L1.R

adam

2025-09-12

```
if(!require(bnlearn)) install.packages("bnlearn")
```

```
## Loading required package: bnlearn
```

```
if(!require(gRain)) install.packages("gRain")
```

```
## Loading required package: gRain

## Loading required package: gRbase

##
## Attaching package: 'gRbase'

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

```
if(!require(caret)) install.packages("caret")
```

```
## Loading required package: caret

## Loading required package: ggplot2

## Loading required package: lattice
```

```
suppressPackageStartupMessages({
  library(bnlearn)
  library(gRain)
  library(caret)
})
```

TASK 1

```
# Load data set
data("asia")
```

Hill-Climbing Algorithms

Multiple runs of the hill-climbing algorithm can return non-equivalent BN structures. We demonstrate this by running hc twice with different seeds.

Hill-climbing is a form of stochastic optimization. One might assume that the algorithm should always produce equivalent graphs, but this is not guaranteed because it can converge to different local optima depending on the starting point. HC is not asymptotically correct under faithfulness, i.e. it may get trapped in local optima.

The choice of score also matters:

- BIC: log-likelihood − (penalty * number_of_params). This is an asymptotic approximation that penalizes complexity, and different runs can still lead to different solutions.
- BDe: Bayesian Dirichlet equivalent uniform prior. This evaluates P(structure | data). The parameter `iss` (imaginary sample size) controls the weight of the prior. Different values of `iss` can lead to different structures.

BIC comparison

```r
set.seed(42)
random_init <- random.graph(nodes = names(asia))
bn1_bic <- hc(asia, score = "bic", start = random_init)


random_init <- random.graph(nodes = names(asia))
bn2_bic <- hc(asia, score = "bic", start = random_init)

# Compare the DAGs
arcs(bn1_bic)
```

```
##       from to
##  [1,] "S"  "T"
##  [2,] "T"  "L"
##  [3,] "T"  "D"
##  [4,] "L"  "D"
##  [5,] "B"  "D"
##  [6,] "E"  "L"
##  [7,] "E"  "X"
##  [8,] "S"  "B"
##  [9,] "E"  "T"
## [10,] "S"  "E"
```

```r
arcs(bn2_bic)
```

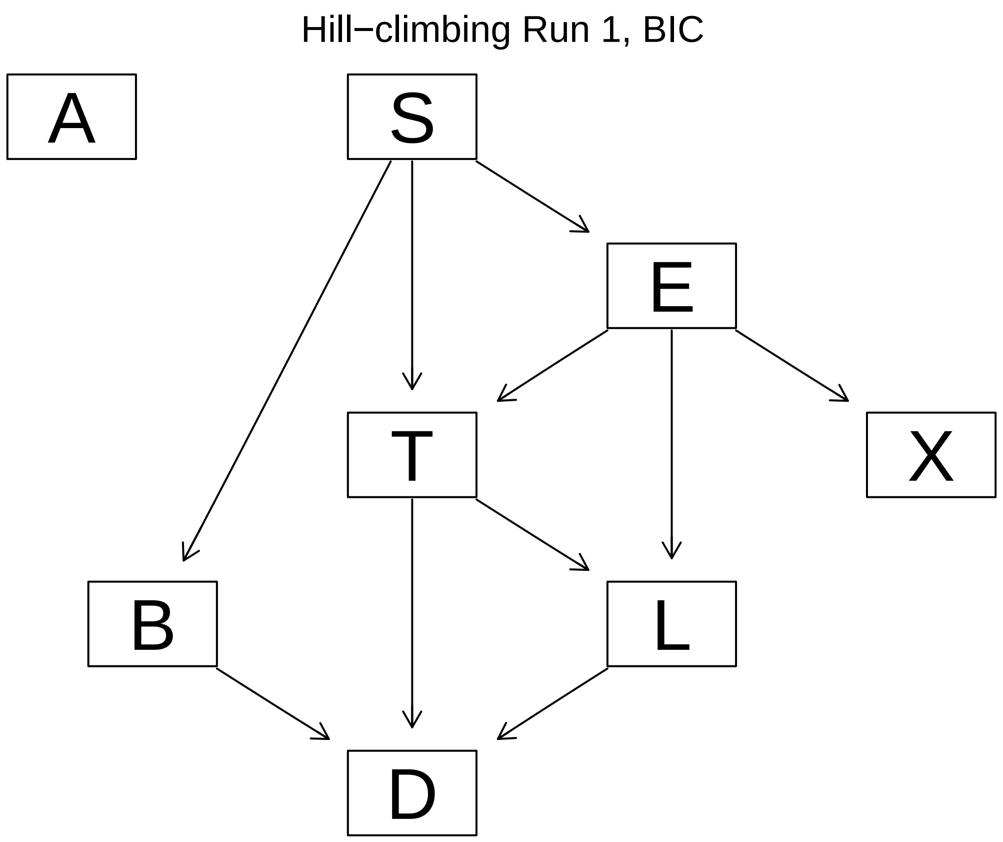
```
##      from to
## [1,] "S"  "L"
## [2,] "S"  "B"
## [3,] "E"  "D"
## [4,] "B"  "D"
## [5,] "L"  "E"
## [6,] "E"  "X"
## [7,] "T"  "E"
```

