Lab 1 - Graphical Models

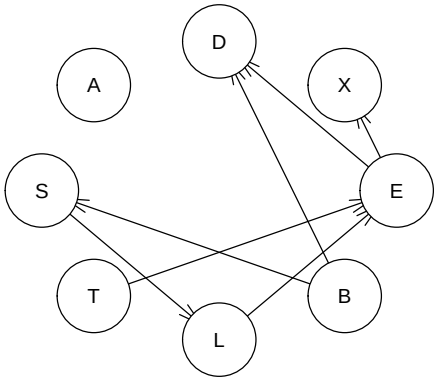
Alice Velander

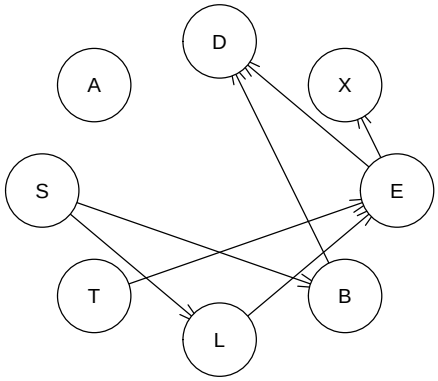
13/9/2020

```
knitr::opts_chunk$set(echo = TRUE)
```

Task 1 - Show that multiple runs of the hill-climbing algorithm can return non-equivalent Bayesian network (BN) structures.

```
#install.packages("bnlearn")
library(bnlearn)

data("asia")
amount=5
network = list()

#save different networks created by the hillclimbing method in a list
for (i in 1:amount){
  climb = hc(asia, start=NULL, restart = 10, optimized = FALSE)
  network[[i]] = climb
}

par(mfrow=c(1,1))
#compare the networks to each other
stop=FALSE
for (i in 1:(amount-1)){
  for (j in (i+1):(amount)){
    if( all.equal(network[[i]], network[[j]]) != TRUE ){
      plot(network[[j]])
      plot(network[[i]])
      stop = TRUE
      break
    }
  }
  if (stop){break}
}
```



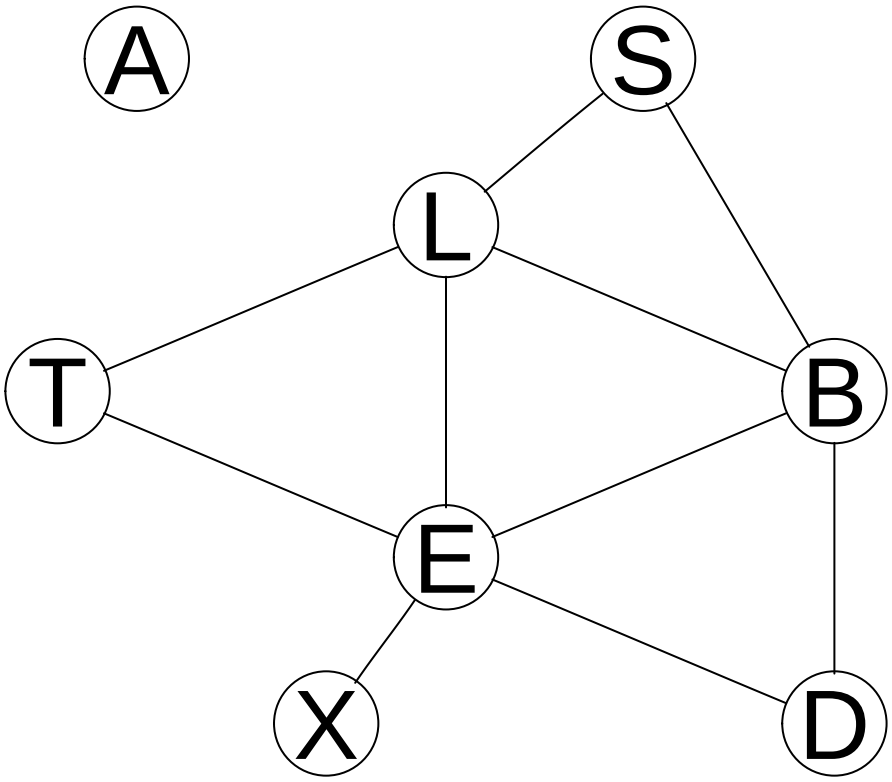


The hill-climbing method starts with an empty DAG and add/remove/reverse edges given that it increases the total log bayesian score. The reason why HC results in different DAGs is because somethimes directions of an edge doesn't give higher or lower score, since it's the same statistical model. This results in a randomized chosen direction of edge. We get different local optimum, and results in different networks in different iterations.

Task 2 - Use the BN learned to classify the remaining 20 % of the Asia dataset in two classes: S = yes and S = no.

