Advanced Machine Learning

Lab 1

Rasmus Holm

2017-09-11

Assignment 1

```
graph_equality <- function(g1, g2) {
    all.equal(cpdag(g1), cpdag(g2)) == TRUE
}

print("No restarts")
#> [1] "No restarts"
sapply(1:5, function(i) {
    g1 <- hc(alarm, score="bde", iss=1, restart=0)
    g2 <- hc(alarm, score="bde", iss=1, restart=0)
    graph_equality(g1, g2)
})
#> [1] TRUE TRUE TRUE TRUE TRUE

print("Restarts with equivalent seed")
#> [1] "Restarts with equivalent seed"
sapply(1:5, function(i) {
    set.seed(i)
    g1 <- hc(alarm, score="bde", iss=1, restart=10)
    set.seed(i)
    g2 <- hc(alarm, score="bde", iss=1, restart=10)
    graph_equality(g1, g2)
})
#> [1] TRUE TRUE TRUE TRUE TRUE

print("Restarts with distinct seed")
#> [1] "Restarts with distinct seed"
sapply(1:5, function(i) {
    set.seed(i)
    g1 <- hc(alarm, score="bde", iss=1, restart=10)
    set.seed(2 * i)
    g2 <- hc(alarm, score="bde", iss=1, restart=10)
    graph_equality(g1, g2)
})
#> [1]  TRUE FALSE FALSE FALSE FALSE
```

The hill climbing algorithm is completely deterministic so unless we do random restarts the result is the same. The random restarts means that we begin the algorithm with different initial graph structures which may possibly end up in different local optima.

Assignment 2

```
data <- asia

g1 <- hc(data, score="bde", iss=1, restart=10)
g10 <- hc(data, score="bde", iss=10, restart=10)
g100 <- hc(data, score="bde", iss=100, restart=10)
g1000 <- hc(data, score="bde", iss=1000, restart=10)

bnlearn::score(g1, data, type="bde")
#> [1] -11095.79
bnlearn::score(g10, data, type="bde")
#> [1] -11132.15
bnlearn::score(g100, data, type="bde")
#> [1] -11467.13
bnlearn::score(g1000, data, type="bde")
#> [1] -13301.65
```

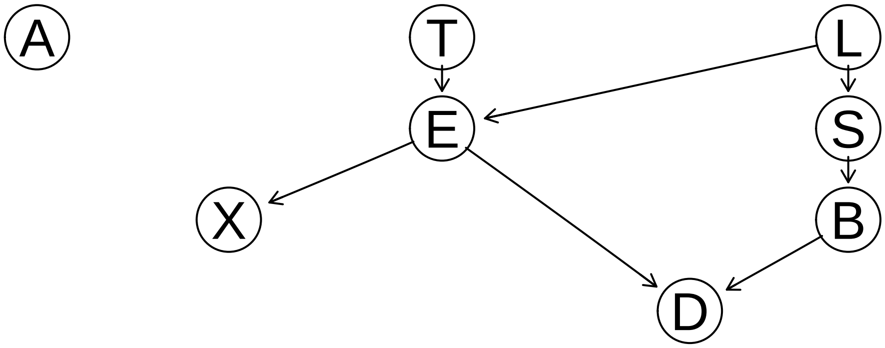
In the BDe (*Bayesian Dirichlet equivalent uniform*) score we assume a priori that the probabilities follow a Dirichlet(α) where α is the *imaginary sample size* (iss) and the posterior

$$P(A|Data) = \frac{iss}{n + iss} P_{\text{prior}}(A) + \frac{n}{n + iss} P_{\text{empirical}}(A),$$

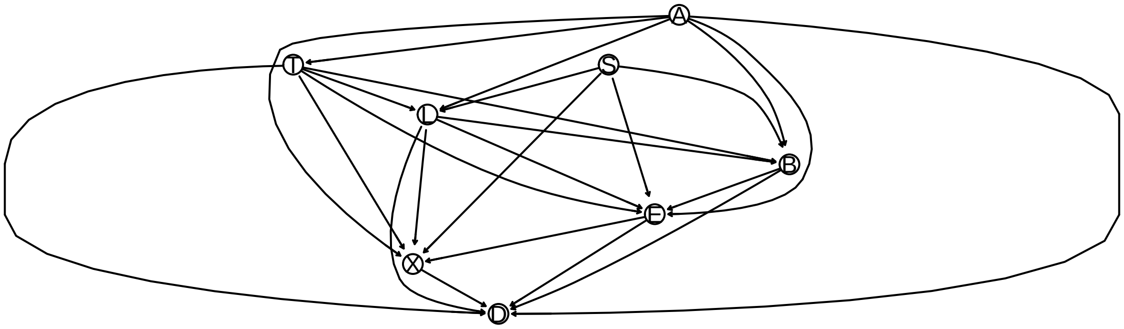
where n is the number of observations in the data. This means that the iss controls how certain we are a priori of the probabilities indicating that having a high iss means the data have less influence on the posterior distribution. Since the Dirichlet is chosen such that it is uniform a higher iss result in more edges in the graph, therefore iss can be seen as a regularization term. A low value entails a sparse graph, i.e. less arcs/dependencies, inferred from data assuming the true distribution is represented by a sparse graph.

