732A96: AML - Computer Lab 1

Julius Kittler (julki092)

September 17, 2019

Contents


```
# Set up general options

knitr::opts_chunk$set(echo = FALSE, warning = FALSE, message = FALSE,
                      fig.width=6, fig.height=5#, collapse=TRUE
                      )


set.seed(12345)
options(scipen=999)

# General libraries
library(ggplot2)

# Specific packages
library("bnlearn") # ls('package:bnlearn') # to show all functions
library("gRain")
library('RBGL')
```

```
# Auxilary functions
analyze_cm = function(cm, true){

  stopifnot(true %in% colnames(cm))
  levels = c(true, colnames(cm)[-which(colnames(cm) == true)]) # ORDER: 1; 0
  cm = as.data.frame(cm); colnames(cm)[1:2] = c("True", "Pred")
  N = sum(cm$Freq)
  Npos = sum(cm$Freq[which(cm$True == levels[1])])
  Nneg = sum(cm$Freq[which(cm$True == levels[2])])
  TP = sum(cm$Freq[which(cm$True == levels[1] & cm$Pred == levels[1])])
```

```
  TN = sum(cm$Freq[which(cm$True == levels[2] & cm$Pred == levels[2])])
  FP = sum(cm$Freq[which(cm$True == levels[2] & cm$Pred == levels[1])])

  FN = sum(cm$Freq[which(cm$True == levels[1] & cm$Pred == levels[2])])
  return(data.frame(MCR = (FP+FN)/N, Accuracy = (TP + TN)/N,
                    Recall = TP/Npos, # recall = TPR = sensitivity,
                    Precision = TP/(TP + FP),
                    FPR = FP/Nneg, TNR = TN/Nneg)) # TNR = specificity
}

# cm = table(Y_true, Y_pred, dnn = c("True", "Predicted"))
# knitr::kable(analyze_cm(cm, true = "yes"))
```

---

1 Assignment 1: Hill Climbing

The purpose of the lab is to put in practice some of the concepts covered in the lectures.

(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 number 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`.

```
# Prepare data
data("asia")
df = data.frame(asia)
rm(asia)
head(df)
```

```
##    A   S   T  L   B   E   X   D
## 1 no yes  no no yes  no  no yes
## 2 no yes  no no  no  no  no  no
## 3 no  no yes no  no yes yes yes
## 4 no  no  no no yes  no  no yes
## 5 no  no  no no  no  no  no yes
## 6 no yes  no no  no  no  no yes
```

Definition: Equivalence

Two DAGs represent the same independencies (i.e. they are equivalent) if and only if they have the same adjacencies and unshielded colliders, i.e. subgraphs $X_i \rightarrow X_k \leftarrow X_j$, where X_i and X_j are

not adjacent (Lecture 4).

How to show non-equivalence

We can test if two DAGs are equivalent by testing if their CPDAGs (=complete partially directed graphs) are the same. If they are, we have equivalent DAGs. If they are not, we have non-equivalent DAGs.

For this we can use the function `cpdag`. The function `cpdag` converts certain directed connections in the network to undirected connections. It does this for all connections in the subgraphs that form a class of equivalence (see below). The subgraphs below are all converted to $X_i - X_k - X_j$.

- $X_i \leftarrow X_k \leftarrow X_j$
- $X_i \rightarrow X_k \rightarrow X_j$
- $X_i \leftarrow X_k \rightarrow X_j$

The function `cpdag` leaves colliders as they are. I.e. $X_i \rightarrow X_k \leftarrow X_j$ is left as is. Since all adjacencies and unshielded colliders are kept in the DAG when applying the function, we only need to compare the two dags after taking their `cpdag` version. If they are different, then obviously we have non-equivalent graphs by the definition of equivalence.

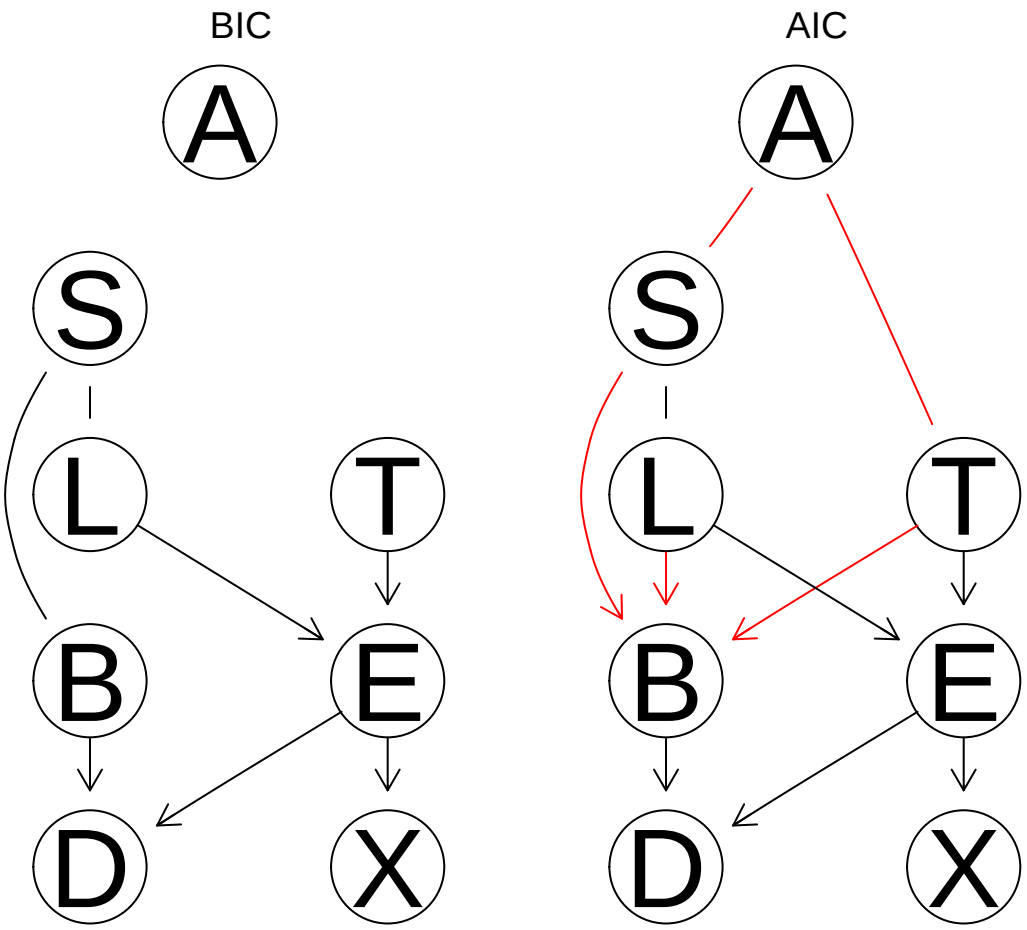
Non-equivalence due to different score metrics