```r
# TRUE => equivalent, else non-equivalent
all.equal(cpdag(bn1_bic), cpdag(bn2_bic))
```

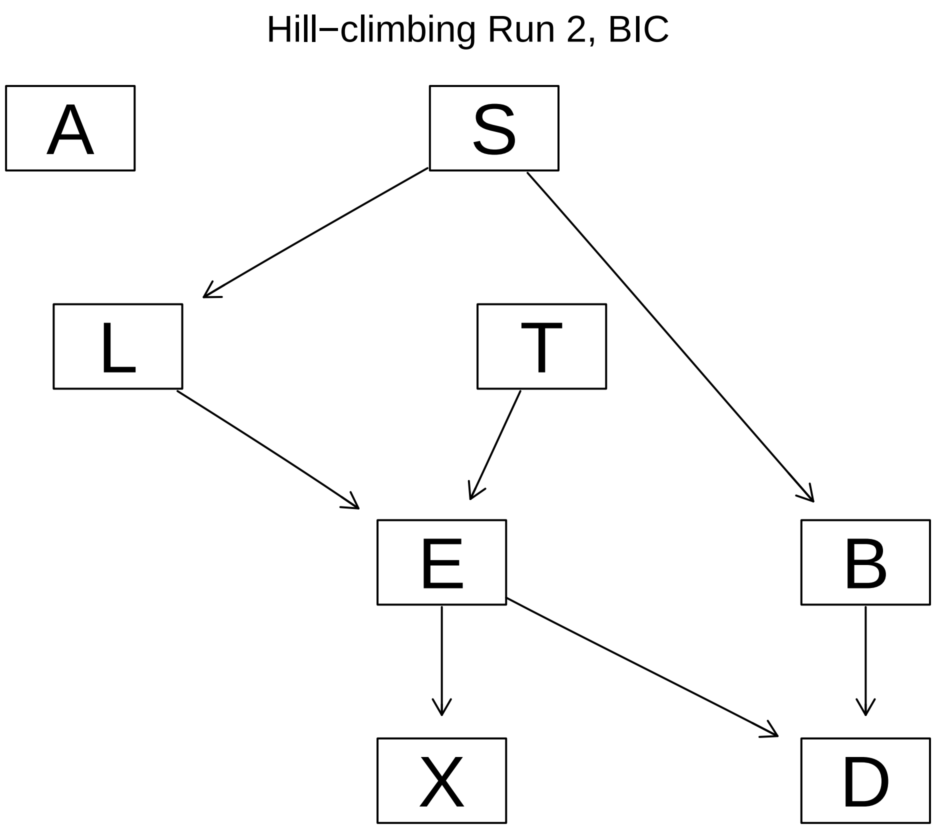
```
## [1] "Different number of directed/undirected arcs"
```

```r
# Plot both DAGs
graphviz.plot(bn1_bic, main = "Hill-climbing Run 1, BIC")
```

```
## Loading required namespace: Rgraphviz
```



```r
graphviz.plot(bn2_bic, main = "Hill-climbing Run 2, BIC")
```



BDe comparison

```r
set.seed(42)
random_init <- random.graph(nodes = names(asia))
bn1_bde <- hc(asia, score = "bde", iss = 10, start = random_init)

set.seed(24)
random_init <- random.graph(nodes = names(asia))
bn2_bde <- hc(asia, score = "bde", iss = 10, start = random_init)

