Advanced Machine Learning: Lab 1 - Individual Report

Shashi Nagarajan (shana299)

```r
library(bnlearn)
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```r
data("asia")
set.seed(42)
```

1

We conduct two runs of the hill-climbing algorithm, each run seeded by a different, randomly initialised DAG. From the results shown below, we can observe that the two runs produce different outputs, thus solving the problem.

The reason we observe two different outputs below is that without exploring all possible initialisations, the hill-climbing algorithm is only guaranteed to converge at a local minimum in finite time. What we observe below are results corresponding to two different local minima, seeded by two different initialisations.

```r
r1 = random.graph(colnames(asia))
r2 = random.graph(colnames(asia))

while (all.equal(cpdag(r1), cpdag(r2)) == T) {

  r2 = random.graph(colnames(asia)) # ensure r2 is different from r1

}

hc1 = hc(asia, r1)
hc2 = hc(asia, r2)

# Check for equivalence of the results of the two runs

flag = F

if (all.equal(cpdag(hc1), cpdag(hc2)) != T) {

  print("The two runs produced graphs with different adjacencies")

} else {

  hc1_uc <- vstructs(hc1, arcs = T)
  hc2_uc <- vstructs(hc2, arcs = T)
  hc1_uc_from <- hc1_uc[, 1]
  hc1_uc_to <- hc1_uc[, 2]
  hc2_uc_from <- hc2_uc[, 1]
  hc2_uc_to <- hc2_uc[, 2]

  if (nrow(hc1_uc) != nrow(hc2_uc)) {

    print("The two runs produced graph with different unshielded colliders")

  } else {

    for (i in 1:nrow(hc1_uc)) {

      match_list = which(hc2_uc_from %in% hc1_uc_from[i])

      if (length(match_list) == 0) {

        flag = T
        break

      } else {

        for (j in match_list) {

          if (hc2_uc_to[j] != hc1_uc_to[i]) {

            flag = T
            break
          }

        }

      }

      if (flag) {break}

    }

  }

}
```

```
## [1] "The two runs produced graphs with different adjacencies"
```

```r
if (flag) {print("The two runs produced graph with different unshielded colliders")}
```

2

Below is the confusion matrix and related statistics, corresponding to the classifier trained on a random sample consisting of 80% of the Asia dataset; its parameters were estimated using through MLE, which in turn was applied on a structure learned through the hill-climbing algorithm. Note that throughout this report, approximate inference via the function 'cpquery' has been used for classification tasks.

```r
train_ind = sample(1:nrow(asia), floor(0.8*nrow(asia)))
test_ind = setdiff(1:nrow(asia), train_ind)

train_data = asia[train_ind, ]
```

```r
test_data = asia[test_ind, ]

hc_train = hc(train_data)
# in case result has undirected arcs, force directions; choice of ordering arbitrary
hc_train_dag = pdag2dag(hc_train, ordering = colnames(asia))
params_hc_train = bn.fit(hc_train_dag, train_data)

pred_S <- function(x, params) { # assume x is named

  evidence_text = paste(
    "(A == \"", x$A, "\") & ",
    "(T == \"", x$T, "\") & ",
    "(L == \"", x$L, "\") & ",
    "(B == \"", x$B, "\") & ",
    "(E == \"", x$E, "\")",
    sep = "")
  query_text = paste("cpquery(params, (S == \"yes\"), ", evidence_text, ")", sep = "")

  return(ifelse(eval(parse(text = query_text)) > 0.5, "yes", "no"))

}

preds = sapply(1:nrow(test_data), function(i) pred_S(test_data[i, ], params_hc_train))
cat("Confusion Matrix:")
```

```
## Confusion Matrix:
```

```r
print(confusionMatrix(as.factor(preds), test_data$S))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  365 130
##        yes 155 350
##
##                Accuracy : 0.715
##                  95% CI : (0.6859, 0.7428)
##     No Information Rate : 0.52
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4302
##
##  Mcnemar's Test P-Value : 0.1551
##
##             Sensitivity : 0.7019
##             Specificity : 0.7292
##          Pos Pred Value : 0.7374
##          Neg Pred Value : 0.6931
##              Prevalence : 0.5200
##          Detection Rate : 0.3650
##    Detection Prevalence : 0.4950
##       Balanced Accuracy : 0.7155
##
```

```
##        'Positive' Class : no
##
```

For benchmarking, we also show below the confusion matrix arising from the classifier learned from MLE using the same training data but the true Asia BN structure.

