Advanced Machine Learning (732A96) - Lab 01

Hector Plata (hecpl268)

Contents

Task 1

Show that multiple runs of the hill-climbing algorithm can return non-equivalent Bayesian network (BN) structures. Explain why this happens. Use the Asia dataset which is included in the `bnlearn` package. To load the data, run data("asia").

Hint: Check the function `hc` in the `bnlearn` package. Note that you can specify the initial structure, the numbeer of random restarts, the score, and the equivalent sample size (a.k.a imaginary sample size) in the BDeu score. You may want to use these options to answer the question. You may also want to use the functions `plot`, `arcs`, `vstructs`, `cpdag` and `all.equal`.

Answer

We see below that multiple runs of the hill-climbing algorithm can return non-equivalent Bayesian networks structures. Since learning the best structure of a Bayesian network is intractable, `hc` uses a greedy search to obtain a "good" result. Thus the optimization problem:

$$\arg\max_{BN} score(BN, \theta)$$

where,

$$BN: \text{ Final Bayesian Network}$$

$$\theta: \text{ initial parameters}$$

could yield different BN given different initial parameters θ since they can potentially end up in different local optima. In this case, θ differs only on the starting graph for the `hc` algorithm and it already outputs two different BN.

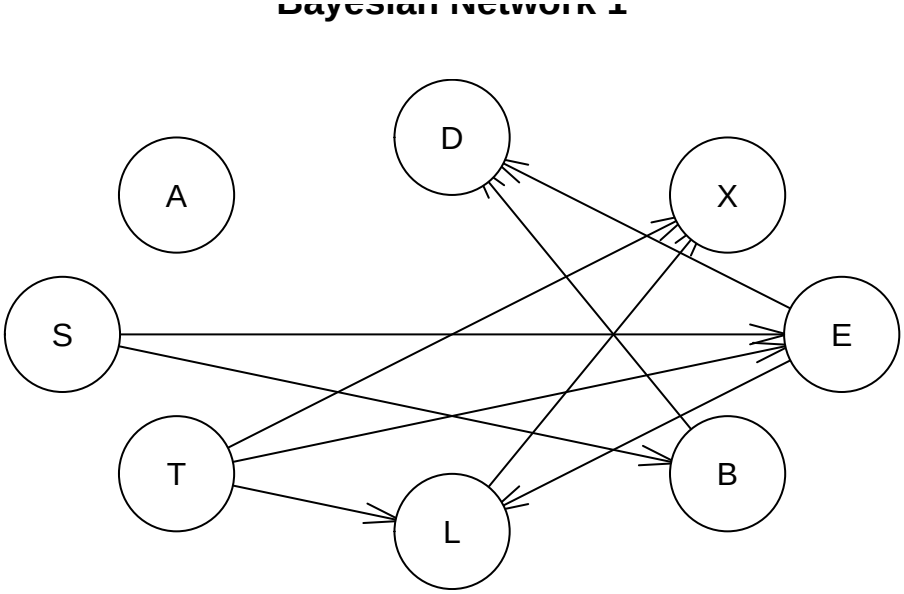
```
library(bnlearn)
```

```
##
## Attaching package: 'bnlearn'
```