# Compare the DAGs
arcs(bn1_bde)
```

```
##       from to
##  [1,] "S"  "T"
##  [2,] "T"  "L"
##  [3,] "T"  "D"
##  [4,] "L"  "D"
##  [5,] "B"  "D"
##  [6,] "E"  "L"
##  [7,] "E"  "X"
##  [8,] "S"  "B"
##  [9,] "E"  "T"
## [10,] "S"  "E"
## [11,] "T"  "A"
## [12,] "T"  "X"
## [13,] "B"  "T"
## [14,] "A"  "L"
## [15,] "L"  "X"
## [16,] "E"  "A"
```

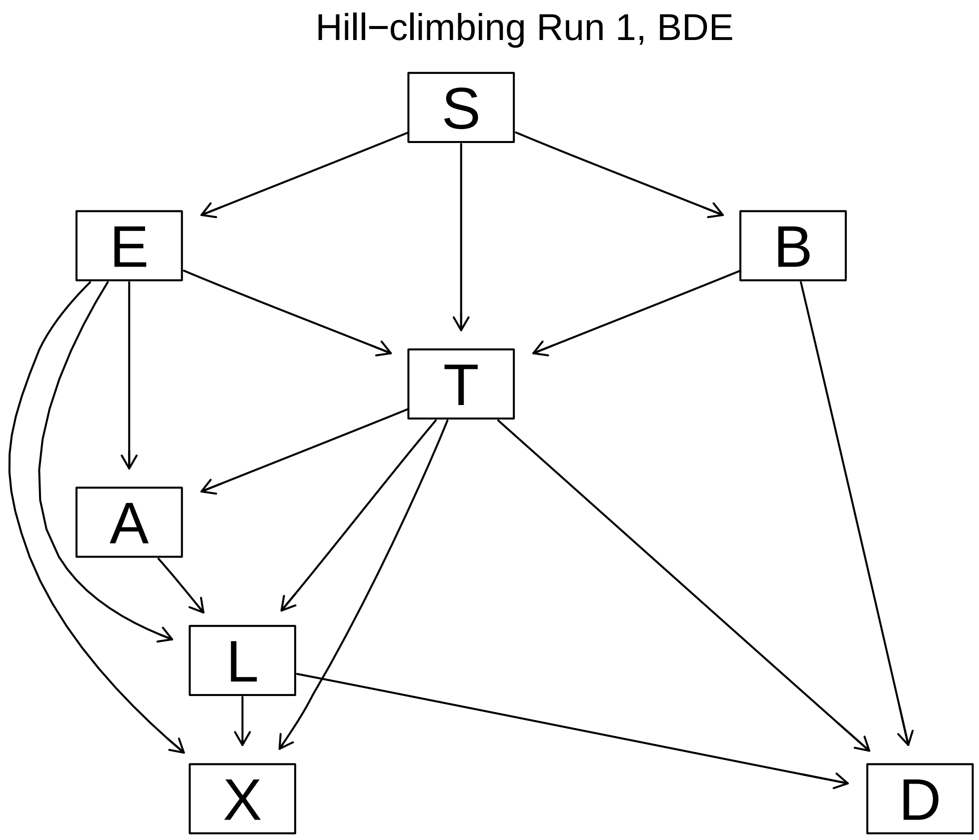
```r
arcs(bn2_bde)
```

```
##       from to
##  [1,] "A"  "T"
##  [2,] "S"  "B"
##  [3,] "T"  "X"
##  [4,] "L"  "E"
##  [5,] "L"  "X"
##  [6,] "B"  "D"
##  [7,] "E"  "D"
##  [8,] "T"  "E"
##  [9,] "S"  "L"
## [10,] "A"  "E"
## [11,] "E"  "X"
## [12,] "T"  "B"
## [13,] "L"  "A"
```

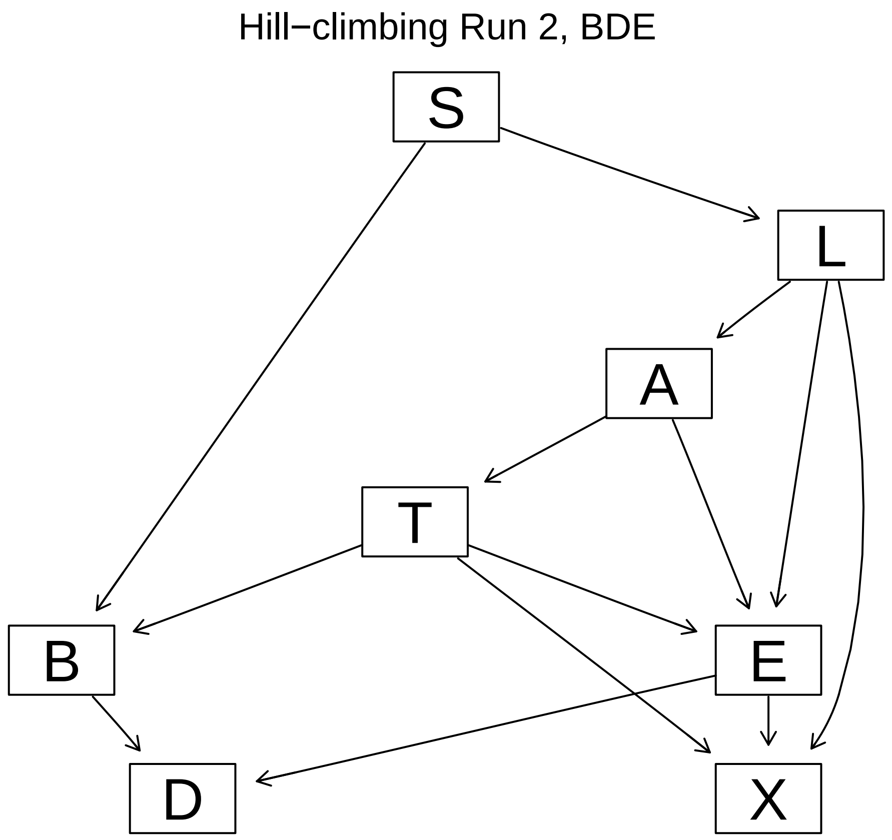
```r
# Are they equivalent? Compare equivalence classes
all.equal(cpdag(bn1_bde), cpdag(bn2_bde))   # TRUE means equivalent, FALSE means non-equivalent
```

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

