```
test.asia <- asia[-train_ix,]

bn.asia <- hc(train.asia)
bn.asia.fit <- bn.fit(bn.asia, train.asia)
grain.asia <- as.grain(bn.asia.fit)

predict_S <- function(grain.network, data.asia, preds) {
  e_cs <- colnames(asia)[preds] # classes excluding S
  no_yes <- factor(c("no", "yes")) # factors for converting back to no, yes
  predict_row <- function(r) no_yes[which.max(querygrain(
    setEvidence(grain.network, e_cs, r[preds]), c("S"))$S)]
  apply(data.asia, 1, predict_row)
}

accuracy <- function(t) (t[1,1] + t[2,2]) / sum(t)

print_t <- function(t) {
  print(t)
  print(paste("Accuracy:", accuracy(t)))
}

res <- predict_S(grain.asia, test.asia, -2)
print_t(table(test.asia[,2], res))
```

```
##      res
##        no yes
##   no  325 161
##   yes 122 392
## [1] "Accuracy: 0.717"
```

We redo the procedure using the true network structure (instead of learning it using hill climbing) and output the corresponding confusion matrix.

```
bn.asia_true <- model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
bn.asia_true.fit <- bn.fit(bn.asia_true, train.asia)
grain.asia_true <- as.grain(bn.asia_true.fit)

res_true <- predict_S(grain.asia_true, test.asia, -2)
print_t(table(test.asia[,2], res_true))
```

```
##      res_true
##        no yes
##   no  325 161
##   yes 122 392
## [1] "Accuracy: 0.717"
```

The results show that learned structure performs equally well to the true structure despite them not being equal. However if we look at the the markov blankets for S for both networks we see that they are the same, meaning the the networks are equivalent given values for all random variables in the markov blanket.

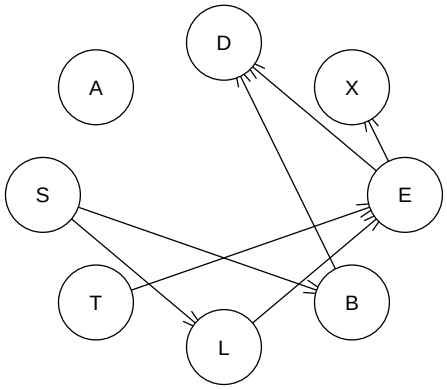
```
all.equal(bn.asia, bn.asia_true)
```

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

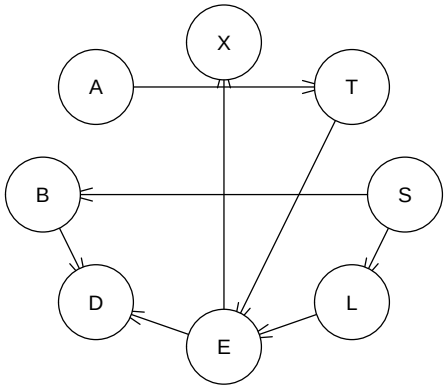
```
sort(mb(bn.asia, "S")) == sort(mb(bn.asia_true, "S"))
```

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

```r
plot(bn.asia)
```




```r
plot(bn.asia_true)
```



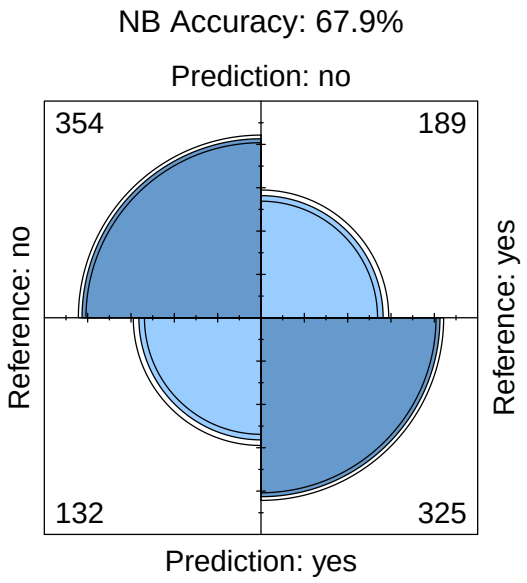
Exercise 3

The figure below shows the results of the inference of S using a markov chain. This is nice indeed, because the total dimentions used are only L, B, while with the previous method are A, T, L, B, E, X, D. The accuracy with the given data stays and the complexity of the model is greatly reduced.

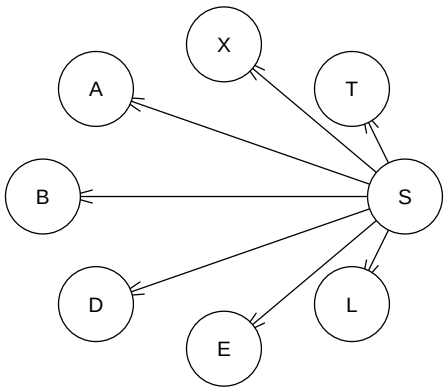
```r
charAsia <- data.frame(lapply(asia, as.character), stringsAsFactors = FALSE)[-train_ix,]
bnAsiaN <- model2network("[S][A|S][B|S][D|S][E|S][X|S][L|S][T|S]")
bnAsiaNFit <- bn.fit(bnAsiaN, train.asia)
grainAsiaN <- as.grain(bnAsiaNFit)
```

```r
# test predicting S
predS <- NULL
for (i in 1:length(test.asia[, 1])) {
  evid <- setEvidence(grainAsiaN, nodes = colnames(test.asia[-2]),
                      states = charAsia[i, -2])
  quer <- querygrain(evid, nodes = c("S"), type = "marginal")
  predS <- c(predS, ifelse(quer$S[1] > quer$S[2], "no", "yes"))
}

predAccS <- confusionMatrix(data=as.factor(predS), reference=test.asia[,2])
fourfoldplot(predAccS$table,
             main=paste0('NB Accuracy: ',
                         round(predAccS[['overall']][['Accuracy']]*100, 3),'%'))
```




```r
plot(bnAsiaN)
```



Exercise 4

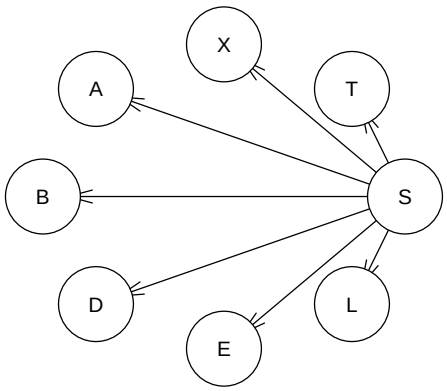
```
naiveBN <- model2network("[S][A|S][B|S][D|S][E|S][L|S][T|S][X|S]")

## Re-use predict_S instead of introducing new similar function
predictedNaiveS <- predict_S(as.grain(bn.fit(naiveBN, train.asia)), test.asia, -2)

confMatrixNaiveB <- table(predictedNaiveS, test.asia$S)
print(confMatrixNaiveB)
```

```
##
## predictedNaiveS  no yes
##             no  354 189
##             yes 132 325
```

```
plot(naiveBN)
```



Exercise 5

The hill climbing algorithm resulted in the structure using the BIC score. We used this structure with all the predictors and with some (in case of Markov blanket), it gave same results. However, with naive bayes, the results were bit different. The reduced predictors from markov blanket is the evidence that a proper subset which is the representative of a whole graph can be used instead of using all of the graph (Although with some information lost during the reduciton process). The naive bayes results in a different structure and hence the probability distribution gets changed and therefore produces results different to that of other models. Naive bayes (in this case) is not a suitable candidate.