We see that classification accuracy is quite similar in both cases. If one assumes that the true BN is the best in terms of representing the relationships between the different variables observed, one may infer that in the run shown above, the hill-climbing algorithm has been able to successfully learn the important structural features of the true graph that are necessary to predict S, with the available data; we may also conclude that MLE has estimated parameters well given the volume of data available.

```
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
params_tg_est = bn.fit(dag, train_data)
preds_tg = sapply(1:nrow(test_data), function(i) pred_S(test_data[i, ], params_tg_est))
confusionMatrix(as.factor(preds_tg), test_data$S)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  365 130
##        yes 155 350
##
##                Accuracy : 0.715
##                  95% CI : (0.6859, 0.7428)
##     No Information Rate : 0.52
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4302
##
##  Mcnemar's Test P-Value : 0.1551
##
##             Sensitivity : 0.7019
##             Specificity : 0.7292
##          Pos Pred Value : 0.7374
##          Neg Pred Value : 0.6931
##              Prevalence : 0.5200
##          Detection Rate : 0.3650
##    Detection Prevalence : 0.4950
##       Balanced Accuracy : 0.7155
##
##        'Positive' Class : no
##
```

3

Shown below is my implementation of the requested classification model based on the Markov blanket of S.

```
pred_S_et <- function(x, params, evidence_terms) {

  num_terms = length(evidence_terms)
  evidence_term_ind = match(evidence_terms, colnames(asia))

  evidence_text = paste("(", evidence_terms[1], " == \"", x[1, evidence_term_ind[1]],
                        "\")", sep = "")
```

```r
  for (i in 2:num_terms) {

    append_text = paste("(", evidence_terms[i], " == \"", x[1, evidence_term_ind[i]],
                        "\")", sep = "")
    evidence_text = paste(evidence_text, " & ", append_text)

  }

  query_text = paste("cpquery(params, (S == \"yes\"), ", evidence_text, ")", sep = "")

  return(ifelse(eval(parse(text = query_text)) > 0.5, "yes", "no"))

}

preds_mb = sapply(1:nrow(test_data), function(i)
  pred_S_et(test_data[i, ], params_hc_train, mb(hc_train_dag, "S")))
confusionMatrix(as.factor(preds_mb), test_data$S)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  365 129
##        yes 155 351
##
##                Accuracy : 0.716
##                  95% CI : (0.6869, 0.7438)
##     No Information Rate : 0.52
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4323
##
##  Mcnemar's Test P-Value : 0.1379
##
##             Sensitivity : 0.7019
##             Specificity : 0.7312
##          Pos Pred Value : 0.7389
##          Neg Pred Value : 0.6937
##              Prevalence : 0.5200
##          Detection Rate : 0.3650
##    Detection Prevalence : 0.4940
##       Balanced Accuracy : 0.7166
##
##        'Positive' Class : no
##
```

4

Shown below is my implementation of the requested Naive Bayes classification model.

```r
nb = empty.graph(colnames(asia))
arc.set = cbind(rep("S", ncol(asia) - 1), colnames(asia)[colnames(asia) != "S"])
colnames(arc.set) = c("from", "to")
arcs(nb) = arc.set
```

```r
params_nb_train = bn.fit(nb, train_data)

preds_nb = sapply(1:nrow(test_data), function(i) pred_S(test_data[i, ], params_nb_train))
confusionMatrix(as.factor(preds_nb), test_data$S)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  365 130
##        yes 155 350
##
##                Accuracy : 0.715
##                  95% CI : (0.6859, 0.7428)
##     No Information Rate : 0.52
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4302
##
##  Mcnemar's Test P-Value : 0.1551
##
##             Sensitivity : 0.7019
##             Specificity : 0.7292
##          Pos Pred Value : 0.7374
##          Neg Pred Value : 0.6931
##              Prevalence : 0.5200
##          Detection Rate : 0.3650
##    Detection Prevalence : 0.4950
##       Balanced Accuracy : 0.7155
##
##        'Positive' Class : no
##
```

5

Results obtained in tasks 2-4 are very similar. In task 2, two results were obtained, both very similar to one another; this observation has already been explained above. The remaining job is to explain why results in Tasks 3 and 4 are similar to that seen in Task 2.