```r
# Plot both DAGs
graphviz.plot(bn1_bde, main = "Hill-climbing Run 1, BDE")
```



```r
graphviz.plot(bn2_bde, main = "Hill-climbing Run 2, BDE")
```



Discussion

The question was:

Show that multiple runs of the hill-climbing algorithm can return non-equivalent Bayesian network (BN) structures. Explain why this happens. Use the Asia data set which is included in the bnlearn package. To load the data, run data("asia"). Recall from the lectures that the concept of non-equivalent BN structures has a precise meaning.

Answer:

When we use different seeds we see that we produce non-equivalent graphs. In our code the use of seeds does not affect how we partition the data, it only affects the initial graph structure.

Since the final graph depends on the initial graph, different seeds produce different graphs. However the algorithm is deterministic given the same initial graph and same score function. So given the same data we see that we converge to the same solutions given different the same initial graphs, i.e the methods are deterministic.

Task 2

Load and Partition Data

```r
set.seed(42)

data("asia")

# Partition data into training and test
n <- nrow(asia)
train_idx <- sample(1:n, size = 0.8*n)
asia_train <- asia[train_idx, ]      # takes all indexes in asia
asia_test  <- asia[-train_idx, ]     # Removes all indexes in asia
```

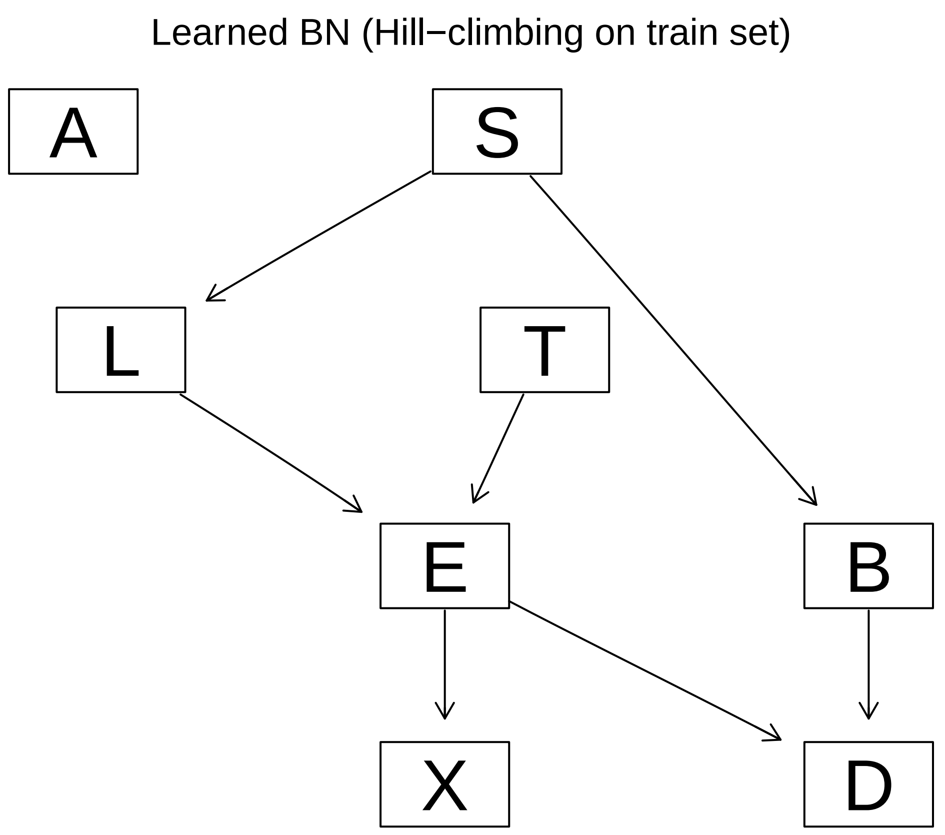
Learn Structure and Parameters

hc() - Hill-climbing algorithm for structure learning, find DAG that maximizes score function.

bn.fit() - Estimates parameters using maximum likelihood.

as.grain() - Converts bn.fit object to gRain format for exact inference. The posterior distributions are calculated on-demand when querying.

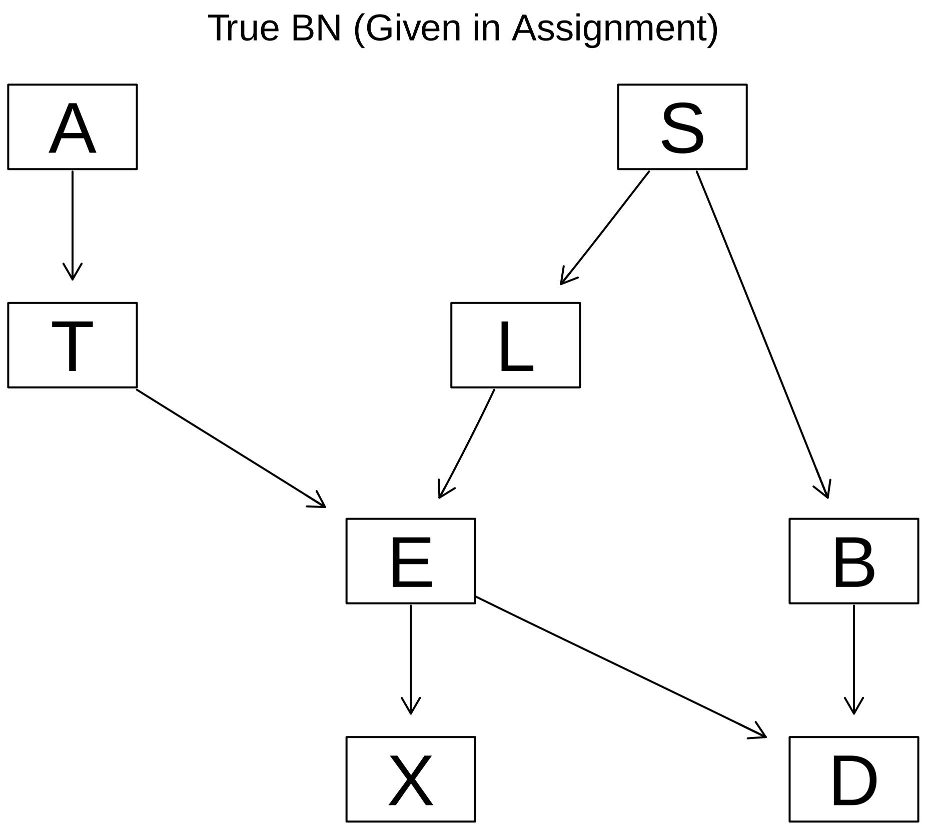
```r
bn <- hc(asia_train) # Here we use default score and start.
fitted_bn <- bn.fit(bn, asia_train) # Again, default here