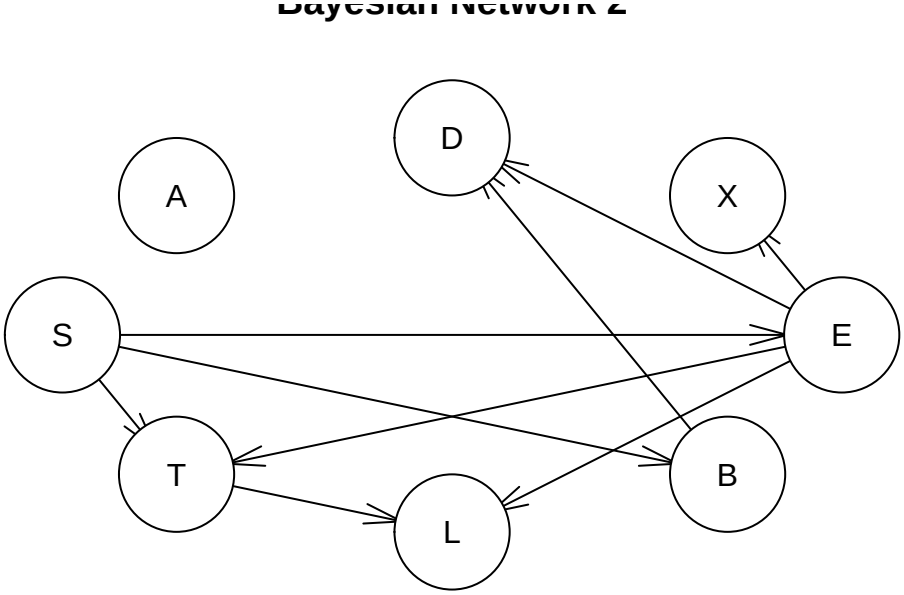
```
## The following object is masked from 'package:stats':
##
##     sigma
```

```r
# Loading the data.
data("asia")

# Creating two random initial graphs.
random_graph_01 = random.graph(colnames(asia))
random_graph_02 = random.graph(colnames(asia))

bn_1 = hc(asia, start=random_graph_01)
bn_2 = hc(asia, start=random_graph_02)
```

```r
plot(bn_1, main="Bayesian Network 1")
```




```r
plot(bn_2, main="Bayesian Network 2")
```



Task 2

Learn a BN from 80% of the Asia dataset. The dataset is included in the `bnlearn` package. To load the data, run `data("asia")`. Learn both the structure and the parameters. Use any learning algorithm and settings that you consider appropriate. Use the BN learned to classify the remaining 20% of the Asia dataset in two classes: $S = yes$ and $S = no$. In other words, compute the posterior probability distribution of S for each case and classify it in the most likely class. To do so, you **have** to use exact or approximate inference with the help of the `bnlearn` and `gRain` packages, i.e. you are not allowed to use functions such as `predict`. Report the confusion matrix, i.e. true/false positives/negatives. Compare your results with those of the true Asian BN, which can be obtained by running `dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")`.

Hint: You already know the Lauritzen-Spiegelhalter algorithm for inference in BNs, which is an exact algorithm. There are also approximate algorithms for when the exact ones are too demanding computationally. For exact inference, you may need the functions `bn.fit` and `as.grain` from the `bnlearn` package, and the functions `compile`, `setFinding` and `querygrain` from the package `gRain`. For approximate inference, you may need the functions `prop.table`, `table` and `cpdist` fomr the `bnlearn` package. When you try to load the package `gRain`, you will get an error as the package `RBGL` cannot be found. You have to install this package by running the following two commands (answer no to any offer to update packages):

```
source("https://bioconductor.org/biocLite.R") biocLite("RBGL")
```

Answer From the graph obtained from the data and the parameters of the node S we can conclude that all the predictions for every observation of the node S are going to be yes.

```
library(gRain)
```

```
## Loading required package: gRbase
```

```
##
## Attaching package: 'gRbase'
```

```
## The following objects are masked from 'package:bnlearn':
##
##     ancestors, children, parents
```

```
library(caret)
```

```
## Loading required package: lattice
```