Below, we use `hc` one time with `score='bic'` and one time with `score='aic'`. We can see that the 2 resulting dags are *not equivalent* by comparing the arcs in their CPDAGs and finding that they differ.

```
# Non-equivalence due to different score metrics ------------------------------

# Train models
set.seed(12345)
res1 = hc(df, restart=10, score='bic')
res2 = hc(df, restart=10, score='aic')
# cat("Score", bnlearn::score(res, df)) # smaller AIC, BIC is better

# Check equivalence: logically
arcs1 = arcs(cpdag(res1))
arcs2 = arcs(cpdag(res2))
identical(arcs1[order(arcs1[, 1]), ], arcs2[order(arcs2[, 1]), ])
```

```
## [1] FALSE
```

```
# Check equivalence: visually
par(mfrow = c(1, 2))
graphviz.compare(cpdag(res1), cpdag(res2), main = c("BIC", "AIC"))
```



Non-equivalence due to different initial random graphs

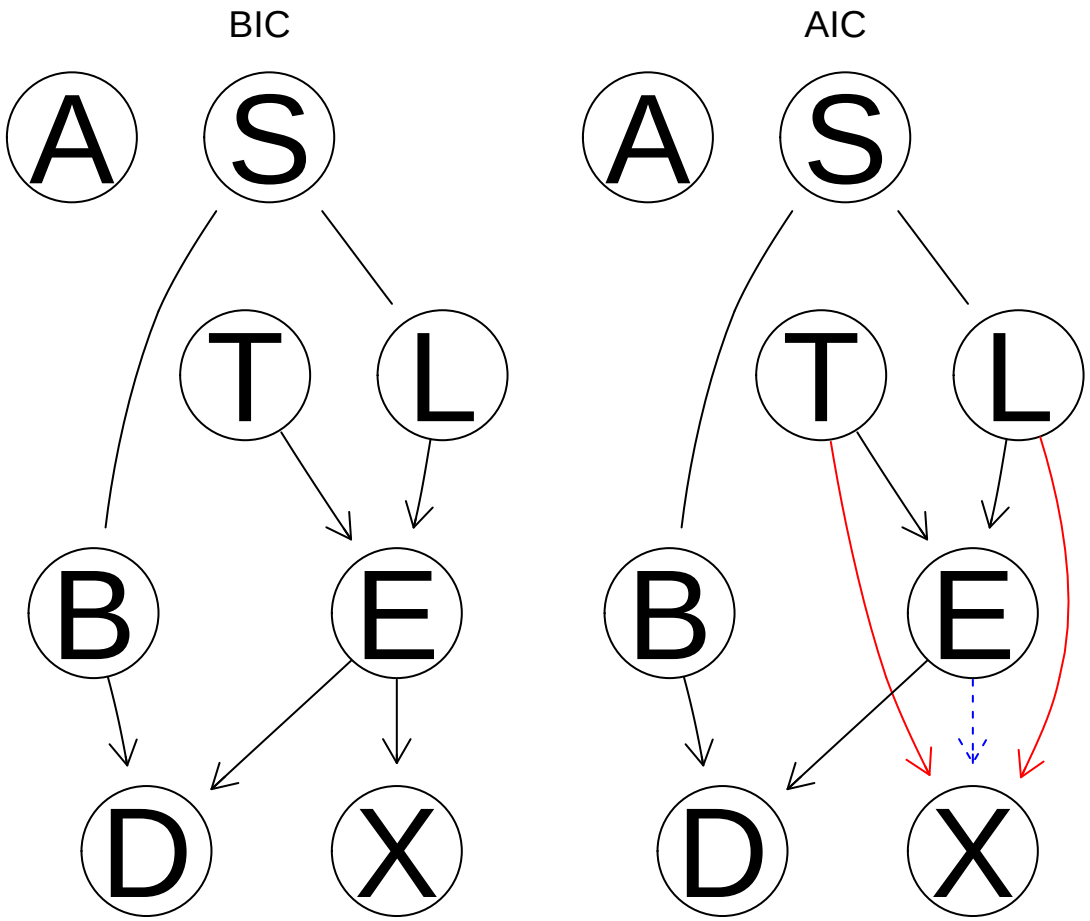
Below, we use `hc` two times, on different initial randomly generated graphs. We can see that the 2 resulting dags are *not equivalent* by comparing the arcs in their CPDAGs and finding that they differ.

```
# Non-equivalence due to different initial random graphs ----------------------

# Train models
set.seed(12345)
rand = random.graph(colnames(df))
res1 = hc(asia, start = rand, restart = 100, score='bic')
rand = random.graph(colnames(df))
res2 = hc(asia, start = rand, restart = 100, score='bic')

# Check equivalence: logically
arcs1 = arcs(cpdag(res1))
arcs2 = arcs(cpdag(res2))
identical(arcs1[order(arcs1[, 1]), ], arcs2[order(arcs2[, 1]), ])
```

```
## [1] FALSE
```

```
# Check equivalence: visually
par(mfrow = c(1, 2))
```

```
graphviz.compare(cpdag(res1), cpdag(res2), main = c("BIC", "AIC"))
```