# Plot learned BN
graphviz.plot(bn, main = "Learned BN (Hill-climbing on train set)")
```



```r
# Convert to gRain object for inference
grain_bn <- as.grain(fitted_bn)

# Classification of S on test data
predictions_bn <- sapply(1:nrow(asia_test), function(i) {
  ev <- asia_test[i, !(names(asia_test) %in% "S"), drop=FALSE]
  ev <- lapply(ev, as.character)
  q  <- querygrain(setEvidence(grain_bn, evidence = ev), nodes="S")$S
  ifelse(q["yes"] > q["no"], "yes", "no")
  # here there is room for changes, instead of criteria q(yes) > q(no) we could
  # instead have like q(yes) > X%
})

# Confusion matrix
bn_table <- table(True = asia_test$S, Pred = predictions_bn)
```

Construct True Graph

```r
# From instructions
dag_true = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")
fitted_true = bn.fit(dag_true, data = asia)
grain_true = as.grain(fitted_true)

graphviz.plot(dag_true, main = "True BN (Given in Assignment)")
```



```r
# Classification of S on test data
predictions_true <- sapply(1:nrow(asia_test), function(i) {
  ev <- asia_test[i, !(names(asia_test) %in% "S"), drop=FALSE]
  ev <- lapply(ev, as.character)
  # Calculate posterios given evidence
  q  <- querygrain(setEvidence(grain_true, evidence = ev), nodes="S")$S
  ifelse(q["yes"] > q["no"], "yes", "no")
})


# Confusion matrix
true_table <- table(True = asia_test$S, Pred = predictions_true)
```

Comparison

Here we simply note that they are more or less identical implying very similar predicative performance.

