Bayesian networks with R

Bojan Mihaljević

November 22-23, 2018

Contents

Introduction **2**
Overview . . . 2
Bayesian networks in R . . . 2

Get code and data **2**

`bnlearn` Bayesian network models **2**

The earthquake network **11**

Representation **12**
Compact representation . . . 12
Independence . . . 12
Conditional independence (d-separation) . . . 12
Markov blanket . . . 12
Equivalence classes . . . 13

Reasoning **14**
Probability queries with no evidence . . . 15
Reasoning patterns: Causal, evidential and intercausal . . . 16
Maximum a posteriori (MAP) queries . . . 19

Inference **20**
The Asia network . . . 21
Conditional probability queries . . . 22
Junction tree message passing . . . 22
Logic sampling . . . 24
Likelihood weighting . . . 26
The marks data set . . . 27
Gaussian networks . . . 27

Learning from data **30**
Breast cancer data . . . 30
Score + search algorithms . . . 31
Constraint-based structure learning . . . 35
Mixed data . . . 37
Latent variables . . . 38
Parameter estimation . . . 40

Classification **43**
The car data set . . . 43
Naive Bayes . . . 44
Irrelevant features . . . 46
Correlated features . . . 47
One-dependence augmented naive Bayes . . . 47
Comparing classifiers . . . 51

Wrapper structure learning 51
Markov blanket 55

The ALARM network **56**
Inspect the network 57
Recover network structure 57
Improve learning 57
Inference 58

Final notes **58**
Some useful functions 58
Books 59
Other Bayesian networks software 59

Introduction

Overview

- Representation, reasoning, inference, learning, classifiers
- Datasets/networks: earthquake, asia, marks, breast cancer, car, alarm
- Mainly discrete variables, also Gaussian

Bayesian networks in R

- bnlearn:
  - constraint-based and score + search learning, approximate inference
  - utilities: moralize, make random dag
  - efficient
- gRain:
  - exact inference
  - not very efficient
- gRbase
  - graph utilities: moralize, make random dag
  - largely replaced by bnlearn
- bnclassify: classifiers
- others: pcalg, ggm
- graphical models task view: https://cran.r-project.org/web/views/gR.html

Get code and data

```r
#install.packages("usethis")
usethis::use_course("https://goo.gl/x9rdpD")
# usethis::use_course("https://github.com/bmihaljevic/intro-bns/raw/master/intro-bns.zip")
```

`bnlearn` Bayesian network models

We start a clean R session and load the `bnlearn` package

```
library(bnlearn)
```

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

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

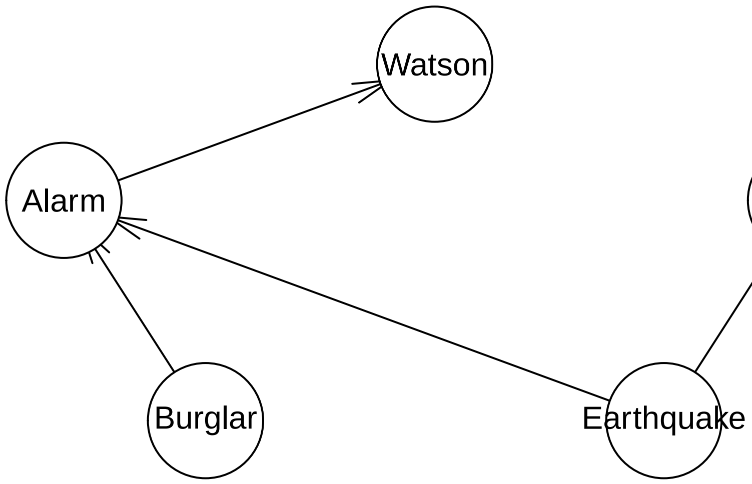
```
?bnlearn
```

bnlearn provides two S3 classes for Bayesian networks:

- `"bn"`: the underlying DAG
- `"bn.fit"`: a fully specified network with parameters

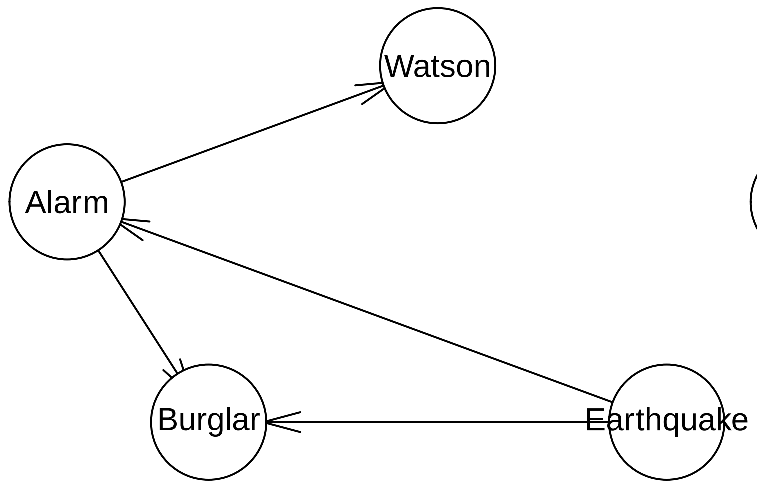
A compact way to specify a `"bn"` is the model string:

```
bl.alarm <- model2network('[Burglar][Earthquake][Alarm|Burglar:Earthquake][News|Earthquake][Watson|Alar
plot(bl.alarm)
```



We can add, remove or reverse arcs, with the acyclic constraint enforced:

```
modified.alarm <- drop.arc(bl.alarm, "Earthquake", "News")
modified.alarm <- reverse.arc(modified.alarm , "Alarm", "Burglar")
modified.alarm <- set.arc(modified.alarm, "Earthquake", "Burglar")
plot(modified.alarm)
```



```r
modified.alarm <- set.arc(modified.alarm, "Watson", "Earthquake")
```

```
## Error in arc.operations(x = x, from = from, to = to, op = "set", check.cycles = check.cycles, : the
```