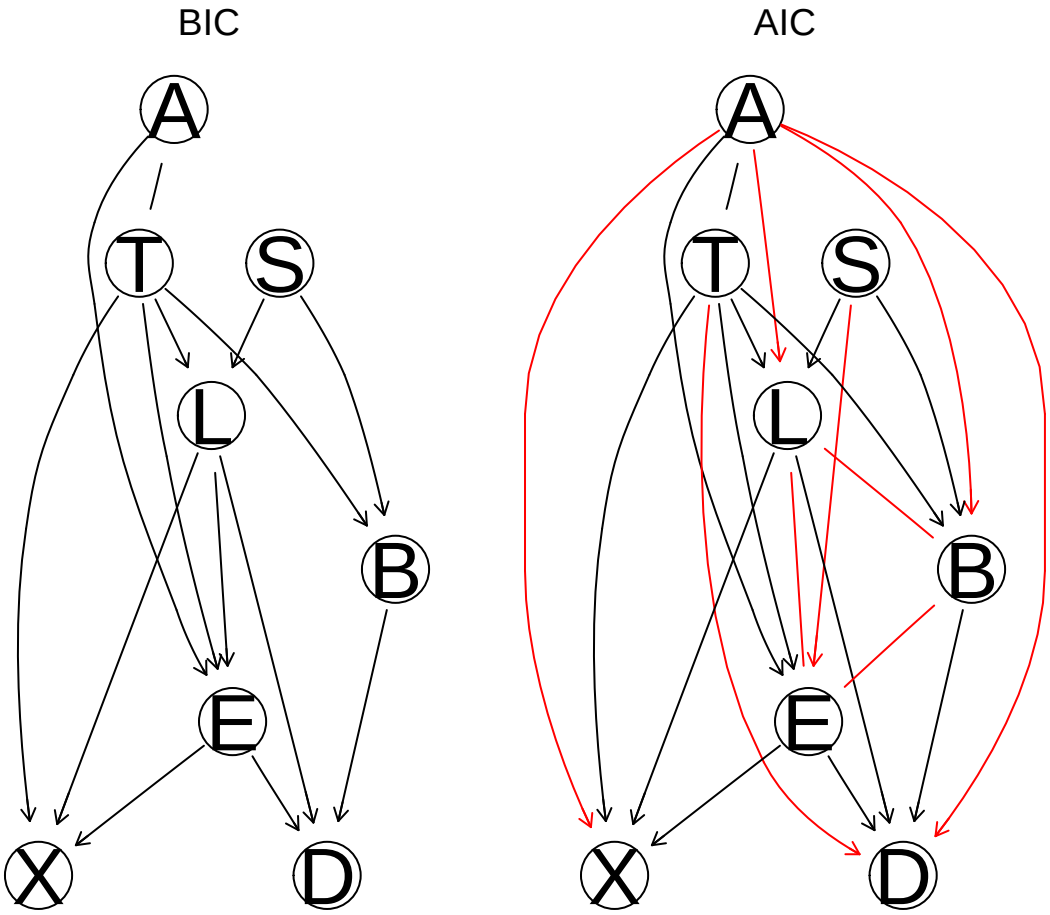
Non-equivalence due to different iss (imaginary sample size)

Below, we use `hc` two times: with `iss = 20` and with `iss = 200`. We can see that the 2 resulting dags are *not equivalent* by comparing the arcs in their CPDAGs and finding that they differ. To be clear, the network learned with a smaller iss is more sparse than the network with a larger iss. In general, iss serves as a regularization parameter with larger iss resulting in less regularization and therefore less sparse networks.

```
# Train models
set.seed(12345)
res1 = hc(asia, restart = 10, score = 'bde', iss = 20)
res2 = hc(asia, restart = 10, score = 'bde', iss = 200)
# cat("Score", bnlearn::score(res, df)) # smaller AIC, BIC is better

# Check equivalence: logically
arcs1 = arcs(cpdag(res1))
arcs2 = arcs(cpdag(res1))
identical(arcs1[order(arcs1[, 1]), ], arcs2[order(arcs2[, 1]), ])
```

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

```r
# Check equivalence: visually
par(mfrow = c(1, 2))
graphviz.compare(cpdag(res1), cpdag(res2), main = c("BIC", "AIC"))
```



References

- http://www.bnlearn.com/examples/compare-dags/
- http://bnlearn.com/examples/score/

2 Assignment 2: Structure/Parameter Learning and Inference

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 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]")`.

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` from the `bnlearn` package. When you try to load the package `gRain`, you will get an error as the pack- age `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")
```

Prepare data

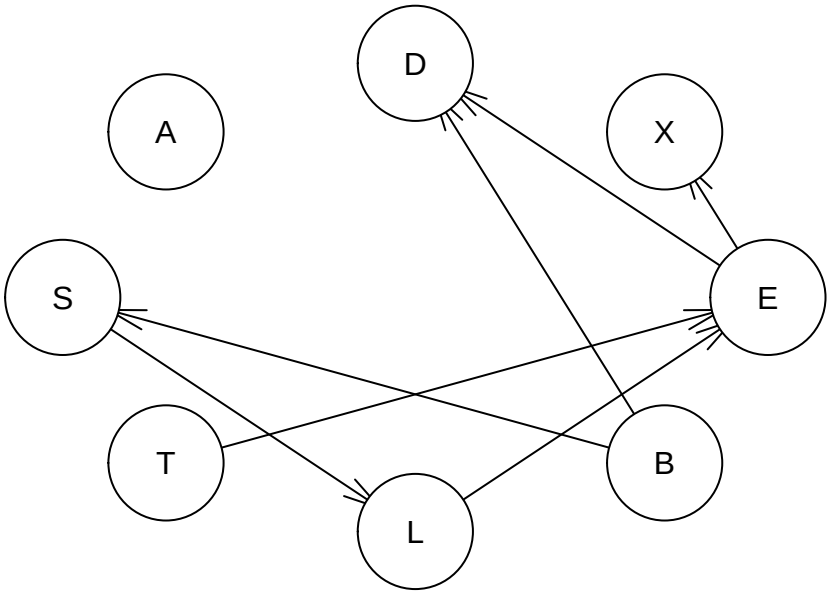
```
# Prepare data
data("asia")
df = asia
#df = data.frame(apply(asia, 2, as.character), stringsAsFactors = F)
rm(asia)