```r
true_table
```

```
##      Pred
## True   no yes
##   no  347 160
##   yes 125 368
```

```r
bn_table
```

```
##      Pred
## True   no yes
##   no  347 160
##   yes 125 368
```

Discussion

Question:

Learn a BN from 80 % of the Asia data set. The data set 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 data set 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 Asia 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]")

Answer

If we compare the confusion matrices we see that the trained BN produce the same results as the true dag (in the confusion matrix metric). However, when we compare the structure of the graphs we see that they differ. Upon inspection of the plots of the graphs this is obvious.

This is interesting because we here see that networks that are different may have the same predictive performance.

Task 3

Load and Partition Data

```r
set.seed(42)

# First run task 2 to get the work space
# Our fitted BN is called "fitted_bn"
```

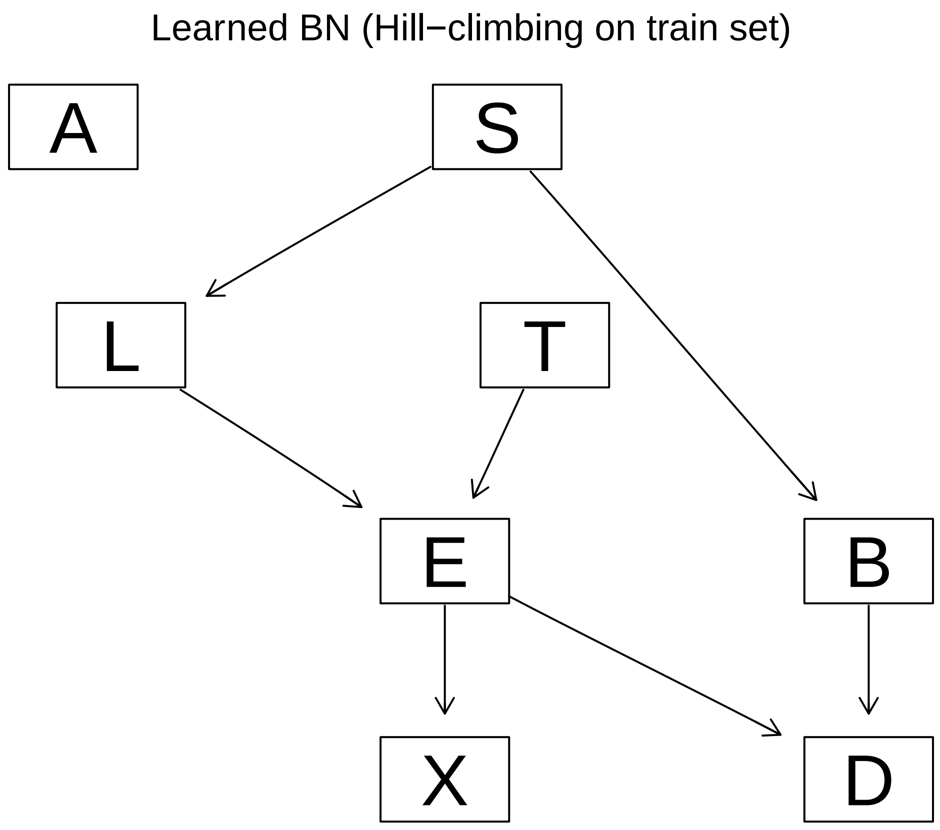
Get the Markov Blankets of S in the Network

```r
mb_vars <- mb(fitted_bn, "S")
mb_vars
```

```
## [1] "L" "B"
```

So we see that the Markov blankets of S are "L" and "B". This we can verify by stating the definition and observe the graph Markov blankets are defined as the minimal set of nodes that separates a given node from the rest.

```r
graphviz.plot(bn, main = "Learned BN (Hill-climbing on train set)")
```



Here it is easy to see that L and B satisfy this.

Now we classify using only Markov blankets.

```r
# Classify using only Markov blanket variables
predictions_mb <- sapply(1:nrow(asia_test), function(i) {
  # Get only the Markov blanket variables from test case
  ev <- asia_test[i, mb_vars, drop = FALSE]
  ev <- lapply(ev, as.character)

  # Query probability of S given only Markov blanket evidence
  q <- querygrain(setEvidence(grain_bn, evidence = ev), nodes = "S")$S
  ifelse(q["yes"] > q["no"], "yes", "no")
})

# Confusion matrix for Markov blanket classification
mb_table <- table(True = asia_test$S, Pred = predictions_mb)
```

Comparison of Results

Using ALL variables (except S)

```r
print(bn_table)
```

```
##      Pred
## True   no yes
##   no  347 160
##   yes 125 368
```

This had accuracy:

```r
print(round(sum(diag(bn_table)) / sum(bn_table), 4))
```

```
## [1] 0.715
```

Using only MARKOV BLANKET variables

```r
print(mb_table)
```

```
##      Pred
## True   no yes
##   no  347 160
##   yes 125 368
```

This had accuracy:

```r
print(round(sum(diag(mb_table)) / sum(mb_table), 4))
```

```
## [1] 0.715
```

Using the True BN structure

```r
print(true_table)
```

```
##      Pred
## True   no yes
##   no  347 160
##   yes 125 368
```

This has accuracy:

```r
round(sum(diag(true_table)) / sum(true_table), 4)
```

```
## [1] 0.715
```

Performance Metrics Comparison

We use this function to avoid code duplication