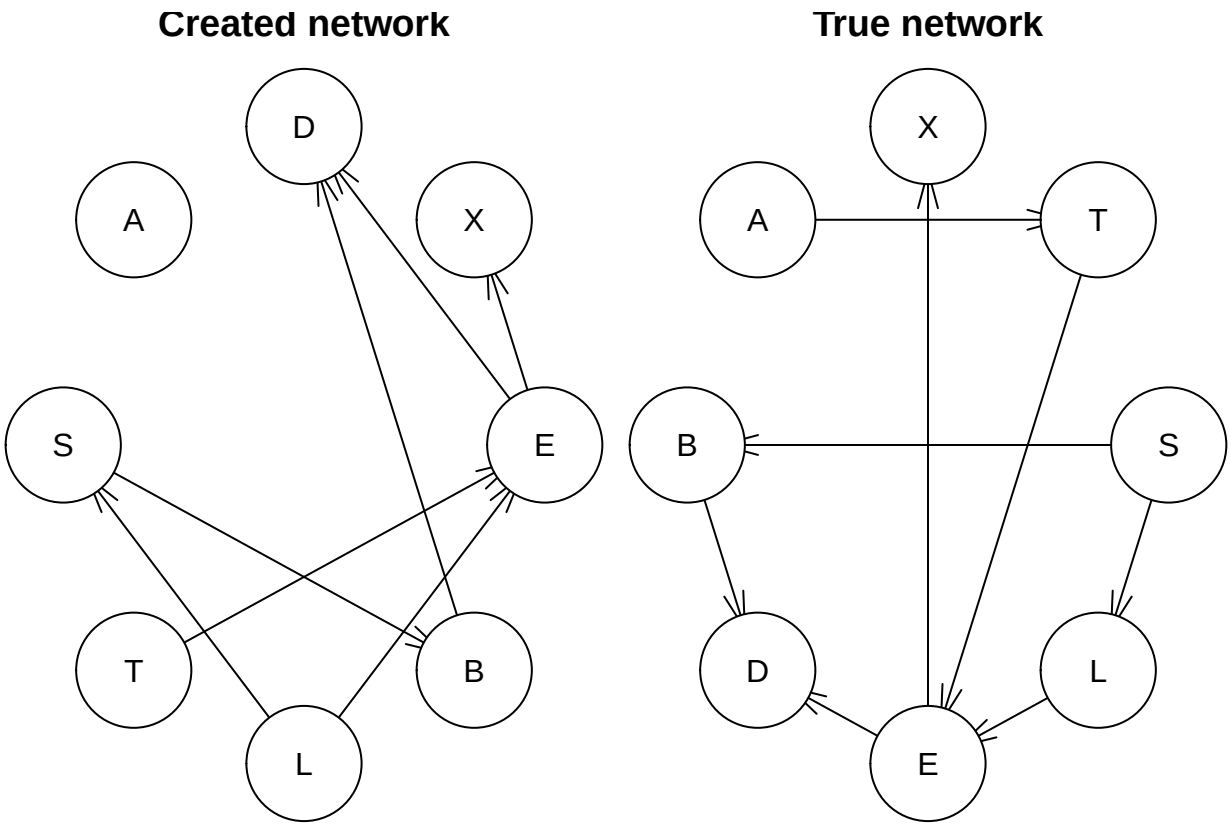
```
dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

#split dataset into traning and test
n=dim(asia)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.8))

train=asia[id,]
test=asia[-id,]

#install.packages("gRain")
library(gRain)

#create network from train
network = hc(train, start=NULL, restart = 10, optimized = FALSE)
#learn parameters for created network, based on same dataset,
#get conditional probabiities in BN - needed for cliques
network_parameters = bn.fit(network, data = train)
#parameters for created network, see dependencies in matrices below
#coefficients(network_parameters)

#find potentials for cliques, checking all independencies
```

```r
#Moralize, triangulaize - we get MN - Graphical independence network (grain)
graphical_Ind_network = as.grain(network_parameters)

#Create junction tree - R, seperators - needed to estimate potential cliques - information on how nodes
junc_tree = compile(graphical_Ind_network)
#We get our final Markov Network:
plot(junc_tree)
```




```r
#Classification for testdata!
pred_node=c("S")
answer = test[,"S"]
observed = test[,-2]
nodes = c("A", "T", "L", "B", "E", "X", "D")

missclass_rate = function(table){
  return(1-sum(diag(table))/sum(table))
}

predict_BN=function(network, nodes, pred_nodes, answer, observed){
  classify=c()
  nrows=nrow(observed)
  for (i in 1:nrows){
    #Sorry for fulkod
    obs = as.character(observed[i,])
    values=c()
```

```
    for (j in 1:length(obs)){
      if(obs[j]==1){
        values=c(values, "no")
      }else{
        values= c(values,"yes")
      }
    }

    #predict node S
    #Given observations - we update clique potentials
    evid = setEvidence(network, nodes = nodes, states = values)
    #Given new clique potential - we predict a specific node.
    pred = querygrain(evid, nodes=pred_node)

    if(pred$S[1] > 0.5){ #[1] = predict is NO
      classify = c(classify, "no")
    } else {
      classify = c(classify, "yes")
    }
  }

  #Get confusion matrix
  conf_matrix=table(answer, classify)

  return(confusionmatrix=conf_matrix)
}

par(mfrow=c(1,2))
plot(network, main="Created network")
plot(dag, main="True network")
```




```
conf_matrix=predict_BN(junc_tree,nodes,pred_node, answer, observed)
#Confusionmatrix for the created network:
print(conf_matrix)
```

```
##        classify
## answer  no yes
##    no  337 176
##    yes 121 366
```

```
#Misclassificationrate for the created network:
missclass_rate(conf_matrix)
```

```
## [1] 0.297
```

```
#compare results with given dag
dag_parameters = bn.fit(dag, data = train)
graphical_Ind_dag = as.grain(dag_parameters)
dag_tree = compile(graphical_Ind_dag)


conf_matrix_true=predict_BN(dag_tree,nodes,pred_node, answer, observed) #We get the same results!

#Confusionmatrix network for the true network:
print(conf_matrix_true)
```

```
##       classify
## answer  no yes
##    no  337 176
##    yes 121 366
```

```
#Misclassificationrate for the true network:
missclass_rate(conf_matrix_true)
```

```
## [1] 0.297
```