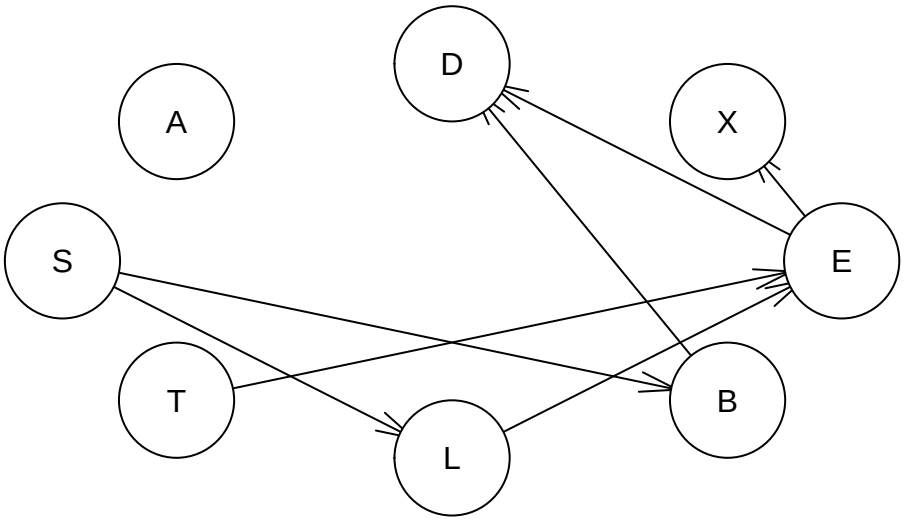
```
## Loading required package: ggplot2
```

```
# Auxiliary function to transform from numeric
# to characters.
number_to_char = function(x)
{
  if (x == 1)
  {
    return("no")
  }
  else
  {
    return("yes")
  }
}

# Function to do an exact prediction.
predict_bn = function(bn_fit, test, yhat_node, markov_blanket=FALSE)
{
  # Creating the junction tree.
```

```
  bn_grain = as.grain(bn_fit)
  junction = compile(bn_grain)

  # Getting the evidence.
  # Names on the information seen.
  if (markov_blanket == FALSE)
  {
    node_names = colnames(test)
    evidence_nodes = node_names[node_names != yhat_node]
  }
  else
  {
    evidence_nodes = mb(bn_fit, yhat_node)
  }

  # Values of the information seen.
  evidence_states = test[, evidence_nodes]

  # Variable that stores the predictions.
  preds = c()

  # Getting the predictions.
  for (i in 1:dim(test)[1])
  {
    # Evidence of the observation i.
    # Transforming the factors into characters because
    # that's what the evidence requires :).
    evidence_states_char = sapply(as.numeric(evidence_states[i, ]), number_to_char)
    evidence = setEvidence(junction, nodes=evidence_nodes, states=evidence_states_char)

    # Probability for the node S.
    p = querygrain(evidence)

    # Predicting
    if (p[[yhat_node]][2] >= 0.5)
    {
      preds = c(preds, "yes")
    }
    else
    {
      preds = c(preds, "no")
    }
  }

  return(preds)
}
```

```
# Prepare data
data("asia")
df = asia
rm(asia)

# Split data randomly
set.seed(12345)
```

```r
N = nrow(df)
idx = sample(1:N, round(N * 0.8))
train = df[idx, ]
test = df[-idx, ]

# Learning the structure and parameters of a BN.
bn = hc(train)
bn_fit = bn.fit(bn, train)
plot(bn)
```




```r
print(bn_fit$S)
```

```
##
##   Parameters of node S (multinomial distribution)
##
## Conditional probability table:
##     no   yes
## 0.493 0.507
```

Below we can see the confusion matrix for the learned structure and parameters.

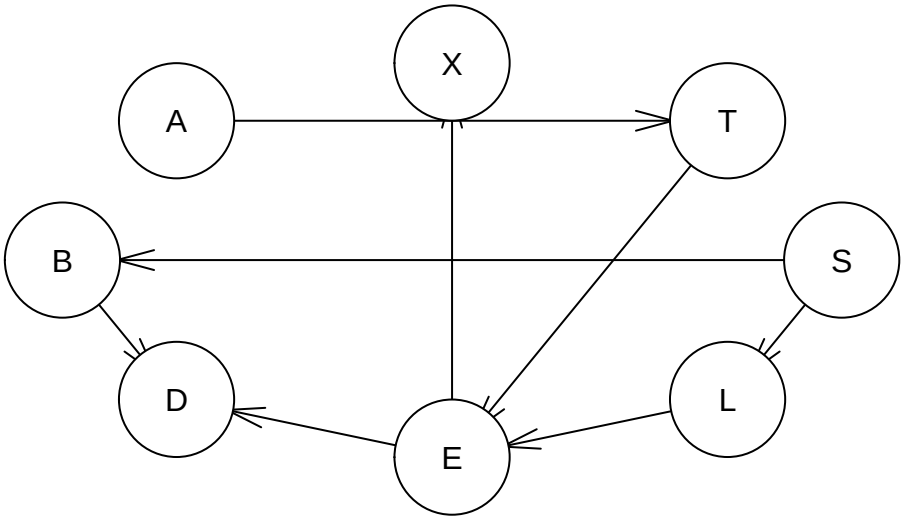
```r
yhat_node = "S"
y_hat = predict_bn(bn_fit, test, yhat_node)

cm = confusionMatrix(factor(y_hat, levels=c("no", "yes")), factor(test[, yhat_node]))
print(cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  337 121
##        yes 176 366
##
##                Accuracy : 0.703
##                  95% CI : (0.6736, 0.7312)
##     No Information Rate : 0.513
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                   Kappa : 0.4073
##
##  Mcnemar's Test P-Value : 0.001728
##
##             Sensitivity : 0.6569
##             Specificity : 0.7515
##          Pos Pred Value : 0.7358
##          Neg Pred Value : 0.6753
##              Prevalence : 0.5130
##          Detection Rate : 0.3370
##    Detection Prevalence : 0.4580
##       Balanced Accuracy : 0.7042
##
##        'Positive' Class : no
##
```