```r
calculate_metrics <- function(conf_matrix, label) {
  tp <- conf_matrix["yes", "yes"]
  fp <- conf_matrix["yes", "no"]
  fn <- conf_matrix["no", "yes"]
  tn <- conf_matrix["no", "no"]

  accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
  precision <- ifelse((tp + fp) > 0, tp / (tp + fp), 0)
  recall <- ifelse((tp + fn) > 0, tp / (tp + fn), 0)
  f1 <- ifelse((precision + recall) > 0, 2 * (precision * recall) / (precision + recall), 0)

  cat(sprintf("\n%s Metrics:", label))
  cat(sprintf("\n  Accuracy:  %.4f", accuracy))
  cat(sprintf("\n  Precision: %.4f", precision))
  cat(sprintf("\n  Recall:    %.4f", recall))
  cat(sprintf("\n  F1-score:  %.4f", f1))
}

calculate_metrics(bn_table, "All Variables")
```

```
##
## All Variables Metrics:
```

```
##   Accuracy:  0.7150
##   Precision: 0.7465
##   Recall:    0.6970
##   F1-score:  0.7209
```

```r
calculate_metrics(mb_table, "Markov Blanket Only")
```

```
##
## Markov Blanket Only Metrics:
##   Accuracy:  0.7150
##   Precision: 0.7465
##   Recall:    0.6970
##   F1-score:  0.7209
```

```r
calculate_metrics(true_table, "True BN Structure")
```

```
##
## True BN Structure Metrics:
##   Accuracy:  0.7150
##   Precision: 0.7465
##   Recall:    0.6970
##   F1-score:  0.7209
```

Discussion

Question:

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 blanket of S, i.e. its parents plus its children plus the parents of its children minus S itself. Report again the confusion matrix.

Answer

The Markov blanket of a variable contains all variables that can provide information about it which is parents, children, and parents of children (called spouses?). According to the Markov property, the variable S is conditionally independent of all other variables given its Markov blanket.

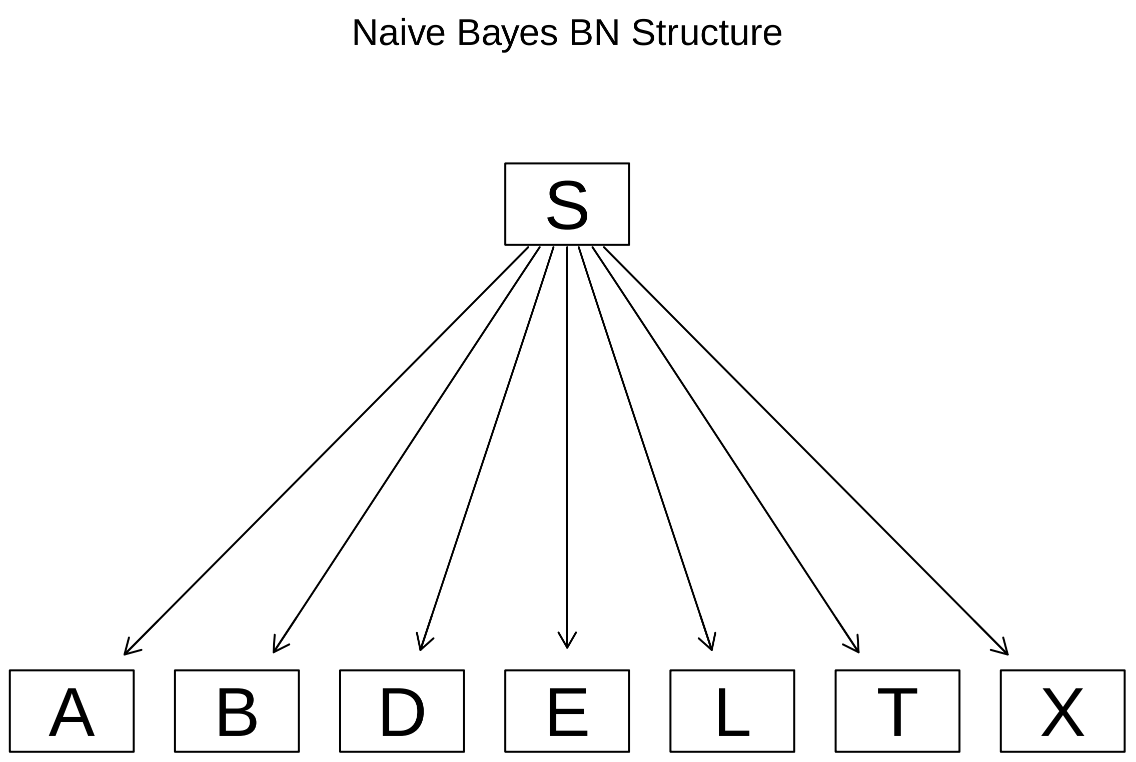
So we expect the predicative preformance of the leaned BN useing all nodes and only the Markov blanket nodes to have the same predicative preformance of S. So S is conditionally independent given the Markov Blanket. That is,

$S \perp \{T, E, X, D\} | \{L, B\}$

And this is supported by the results.

Task 4