This can clearly be shown by the plot below. Given that there are 8 nodes in the graph we would expect the average branching factor to be around $7/2 = 3.5$ with a uniform prior and we get 3 with a imaginary sample size of 1000 which is close. Bayesian networks have certain constraints that prevent edges from being added arbitrarly.

iss=1

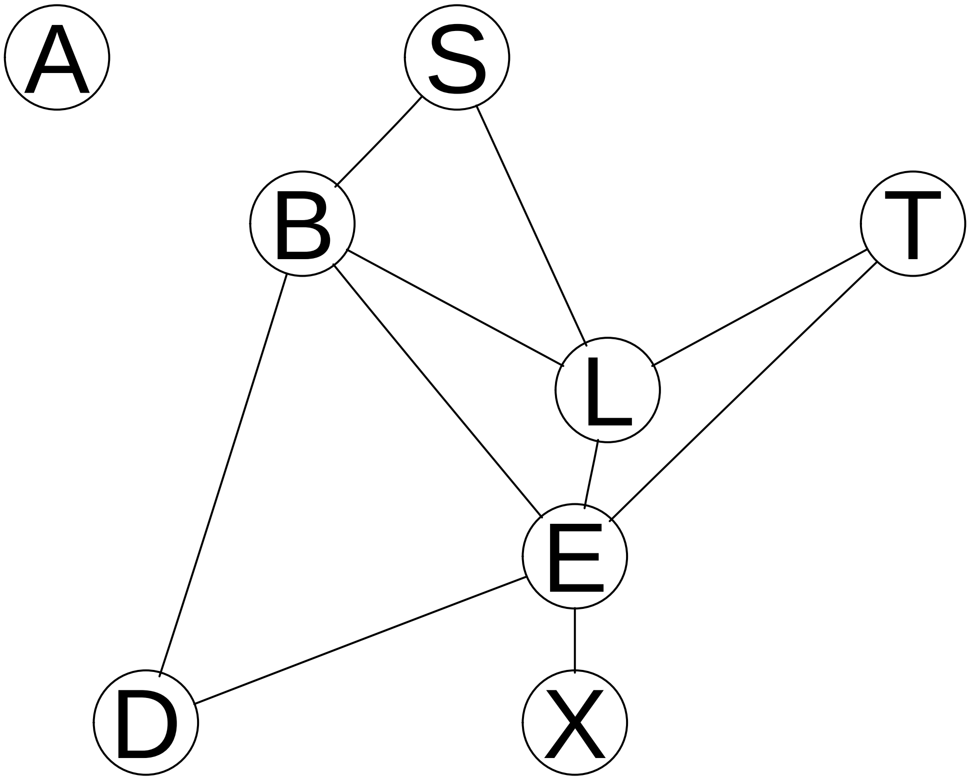


iss=1000



Assignment 3

```
data <- asia
graph <- hc(data, score="bde", iss=1, restart=10)
bayes_net <- bn.fit(graph, data, method="bayes", iss=1)
junction_tree <- compile(as.grain(bayes_net))
```



```
## Exact inference
querygrain(junction_tree, nodes=c("B"), type="marginal")
#> $B
#> B
#>       no      yes 
#> 0.490202 0.509798 

## Approximate inference
dist <- cpdist(fitted=bayes_net, nodes=c("B"), evidence=TRUE)
prop.table(table(dist))
#> dist
#>     no    yes 
#> 0.4878 0.5122 

dist <- cpdist(fitted=bayes_net, nodes=c("B"), evidence=TRUE, method="lw")
prop.table(table(dist))
#> dist
#>     no    yes 
#> 0.4846 0.5154 
```

```
## Exact inference
querygrain(junction_tree, nodes=c("L", "T"), type="joint")
#>      T
#> L             no          yes
#>   no  0.92560305 0.0083101656
#>   yes 0.06549873 0.0005880548

## Approximate inference
dist <- cpdist(fitted=bayes_net, nodes=c("L", "T"), evidence=TRUE)
prop.table(table(dist))
#>      T
#> L         no    yes
#>   no  0.9286 0.0090
#>   yes 0.0622 0.0002

dist <- cpdist(fitted=bayes_net, nodes=c("L", "T"), evidence=TRUE, method="lw")
prop.table(table(dist))
#>      T
#> L         no    yes
#>   no  0.9226 0.0072
#>   yes 0.0694 0.0008

## Exact Inference
querygrain(setEvidence(junction_tree, nodes="E", states="yes"),
           nodes=c("L", "T"), type="joint")
#>      T
#> L               no         yes
#>   no  0.0003360683 0.111418174
#>   yes 0.8805694921 0.007676265

## Approximate inference
dist <- cpdist(fitted=bayes_net, nodes=c("L", "T"), evidence=(E == "yes"))
prop.table(table(dist))
#>      T
#> L              no         yes
#>   no  0.000000000 0.111111111
#>   yes 0.883468835 0.005420054

## Exact Inference
querygrain(junction_tree, nodes=c("D", "B", "E"), type="conditional")
#> , , D = no
#> 
#>      E
#> B            no       yes
#>   no  0.9001296 0.2777778
#>   yes 0.2137615 0.1463023
#> 
#> , , D = yes
#> 
#>      E
#> B             no       yes
#>   no  0.09987037 0.7222222
#>   yes 0.78623853 0.8536977
```

```
## Approximate inference
dist <- cpdist(bayes_net, nodes="D", evidence=(B=="yes") & (E=="yes"), n=10^6)
prop.table(table(dist))
#> dist
#>        no       yes