# Split data randomly
set.seed(12345)
N = nrow(df)
idx = sample(1:N, round(N * 0.8))
df_tr = df[idx, ]
df_te = df[-idx, ]
```

Fit model on training data

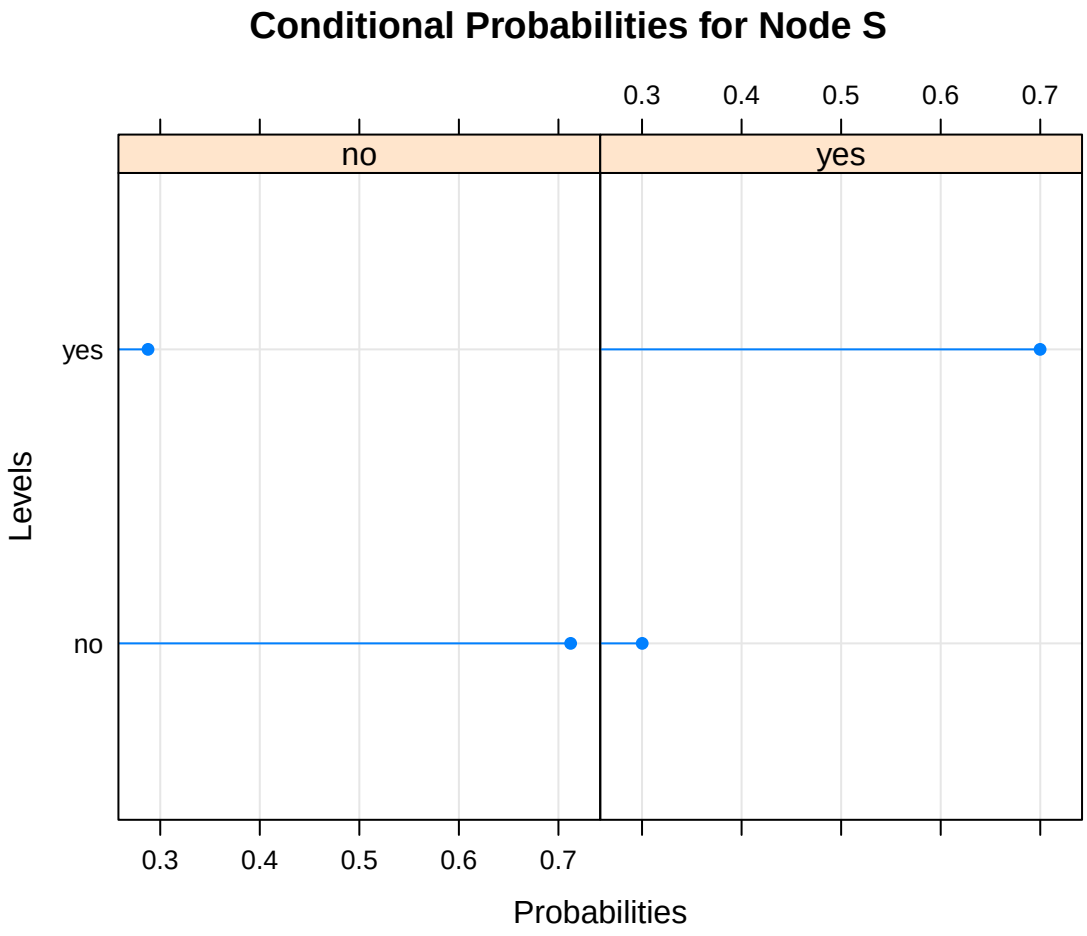
First, we have to learn the network structure and second we have to learn the parameters. To learn the stucture, we could choose one out of many different algorithms: constraint-based, score based, local search. Here, the (BIC) score-based algorithm `Hill-Climbing` is used. For learning the parameters, the method `mle` was used (for Bayesian parameter estimation).

```
# Fit the structure (given data set)
dag = hc(df_tr, restart=100, score='bic')
# dag = iamb(df_tr)

# Fit the parameters (given structure and data set)
dag_fitted = bn.fit(dag, df_tr, method='mle')
```

Show model




```
##
##   Parameters of node S (multinomial distribution)
##
## Conditional probability table:
##
##      B
## S            no       yes
##   no  0.7122665 0.3001486
##   yes 0.2877335 0.6998514
```



Evaluate model on test data

Here, we use the BN learned to classify the remaining 20 % of the Asia dataset in two classes: S = yes and S = no. We compute the posterior probability distribution of S for each case and classify it in the most likely class. For that, we use the packages `bnlearn` (for approximate inference with Monte Carlo) or `gRain` (for exact inference).

Exact inference

```
predict_exact = function(dag_fitted, df_te, y_name, markov = FALSE){

  # ------------------------------------------------------------------------
  # Input:
  # - dag_fitted: dag with learned structure and parameters
  # - df_te: data frame with all features and the Y (all factors)
  # - y_name: string with the name of the Y variable

  # Output:
  # - list with factor vectors: Y_true and Y_pred
  # ------------------------------------------------------------------------

  junction = compile(as.grain(dag_fitted))
```

```
  N = nrow(df_te)
  Y_true = df_te[[y_name]]
  Y_pred = numeric(length = N)
  Y_space = as.character(unique(Y_true))

  for (i in 1:N){

    # With all RVs
    x = df_te[i, colnames(df_te) != y_name]

    if (markov == FALSE){
      var_names = names(x)
      var_values = vapply(x, as.character, character(1))
    } else {
      var_names = mb(dag_fitted, y_name)
      var_values = vapply(x, as.character, character(1))[var_names]
    }
    evidence = setEvidence(junction, nodes = var_names, states = var_values)

    Y_prob = querygrain(evidence, nodes = y_name)[[1]]
    Y_pred[i] = Y_space[which.max(Y_prob)]

  }

  return(list(Y_pred = Y_pred, Y_true = Y_true))

}


# Make predictions
res = predict_exact(dag_fitted, df_te, "S")
Y_true = res$Y_true
Y_pred = res$Y_pred
```

```
## Confusion matrix (absolute):
```

```
##      Predicted
## True   no yes
##   no  322 146
##   yes 120 412
```

```
##
## Confusion matrix (relative):
```

```
##      Predicted
## True     no   yes
##   no  0.322 0.146
##   yes 0.120 0.412
```

```
##
## Performance statistics:
```

MCR	Accuracy	Recall	Precision	FPR	TNR
0.266	0.734	0.7744361	0.7383513	0.3119658	0.6880342

Approximate inference

Note: `cpquery` was used here instead of `cdist`. Unfortunately, we had to hardcode various values for that. Also, there was an issue with using `cpquery` in a function: It could only find `x` if it is in the main working environment.