We get the same results (confusionmatrix and masclassification rate). Given the markov network, S is independenten to the remaining nodes in the network. Comparing the networks, S has the same markov blanket for both networks. Since the markov blanket is "the seperators" from the rest of the network, it only dependes on those specific nodes. This gives us the same results, with the two different networks.

Task 3 - Classify S given observations only for the so-called Markov blanket

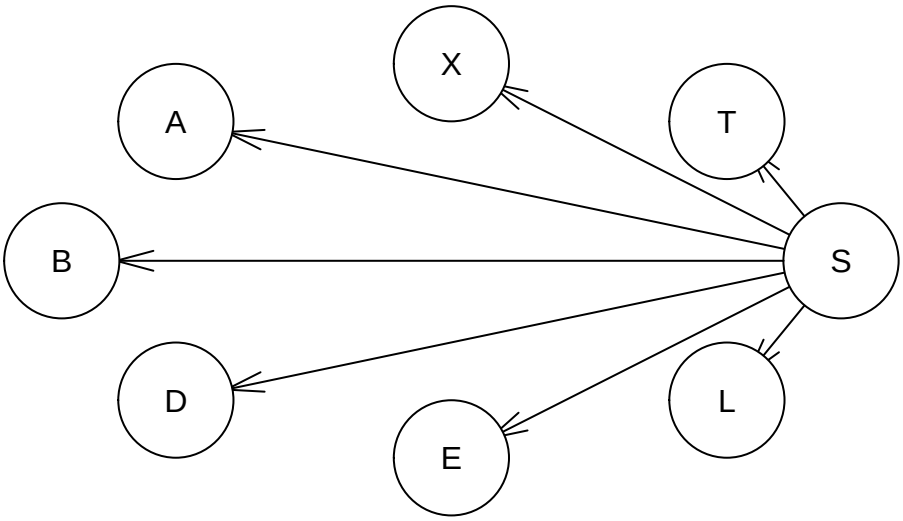
```
#Classify according to Markov blanket.

#Get blanket
blanket = mb(network, node=c("S"))
#take all values from blanket in observations
observed_mb = observed[,blanket]
#Get exact same results, since the node only depends on the values in the blanket:)
#Confusionmatrix network for the MB network:
MB_conf_matrix = predict_BN(junc_tree, blanket, pred_node, answer, observed_mb)
#Misclassification rate for the MB network:
missclass_rate(MB_conf_matrix)
```

```
## [1] 0.297
```

We get the same confusionmatrices and misclassification rate, since S only depends on the variables in the market blanket in the network. The blanket can be seen as the variables/nodes that a specific node depends on in a given network. This makes computation easier, since less computation is needed in order to predict the node S.

Task 4 - Classify S using a naive Bayes classifier.

```
#We assume independence between the nodes (except S) since we use Bayes Classifier
BN_dag = model2network("[S][A|S][T|S][L|S][B|S][E|S][X|S][D|S]")
plot(BN_dag)
```




```
#compare results with given BN dag
BN_dag_parameters = bn.fit(BN_dag, data = train)
graphical_BN_dag = as.grain(BN_dag_parameters)
BN_dag_tree = compile(graphical_BN_dag)

conf_matrix_BN = predict_BN(BN_dag_tree,nodes,pred_node, answer, observed)
#confusionmatrix for the NB network:
print(conf_matrix_BN)
```

```
##       classify
## answer  no yes
##    no  359 154
##    yes 180 307
```

```
#Misclassification rate for the NB network:
missclass_rate(conf_matrix_BN)
```

```
## [1] 0.334
```

We get an higher missclassification rate, and therefore less correct predictions/result of S. This is because of the we have a different markov blanket for S, and the assumption of independencies between the other nodes in the network, when using the Bayesian classifier. Trade-off is a simpler model with less computation, but a worsen result in prediction.

Task 5 - Explain why you obtain the same or different results in the exercises (2-4).

In exercise 2-3 we obtain the same results, this is due to all networks having the same Markov blanket, which are the nodes S depend on. In Exercise 5 we have a different markov blanket, with a naive assumption of independence between nodes in the network (except for S), which gives us a higher misclassification rate (and less accurate predictions).