#> 0.1478338 0.8521662

dist <- cpdist(bayes_net, nodes="D", evidence=(B=="yes") & (E=="no"))
prop.table(table(dist))
#> dist
#>        no       yes
#> 0.2231441 0.7768559

dist <- cpdist(bayes_net, nodes="D", evidence=(B=="no") & (E=="yes"), n=10^6)
prop.table(table(dist))
#> dist
#>       no      yes
#> 0.272911 0.727089

dist <- cpdist(bayes_net, nodes="D", evidence=(B=="no") & (E=="no"))
prop.table(table(dist))
#> dist
#>        no       yes
#> 0.8959862 0.1040138
```

In the approximate inference methods we use simulated data to infer the queries and the samples are random so the inference will also be random. When the query includes observed nodes it seems that the approximate inference algorithm require more samples to be reasonably accurate than for queries without any observed nodes. This is probabily due to the sampling process is sampling from the full joint distribution rather than the conditional distribution. This means that not all samples fullfil the conditional so the actual sample size is far less than specified. This is evident from the following code block.

```
## No observed nodes
number_of_samples <- 10^6
dist <- cpdist(bayes_net, nodes="D", evidence=TRUE, n=number_of_samples)
number_of_actual_samples <- nrow(dist)
sprintf("Number of samples to generate: %i, Number of samples returned: %i",
        number_of_samples, number_of_actual_samples)
#> [1] "Number of samples to generate: 1000000, Number of samples returned: 1000000"

## Two observed nodes
number_of_samples <- 10^6
dist <- cpdist(bayes_net, nodes="D", evidence=(B=="no") & (E=="yes"), n=number_of_samples)
number_of_actual_samples <- nrow(dist)
sprintf("Number of samples to generate: %i, Number of samples returned: %i",
        number_of_samples, number_of_actual_samples)
#> [1] "Number of samples to generate: 1000000, Number of samples returned: 25685"
```

Assignment 4

```r
n <- 10000
rgraphs <- unique(random.graph(nodes=c("1", "2", "3", "4", "5"), num=n,
                               method="ic-dag", burn.in=500, every=10))
length(unique(lapply(rgraphs, cpdag))) / length(rgraphs)
#> [1] 0.5537917
```

From the evidence we would reduce the search space by approximately 45% which is a lot relative to how many DAGs there are as the number of nodes increases. However, I imagine it is more difficult to work with the essential graphs due to containing both directed and undirected arcs. Depending on the increased complexity of the algorithm working in the essential graph space could potentially be appropriate for structure learning.