Now we are going to do the same but for the true Asian BN.

```
# Creating the true model.
bn_true = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
plot(bn_true)
```




```
# Fitting the network and getting
# the junction tree.
bn_true_fit = bn.fit(bn_true, train)

# Getting the predictions for the true network
# and printing the confusion matrix.
y_hat_true = predict_bn(bn_true_fit, test, yhat_node)
cm = confusionMatrix(factor(y_hat_true, levels=c("no", "yes")), factor(test[, yhat_node]))
print(cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  337 121
##        yes 176 366
##
```

```
##                Accuracy : 0.703
##                  95% CI : (0.6736, 0.7312)
##     No Information Rate : 0.513
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4073
##
##  Mcnemar's Test P-Value : 0.001728
##
##             Sensitivity : 0.6569
##             Specificity : 0.7515
##          Pos Pred Value : 0.7358
##          Neg Pred Value : 0.6753
##              Prevalence : 0.5130
##          Detection Rate : 0.3370
##    Detection Prevalence : 0.4580
##       Balanced Accuracy : 0.7042
##
##        'Positive' Class : no
##
```

Task 3

In the previous exercise, you classified the variable S given observations for all the rest of the variables. Now, you are asked to classify S given observations only for the so-called Markov blanker of S, i.e. its parents plus its children plus the parents of its children minus S itself. Report again the confusion matrix.

Hint: You may want to use the function `mb` from the `bnlearn` package.

Answer

We get similar results on both.

```
# Getting the predictions of each BN.
y_hat_leart = predict_bn(bn_fit, test, yhat_node, TRUE)
y_hat_true = predict_bn(bn_true_fit, test, yhat_node, TRUE)


cm_learnt = confusionMatrix(factor(y_hat_leart, levels=c("no", "yes")), factor(test[, yhat_node]))
cm_true = confusionMatrix(factor(y_hat_true, levels=c("no", "yes")), factor(test[, yhat_node]))

# Printing results.
print("Confusion Matrix of the learnt Bayesian Network with a markov blanket")
```

```
## [1] "Confusion Matrix of the learnt Bayesian Network with a markov blanket"
```

```
print(cm_learnt)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  337 121
##        yes 176 366
##
##                Accuracy : 0.703
```

```
##                  95% CI : (0.6736, 0.7312)
##     No Information Rate : 0.513
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4073
##
##  Mcnemar's Test P-Value : 0.001728
##
##             Sensitivity : 0.6569
##             Specificity : 0.7515
##          Pos Pred Value : 0.7358
##          Neg Pred Value : 0.6753
##              Prevalence : 0.5130
##          Detection Rate : 0.3370
##    Detection Prevalence : 0.4580
##       Balanced Accuracy : 0.7042
##
##        'Positive' Class : no
##
```

```
print("Confusion Matrix of the true Bayesian Network with a markov blanket")
```

```
## [1] "Confusion Matrix of the true Bayesian Network with a markov blanket"
```

```
print(cm_true)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  337 121
##        yes 176 366
##
##                Accuracy : 0.703
##                  95% CI : (0.6736, 0.7312)
##     No Information Rate : 0.513
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4073
##
##  Mcnemar's Test P-Value : 0.001728
##
##             Sensitivity : 0.6569
##             Specificity : 0.7515
##          Pos Pred Value : 0.7358
##          Neg Pred Value : 0.6753
##              Prevalence : 0.5130
##          Detection Rate : 0.3370
##    Detection Prevalence : 0.4580
##       Balanced Accuracy : 0.7042
##
##        'Positive' Class : no
##
```

Task 4

Repeat the exercise (2) using a naive Bayes classifier, i.e. the predictive variables are independent given the class variable. See p. 380 in Bishop's book or Wikipedia for more information on the naive Bayes classifier. Model the naive Bayes classifier as a BN. You **have** to create the BN by hand, i.e. you are not allowed to use the function `naive.bayes` from the `bnlearn` package.