```
# ------------------------------------------------------------------------
# Input:
# - dag_fitted: dag with learned structure and parameters
# - df_te: data frame with all features and the Y (all factors)

# MAKE SURE TO ADJUST THE EVIDENCE AND EVENT!
# ------------------------------------------------------------------------

N = nrow(df_te)
Y_true = df_te[['S']]
Y_pred = numeric(length = N)
Y_space = as.character(unique(Y_true))
Y_prob = numeric(length = length(Y_space))
junction = compile(as.grain(dag_fitted))

for (i in 1:N){

  # With all RVs
  x = vapply(df_te[i, colnames(df_te) != 'S'], as.character, character(1))

  for (j in 1:length(Y_space)){
    #var_names = names(x)
    #var_values = x
    Y_prob[j] = cpquery(dag_fitted, event = (S == Y_space[j]),
                        # evidence = setEvidence(junction, nodes = var_names,
                        #                        states = var_values)
                          evidence = ((A == x['A']) & (T == x['T'])
                                      & (L == x['L']) & (B == x['B'])
                                      & (E == x['E']) & (X == x['X'])
                                      & (D == x['E']))
                        )
  }

  Y_pred[i] = Y_space[which.max(Y_prob)]
}
```

```
## Confusion matrix (absolute):
```

```
##      Predicted
## True   no yes
##   no  322 146
##   yes 122 410
```

```
##
## Confusion matrix (relative):
```

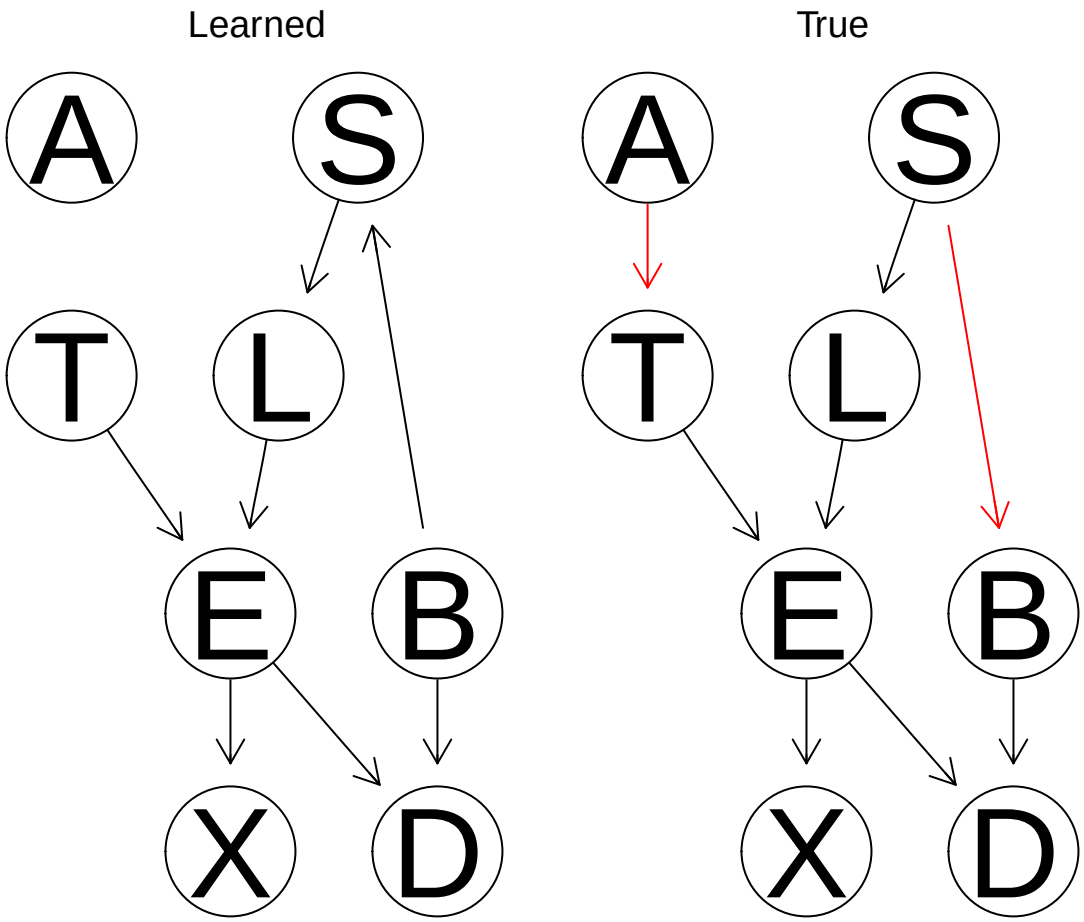
```
##      Predicted
## True     no   yes
##   no  0.322 0.146
##   yes 0.122 0.410
```

```
##
## Performance statistics:
```

MCR	Accuracy	Recall	Precision	FPR	TNR
0.268	0.732	0.7706767	0.7374101	0.3119658	0.6880342

Comparison with true DAG (only exact inference)

```
# Prepare the structure

# Option 1:
dag_true = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

# #Option 2
# dag_true = empty.graph(names(df))
# modelstring(dag_true) = "[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]"

# Visual comparison only:
par(mfrow = c(1, 2))
graphviz.compare(dag, dag_true, main = c("Learned", "True"))
```




```
# Fit the parameters (given structure and data set)
dag_fitted = bn.fit(dag_true, df_tr, method='mle')

# Make predictions
res = predict_exact(dag_fitted, df_te, "S")
Y_true = res$Y_true
Y_pred = res$Y_pred
```

```
## Confusion matrix (absolute):
##      Predicted
## True   no yes
##   no  322 146
##   yes 120 412
##
## Confusion matrix (relative):
##      Predicted
## True     no   yes
##   no  0.322 0.146
##   yes 0.120 0.412
##
```

```
## Performance statistics:
```

MCR	Accuracy	Recall	Precision	FPR	TNR
0.266	0.734	0.7744361	0.7383513	0.3119658	0.6880342

References

- http://bnlearn.com/examples/whitelist/
- http://bnlearn.com/documentation/man/structure.learning.html
- http://bnlearn.com/examples/fit/

3 Assignment 3: Structure/Parameter Learning and Inference (Markov Blanket)

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.

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

Exact inference

Note: This time we only use *Exact inference*.

The confusion matrix looks exactly the same as before. This seems expected since variables that are not in the Markov blanket of `S` only have a rather small relationship with `S` and hence don't help with the classification.