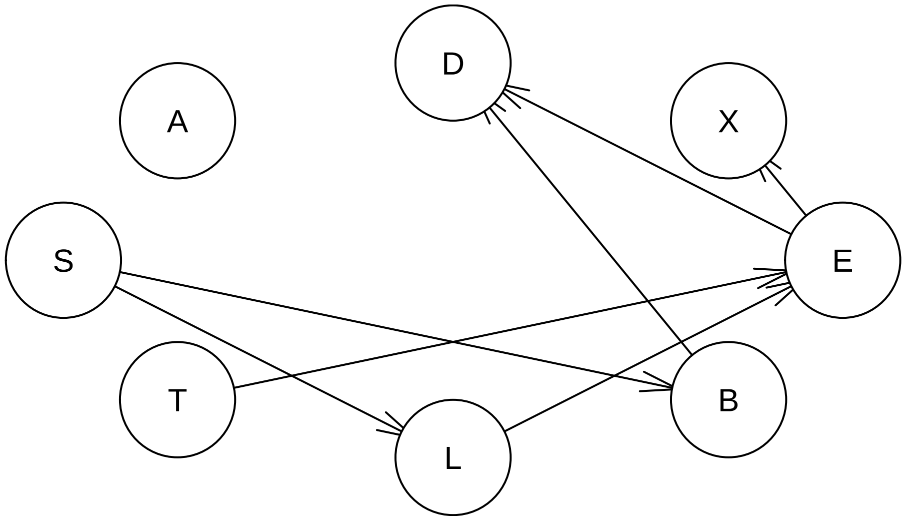
Task 2 vs Task 3

In the true causal BN corresponding to the Asia dataset, we see that $S \perp_G \{A, D, E, T, X\} \,|L, B$. This must mean that $S \perp_p \{A, D, E, T, X\} \,|L, B$ Furthermore, the Markov blanket of S is precisely {L, B} in the DAG obtained from the 80%-random-sample of the Asia dataset. Thus, given the said blanket, other variables serve no purpose, causing result in Tasks 2 and 3 to output nearly identical results (NB: approximate inference used for classification)

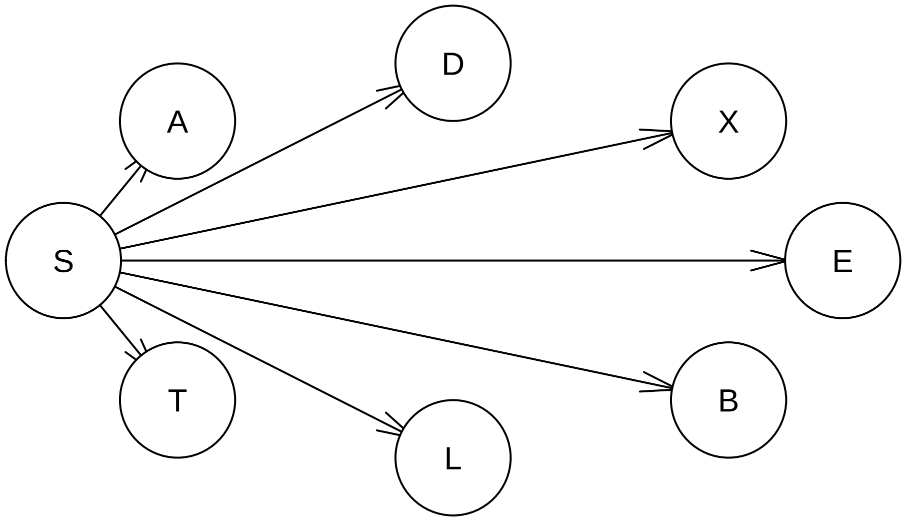
Task 2 vs Task 4

To compare the results of Tasks 2 and 4, we can refer to the graphs learned in the two tasks.

```r
plot(hc_train) # graph learned in task 2
```



```r
plot(nb) # naive-bayes graph
```



Based on the graphs obtained, it is hard to reason why we obtained the same results in Tasks 2 and 4; it could be that errors on account of approximate inference and/or the specifications of data that we happened to observe are making it appear as though the results are identical though one would not exactly expect it.

In fact, we would not, in general, expect the Naive Bayes algorithm to return similar output vis-a-vis the generic probabilistic-graphical method; we would expect the latter to perform at least as well as the Naive Bayes algorithm, however, since Naive Bayes imposes some independence constraints that may not be warranted prior to observing data.