Hint: Check http://www.bnlearn.com/examples/dag/ to see how to create a BN by hand.

Answer A naive bayes clasifier is defined as:

$$p\left(S|\mathbf{x}\right) = \frac{p\left(S\right)p\left(\mathbf{x}|S\right)}{p(\mathbf{x})} = \frac{p\left(S\right)\prod_i^N p\left(x_i|S\right)}{p(\mathbf{x})}$$

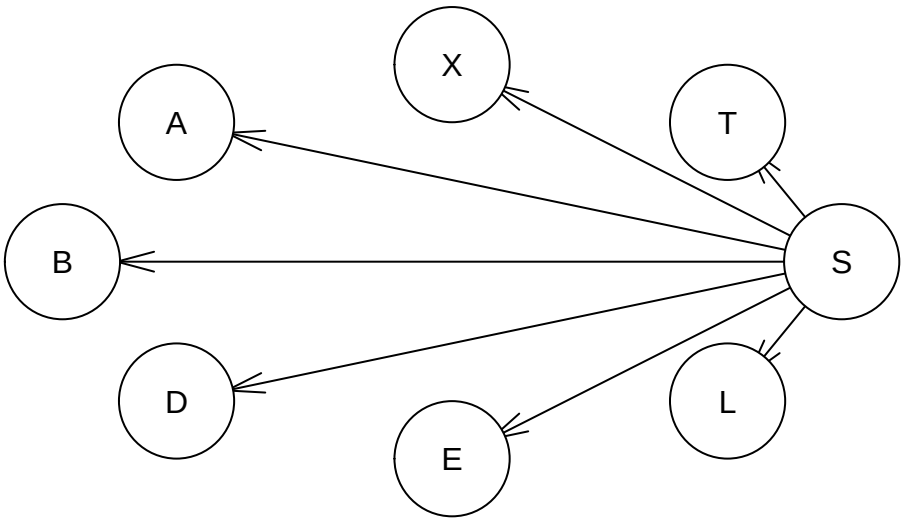
where,

S: node S

x_i: node i

This means that all nodes except S wont have any childs and only one parent (S) and that S will have every other node as a child as depicted below in the plot.

```
bn_nb = model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")
plot(bn_nb)
```



The confusion matrix is presented below.

```
# Training.
bn_nb_fit = bn.fit(bn_nb, train)

# Predicting.
y_hat_nb = predict_bn(bn_nb_fit, test, yhat_node)

# Confusion matrix.
cm = confusionMatrix(factor(y_hat_nb, levels=c("no", "yes")), factor(test[, yhat_node]))
print(cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  359 180
##        yes 154 307
##
##                Accuracy : 0.666
##                  95% CI : (0.6358, 0.6952)
##     No Information Rate : 0.513
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.3306
##
##  Mcnemar's Test P-Value : 0.1713
##
##             Sensitivity : 0.6998
##             Specificity : 0.6304
##          Pos Pred Value : 0.6660
##          Neg Pred Value : 0.6659
##              Prevalence : 0.5130
##          Detection Rate : 0.3590
##    Detection Prevalence : 0.5390
##       Balanced Accuracy : 0.6651
##
##        'Positive' Class : no
##
```

Task 5

Explain why you obtain the same or different results in the exercises (2-4).

In exercise 2 the results are the same because both of the graphs are quite similar regarding the graph and the nodes that are related with S. This yields the same posterior probabilities.

As for exercise 3, the results are the same because the markov blanket doesn't change the dependencies of the graph with S in both cases since the probability of S only depends on it's childrens.

Finally in exercise 4 we get worse results because we are inserting noise into the predictions since we assume somehow that all variables somehow determine the state of S