```
# Fit the structure (given data set)
dag = hc(df_tr, restart=10, score='bic')

# Fit the parameters (given structure and data set)
dag_fitted = bn.fit(dag, df_tr, method='mle')

# Make predictions
res = predict_exact(dag_fitted, df_te, "S", markov = TRUE)
Y_true = res$Y_true
Y_pred = res$Y_pred
```

```
## Confusion matrix (absolute):
```

```
##      Predicted
## True   no yes
##   no  322 146
##   yes 120 412
```

```
##
## Confusion matrix (relative):
##      Predicted
## True     no   yes
##   no  0.322 0.146
##   yes 0.120 0.412
##
## Performance statistics:
```

MCR	Accuracy	Recall	Precision	FPR	TNR
0.266	0.734	0.7744361	0.7383513	0.3119658	0.6880342

4 Assignment 4: Naive Bayes

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.

Review Naive Bayes

Conditional probability is defined as:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$
$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

Solving both equations for $P(A \cap B)$ and setting them equal, we get:

$$P(A|B)P(B) = P(B|A)P(A)$$
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Replace A with S and B with X (all other variables):

$$P(S|X) = \frac{P(X|S)P(S)}{P(X)} \propto P(X|S)P(S)$$

Since we have several `Xs` and since they are assumed to be independent (recall that independence implies that $P(A, B) = P(A)P(B)$):

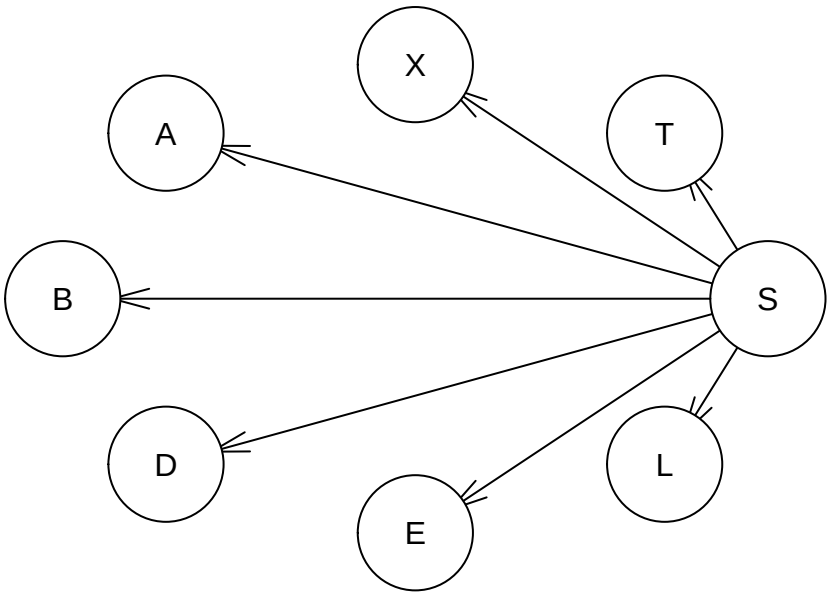
$$P(S|X) \propto P(X_1, X_2, ..., X_n|S)P(S) = P(X_1|S)P(X_2|S)...P(X_D|S)P(S) = P(S)\prod_{d=1}^{D} P(X_d|S)$$

The above tells us that S has to point to all X. Finally, classifications in naive bayes for observation i are done by:

$$\text{argmax}_{S_i}\, P(S_i)\prod_{d=1}^{D} P(X_{i,d}|S_i)$$

Creating network structure

```
# S pointing to all variables
IVs = paste0("[", colnames(df_tr)[-which(colnames(df_tr)=='S')], "|S]", collapse = "")
naive_bayes = model2network(paste0("[S]", IVs))
plot(naive_bayes)
```



Exact inference

Note: This time we only use *Exact inference*.

The confusion matrix looks exactly the same as before. This seems expected since variables that are not in the Markov blanket of `S` only have a rather small relationship with `S` and hence don't help with the classification.

```
# Fit the parameters (given structure and data set)
dag_fitted = bn.fit(naive_bayes, df_tr, method = 'mle')

# Make predictions
```

```
res = predict_exact(dag_fitted, df_te, "S", markov = TRUE)
Y_true = res$Y_true
Y_pred = res$Y_pred
```

```
## Confusion matrix (absolute):
```

```
##      Predicted
## True   no yes
##   no  349 119
##   yes 188 344
```

```
##
## Confusion matrix (relative):
```

```
##      Predicted
## True     no   yes
##   no  0.349 0.119
##   yes 0.188 0.344
```

```
##
## Performance statistics:
```

MCR	Accuracy	Recall	Precision	FPR	TNR
0.307	0.693	0.6466165	0.7429806	0.2542735	0.7457265

5 Assignment 5: Explanation

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

Overall, we can see that the differences are very small for asg. 2-3. It seems like the nodes that are not in the markov blanket don't provide incremental value in the classification. This is because all variables outside the markov blanket are conditionally independent of `S` when conditioned on `MB(S)`, the markov blanket of `S` (https://en.wikipedia.org/wiki/Markov_blanket).

The result with naive bayes asg. 4 sticks out. This is because it assumes independence between all the features which is a strong assumption that does not seem to be fulfilled. This is reflected in the fact that the structures learned in asg. 2-3 look very different. Therefore, the results are slightly worse for naive bayes.

```
## [1] "asg2_exact"
```

```
##      Predicted
## True   no yes
##   no  322 146
##   yes 120 412
```

```
## [1] "asg2_approx"
```

```
##      Predicted
## True   no yes
##   no  322 146
##   yes 122 410
```

```
## [1] "asg2_true"
```

```
##      Predicted
## True   no yes
##   no  322 146
##   yes 120 412
```

```
## [1] "asg3_markov"
```

```
##      Predicted
## True   no yes
##   no  322 146
##   yes 120 412
```

```
## [1] "asg4_naivebayes"
```

```
##      Predicted
## True   no yes
##   no  349 119
##   yes 188 344
```

	MCR	Accuracy	Recall	Precision	FPR	TNR
asg2_exact	0.266	0.734	0.7744	0.7384	0.3120	0.6880
asg2_approx	0.268	0.732	0.7707	0.7374	0.3120	0.6880
asg2_true	0.266	0.734	0.7744	0.7384	0.3120	0.6880
asg3_markov	0.266	0.734	0.7744	0.7384	0.3120	0.6880
asg4_naivebayes	0.307	0.693	0.6466	0.7430	0.2543	0.7457

6 Appendix

```
# Set up general options

knitr::opts_chunk$set(echo = FALSE, warning = FALSE, message = FALSE,
                      fig.width=6, fig.height=5#, collapse=TRUE
                      )

set.seed(12345)
options(scipen=999)

# General libraries
library(ggplot2)
```

```r
# Specific packages
library("bnlearn") # ls('package:bnlearn') # to show all functions
library("gRain")
library('RBGL')


# Auxilary functions
analyze_cm = function(cm, true){

  stopifnot(true %in% colnames(cm))
  levels = c(true, colnames(cm)[-which(colnames(cm) == true)]) # ORDER: 1; 0
  cm = as.data.frame(cm); colnames(cm)[1:2] = c("True", "Pred")
  N = sum(cm$Freq)
  Npos = sum(cm$Freq[which(cm$True == levels[1])])
  Nneg = sum(cm$Freq[which(cm$True == levels[2])])
  TP = sum(cm$Freq[which(cm$True == levels[1] & cm$Pred == levels[1])])
  TN = sum(cm$Freq[which(cm$True == levels[2] & cm$Pred == levels[2])])
  FP = sum(cm$Freq[which(cm$True == levels[2] & cm$Pred == levels[1])])