```r
# Create the naive Bayes DAG structure manually
# S is the parent of all other variables
nb_dag <- model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")
graphviz.plot(nb_dag, main="Naive Bayes BN Structure")
```



```r
# Fit the parameters using the training data
nb_fitted <- bn.fit(nb_dag, data = asia_train)

# Convert to gRain object for inference
nb_grain <- as.grain(nb_fitted)
```

Classification on test set using Naive Bayes

```r
predictions_nb <- sapply(1:nrow(asia_test), function(i) {
  case <- asia_test[i, ]

  # Evidence: all variables except S (all predictive features)
  ev <- as.list(case[ , !(names(case) %in% "S"), drop = FALSE])

  # Ensure character values (not factors)
  ev <- lapply(ev, as.character)

  # Calculates posterior distributions given evidence
  nb_evid <- setEvidence(nb_grain, evidence = ev)

  # Get posterior probability for S
  prob_S <- querygrain(nb_evid, nodes = "S")$S

  # Pick most probable class
  if (prob_S["yes"] > prob_S["no"]) "yes" else "no"
})
```

Confusion matrix for Naive Bayes

```
conf_matrix_nb <- table(Actual = asia_test$S, Predicted = predictions_nb)
print("Confusion Matrix for Naive Bayes Classifier:")
```

```
## [1] "Confusion Matrix for Naive Bayes Classifier:"
```

```
print(conf_matrix_nb)
```

```
##       Predicted
## Actual  no yes
##    no  364 143
##    yes 178 315
```

Compare with previous results

```
print("Confusion Matrix using Full BN (from exercise 1):")
```

```
## [1] "Confusion Matrix using Full BN (from exercise 1):"
```

```
print(bn_table)
```

```
##      Pred
## True   no yes
##   no  347 160
##   yes 125 368
```

```
calculate_metrics(bn_table, "With HC")
```

```
##
## With HC Metrics:
##   Accuracy:  0.7150
##   Precision: 0.7465
##   Recall:    0.6970
##   F1-score:  0.7209
```

```
calculate_metrics(conf_matrix_nb, "With naîve Bay")
```

```
##
## With naîve Bay Metrics:
##   Accuracy:  0.6790
##   Precision: 0.6389
##   Recall:    0.6878
##   F1-score:  0.6625
```

Discussion

Question:

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.

Answer

We state that S depends on all variables, then given this structure we calculate all posteriors.

In previous tasks we did this,

Find suitable structure, THEN GIVEN this structure we fit parameters using maximum likelihood, then use grain to make queries.

Now we do this,

Assume a structure, THEN GIVEN this structure we fit parameters using maximum likelihood, then use grain to make queries.

This is reasonable since to find the optimal structure is O(N!), HC is a heuristic that finds a "Good" structure. And in this task we see that we can get similar but worse performance by naively chosing a structure.

What is important here is the fact that the structure with Naive bay is O(1).

Task 5

Comparison

We summarize the results again for the discussion.

```r
calculate_metrics(bn_table, "Task 2: All Variables")
```

```
##
## Task 2: All Variables Metrics:
##   Accuracy:  0.7150
##   Precision: 0.7465
##   Recall:    0.6970
##   F1-score:  0.7209
```

```r
calculate_metrics(mb_table, "Task 3: Markov Blanket Only")
```

```
##
## Task 3: Markov Blanket Only Metrics:
##   Accuracy:  0.7150
##   Precision: 0.7465
##   Recall:    0.6970
##   F1-score:  0.7209
```

```r
calculate_metrics(conf_matrix_nb, "Task 4: Naîve Bayes")
```

```
##
## Task 4: Naîve Bayes Metrics:
##   Accuracy:  0.6790
##   Precision: 0.6389
##   Recall:    0.6878
##   F1-score:  0.6625
```

And performance of the true network structure for reference.

```r
calculate_metrics(true_table, "Reference: True BN Structure")
```

```
##
## Reference: True BN Structure Metrics:
##   Accuracy:  0.7150
##   Precision: 0.7465
##   Recall:    0.6970
##   F1-score:  0.7209
```

Question:

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

Answer:

So, in the task 2 a BN structure learned from data using hill-climbing, in task 3 same learned BN, but using only the Markov blanket of S for classification and in task 4 we manually constructed naive Bayes classifier where S is the parent of all other variables.

We found that task 2 and task 3, but this was expected from theory. What was interesting is that the performance of the naîve Bayes network was comparable to the first 2 networks. However when considering that to find the optimal structure is O(N!) it is not that far fetched that we get an OK performance with the naîve approach.