  FN = sum(cm$Freq[which(cm$True == levels[1] & cm$Pred == levels[2])])
  return(data.frame(MCR = (FP+FN)/N, Accuracy = (TP + TN)/N,
                    Recall = TP/Npos, # recall = TPR = sensitivity,
                    Precision = TP/(TP + FP),
                    FPR = FP/Nneg, TNR = TN/Nneg)) # TNR = specificity
}

# cm = table(Y_true, Y_pred, dnn = c("True", "Predicted"))
# knitr::kable(analyze_cm(cm, true = "yes"))


# ------------------------------------------------------------------------
# Assignment 1
# ------------------------------------------------------------------------


# Prepare data
data("asia")
df = data.frame(asia)
rm(asia)
head(df)


# Non-equivalence due to different score metrics -----------------------------

# Train models
set.seed(12345)
```

```r
res1 = hc(df, restart=10, score='bic')
res2 = hc(df, restart=10, score='aic')
# cat("Score", bnlearn::score(res, df)) # smaller AIC, BIC is better

# Check equivalence: logically
arcs1 = arcs(cpdag(res1))
arcs2 = arcs(cpdag(res2))
identical(arcs1[order(arcs1[, 1]), ], arcs2[order(arcs2[, 1]), ])

# Check equivalence: visually
par(mfrow = c(1, 2))
graphviz.compare(cpdag(res1), cpdag(res2), main = c("BIC", "AIC"))


# Non-equivalence due to different initial random graphs ----------------------

# Train models
set.seed(12345)
rand = random.graph(colnames(df))
res1 = hc(asia, start = rand, restart = 100, score='bic')
rand = random.graph(colnames(df))
res2 = hc(asia, start = rand, restart = 100, score='bic')

# Check equivalence: logically
arcs1 = arcs(cpdag(res1))
arcs2 = arcs(cpdag(res2))
identical(arcs1[order(arcs1[, 1]), ], arcs2[order(arcs2[, 1]), ])

# Check equivalence: visually
par(mfrow = c(1, 2))
graphviz.compare(cpdag(res1), cpdag(res2), main = c("BIC", "AIC"))


# Train models
set.seed(12345)
res1 = hc(asia, restart = 10, score = 'bde', iss = 20)
res2 = hc(asia, restart = 10, score = 'bde', iss = 200)
# cat("Score", bnlearn::score(res, df)) # smaller AIC, BIC is better

# Check equivalence: logically
arcs1 = arcs(cpdag(res1))
arcs2 = arcs(cpdag(res1))
identical(arcs1[order(arcs1[, 1]), ], arcs2[order(arcs2[, 1]), ])

# Check equivalence: visually
par(mfrow = c(1, 2))
```

```r
graphviz.compare(cpdag(res1), cpdag(res2), main = c("BIC", "AIC"))


# Old solution Asg. 1 (eval = FALSE) ----------------------------------------

# Train models
res1 = hc(df, restart=10, score='bic')
res2 = hc(df, restart=10, score='bic')

# Overview and scores
all.equal(res1, res2)
cat("Score BN 1: ", bnlearn::score(res1, df)) # smaller BIC is better
cat("Score BN 2: ", bnlearn::score(res2, df))

# Visual Comparison of arcs
par(mfrow = c(1, 2))
graphviz.compare(res1, res2, main = c("Run 1 (with BIC)", "Run 2 (with BIC)"))

# Printed comparison of arcs
comp = compare(res1, res2)
cat("true positive (tp) arcs, which appear both in target and in current:", comp$tp)
cat("false positive (fp) arcs, which appear in current but not in target:", comp$fp)
cat("false negative (fn) arcs, which appear in target but not in current:", comp$fn)

# Hamming
cat('Same Skeleton:', all.equal(skeleton(res1), skeleton(res2)))
cat("Hamming distance (zero if same skeleton):", hamming(res1, res2))
cat("Struct. hamming distance:", shd(res1, res2))

# Other ----------

# # Create individual plots for each dag
# plot(res1)
# plot(res2)

# # Print the arcs (from, to) for each dag
# arcs(res1)
# arcs(res12

# vstructs(res1)
# vstructs(res2)

# cpdag(res1)
# cpdag(res2)

# Alternatives ----------
```

```r
# rand = random.graph(colnames(df))
# res4 = hc(asia, start = rand)


# ------------------------------------------------------------------------
# Assignment 2
# ------------------------------------------------------------------------


# Prepare data
data("asia")
df = asia
#df = data.frame(apply(asia, 2, as.character), stringsAsFactors = F)
rm(asia)

# Split data randomly
set.seed(12345)
N = nrow(df)
idx = sample(1:N, round(N * 0.8))
df_tr = df[idx, ]
df_te = df[-idx, ]


# Fit the structure (given data set)
dag = hc(df_tr, restart=100, score='bic')
# dag = iamb(df_tr)

# Fit the parameters (given structure and data set)
dag_fitted = bn.fit(dag, df_tr, method='mle')


# Show model
plot(dag)

# Show conditional probabilities for S
dag_fitted$S
bn.fit.dotplot(dag_fitted$S) # bn.fit.barchart(dag_fitted$S)


predict_exact = function(dag_fitted, df_te, y_name, markov = FALSE){

  # ------------------------------------------------------------------------
  # Input:
  # - dag_fitted: dag with learned structure and parameters
  # - df_te: data frame with all features and the Y (all factors)
  # - y_name: string with the name of the Y variable
```

```r
  # Output:
  # - list with factor vectors: Y_true and Y_pred
  # ------------------------------------------------------------------------

  junction = compile(as.grain(dag_fitted))

  N = nrow(df_te)
  Y_true = df_te[[y_name]]
  Y_pred = numeric(length = N)
  Y_space = as.character(unique(Y_true))

  for (i in 1:N){

    # With all RVs
    x = df_te[i, colnames(df_te) != y_name]

    if (markov == FALSE){
      var_names = names(x)
      var_values = vapply(x, as.character, character(1))
    } else {
      var_names = mb(dag_fitted, y_name)
      var_values = vapply(x, as.character, character(1))[var_names]
    }
    evidence = setEvidence(junction, nodes = var_names, states = var_values)

    Y_prob = querygrain(evidence, nodes = y_name)[[1]]
    Y_pred[i] = Y_space[which.max(Y_prob)]

  }

  return(list(Y_pred = Y_pred, Y_true = Y_true))

}


# Make predictions
res = predict_exact(dag_fitted, df_te, "S")
Y_true = res$Y_true
Y_pred = res$Y_pred


# Evaluate predictions

cat("Confusion matrix (absolute):\n")
cm = table(Y_true, Y_pred, dnn = c("True", "Predicted"))
print(cm)
```

```r
asg2_exact = cm

cat("\nConfusion matrix (relative):\n")
cm/sum(cm)

cat("\nPerformance statistics:\n")
knitr::kable(analyze_cm(cm, true = "yes"))


# Alternative: The below code could be used if we only wanted to use node "B"
# for prediction

# # With only B as RV
# val = as.character(df_te[i, 'B'])
# evidence = setEvidence(junction, nodes = "B", states = val)


# ------------------------------------------------------------------------
# Input:
# - dag_fitted: dag with learned structure and parameters
# - df_te: data frame with all features and the Y (all factors)

# MAKE SURE TO ADJUST THE EVIDENCE AND EVENT!
# ------------------------------------------------------------------------

N = nrow(df_te)
Y_true = df_te[['S']]
Y_pred = numeric(length = N)
Y_space = as.character(unique(Y_true))
Y_prob = numeric(length = length(Y_space))
junction = compile(as.grain(dag_fitted))

for (i in 1:N){

  # With all RVs
  x = vapply(df_te[i, colnames(df_te) != 'S'], as.character, character(1))

  for (j in 1:length(Y_space)){
    #var_names = names(x)
    #var_values = x
    Y_prob[j] = cpquery(dag_fitted, event = (S == Y_space[j]),
                        # evidence = setEvidence(junction, nodes = var_names,
                        #                        states = var_values)
                          evidence = ((A == x['A']) & (T == x['T'])
                                      & (L == x['L']) & (B == x['B'])
                                      & (E == x['E']) & (X == x['X'])
```

```
                                        & (D == x['E']))
                        )
  }

  Y_pred[i] = Y_space[which.max(Y_prob)]
}



# Evaluate predictions

cat("Confusion matrix (absolute):\n")
cm = table(Y_true, Y_pred, dnn = c("True", "Predicted"))
print(cm)
asg2_approx = cm

cat("\nConfusion matrix (relative):\n")
cm/sum(cm)

cat("\nPerformance statistics:\n")
knitr::kable(analyze_cm(cm, true = "yes"))


# Prepare the structure

# Option 1:
dag_true = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

# #Option 2
# dag_true = empty.graph(names(df))
# modelstring(dag_true) = "[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]"

# Visual comparison only:
par(mfrow = c(1, 2))
graphviz.compare(dag, dag_true, main = c("Learned", "True"))


# Fit the parameters (given structure and data set)
dag_fitted = bn.fit(dag_true, df_tr, method='mle')

# Make predictions
res = predict_exact(dag_fitted, df_te, "S")
Y_true = res$Y_true
Y_pred = res$Y_pred
```

```r
# Evaluate predictions

cat("Confusion matrix (absolute):\n")
cm = table(Y_true, Y_pred, dnn = c("True", "Predicted"))
print(cm)
asg2_true = cm

cat("\nConfusion matrix (relative):\n")
cm/sum(cm)

cat("\nPerformance statistics:\n")
knitr::kable(analyze_cm(cm, true = "yes"))


# ------------------------------------------------------------------------
# Assignment 3
# ------------------------------------------------------------------------


# Fit the structure (given data set)
dag = hc(df_tr, restart=10, score='bic')

# Fit the parameters (given structure and data set)
dag_fitted = bn.fit(dag, df_tr, method='mle')

# Make predictions
res = predict_exact(dag_fitted, df_te, "S", markov = TRUE)
Y_true = res$Y_true
Y_pred = res$Y_pred



# Evaluate predictions

cat("Confusion matrix (absolute):\n")
cm = table(Y_true, Y_pred, dnn = c("True", "Predicted"))
print(cm)
asg3_markov = cm

cat("\nConfusion matrix (relative):\n")
cm/sum(cm)

cat("\nPerformance statistics:\n")
knitr::kable(analyze_cm(cm, true = "yes"))
```

```r
# ------------------------------------------------------------------------
# Assignment 4
# ------------------------------------------------------------------------


# S pointing to all variables
IVs = paste0("[", colnames(df_tr)[-which(colnames(df_tr)=='S')], "|S]", collapse = "")
naive_bayes = model2network(paste0("[S]", IVs))
plot(naive_bayes)


# # All variables pointing to S (wrong)
# cond = paste0(colnames(df_tr)[-which(colnames(df_tr)=='S')], ":", collapse = "")
# cond = substr(cond, 1, nchar(cond)-1)
# IVs = paste0("[", colnames(df_tr)[-which(colnames(df_tr)=='S')], "]", collapse = "")
# naive_bayes = model2network(paste0("[S|", cond, "]", IVs))
# plot(naive_bayes)



# Fit the parameters (given structure and data set)
dag_fitted = bn.fit(naive_bayes, df_tr, method = 'mle')

# Make predictions
res = predict_exact(dag_fitted, df_te, "S", markov = TRUE)
Y_true = res$Y_true
Y_pred = res$Y_pred


# Evaluate predictions

cat("Confusion matrix (absolute):\n")
cm = table(Y_true, Y_pred, dnn = c("True", "Predicted"))
print(cm)
asg4_naivebayes = cm

cat("\nConfusion matrix (relative):\n")
cm/sum(cm)

cat("\nPerformance statistics:\n")
knitr::kable(analyze_cm(cm, true = "yes"))


# ------------------------------------------------------------------------
# Assignment 5
# ------------------------------------------------------------------------
```

```r
print("asg2_exact")
asg2_exact

print("asg2_approx")
asg2_approx

print("asg2_true")
asg2_true

print("asg3_markov")
asg3_markov

print("asg4_naivebayes")
asg4_naivebayes


knitr::kable(round(rbind(asg2_exact = analyze_cm(asg2_exact, true = "yes"),
      asg2_approx = analyze_cm(asg2_approx, true = "yes"),
      asg2_true = analyze_cm(asg2_true, true = "yes"),
      asg3_markov = analyze_cm(asg3_markov, true = "yes"),
      asg4_naivebayes = analyze_cm(asg4_naivebayes, true = "yes")), 4))
```