Printing the object to console provides basics information:

```r
bl.alarm
```

```
##
##   Random/Generated Bayesian network
##
##   model:
##    [Burglar][Earthquake][Alarm|Burglar:Earthquake][News|Earthquake]
##    [Watson|Alarm]
##   nodes:                                 5
##   arcs:                                  4
##     undirected arcs:                     0
##     directed arcs:                       4
##   average markov blanket size:           2.00
##   average neighbourhood size:            1.60
##   average branching factor:              0.80
##
##   generation algorithm:                  Empty
```

When the network is too large for plotting, the model string can be useful. We can also ask about the nodes close to a particular node:

```r
nbr(bl.alarm, node = 'Alarm')
```

```
## [1] "Burglar"    "Earthquake" "Watson"
```

```r
parents(bl.alarm, node = 'Alarm')
```

```
## [1] "Burglar"    "Earthquake"
```

```r
children(bl.alarm, node = 'Alarm')
```

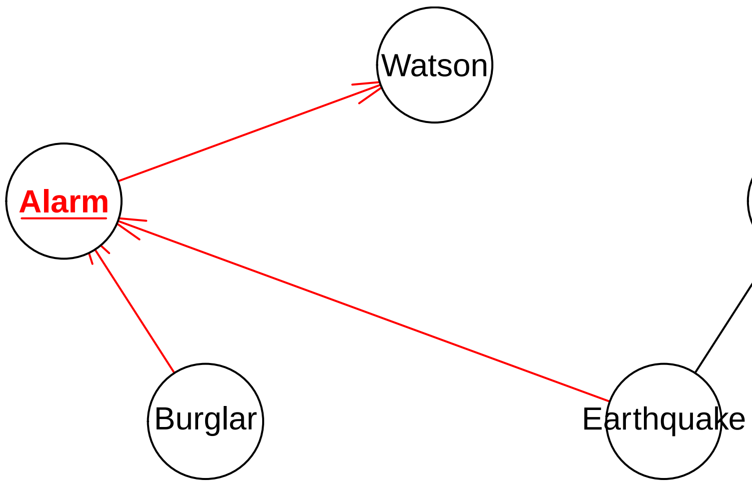
```
## [1] "Watson"
```

```r
mb(bl.alarm, node = 'Alarm')
```

```
## [1] "Burglar"    "Earthquake" "Watson"
```

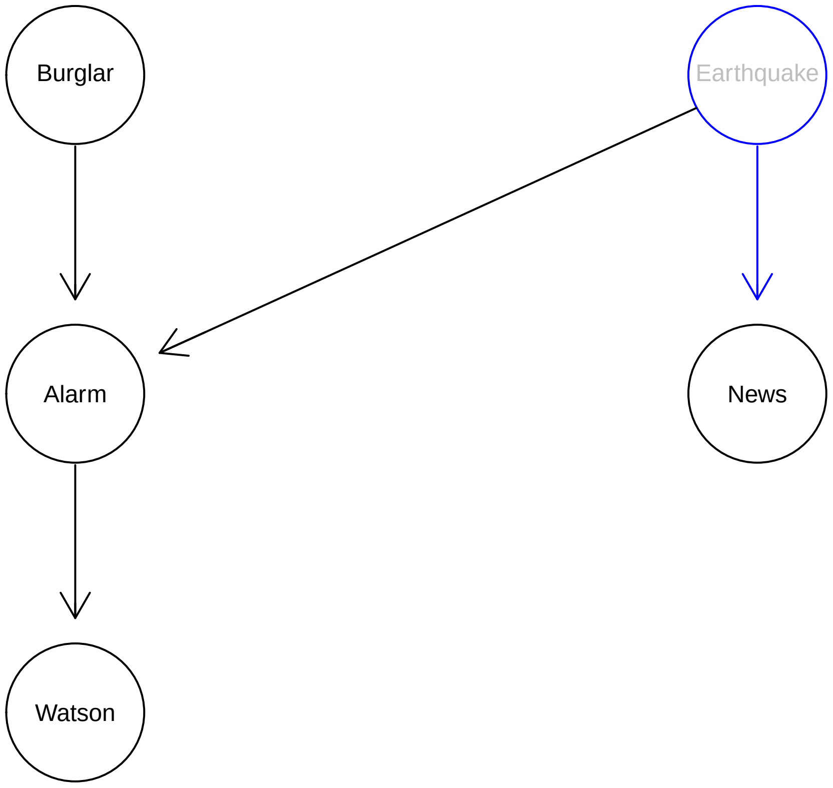
We can somewhat customize the network plot directly

```r
plot(bl.alarm, highlight = list(nodes='Alarm'))
```



Or use `Rgraphviz` for more advanced options:

```r
hlight <- list(nodes = c("Earthquake"), arcs = c("Earthquake", "News"), col = "blue", textCol = "grey")
pp <- graphviz.plot(bl.alarm, highlight = hlight)
Rgraphviz::renderGraph(pp)
```



We can list all arcs or ask for a path between a pair of nodes:

```r
arcs(bl.alarm)
```

```
##      from         to
## [1,] "Burglar"    "Alarm"
## [2,] "Earthquake" "Alarm"
## [3,] "Earthquake" "News"
## [4,] "Alarm"      "Watson"
```

```
path(bl.alarm, from = "Burglar", to = "Watson")
```

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

```
path(bl.alarm, from = "Watson", to = "Burglar")
```

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

We can check for d-separation or obtain the CPDAG:

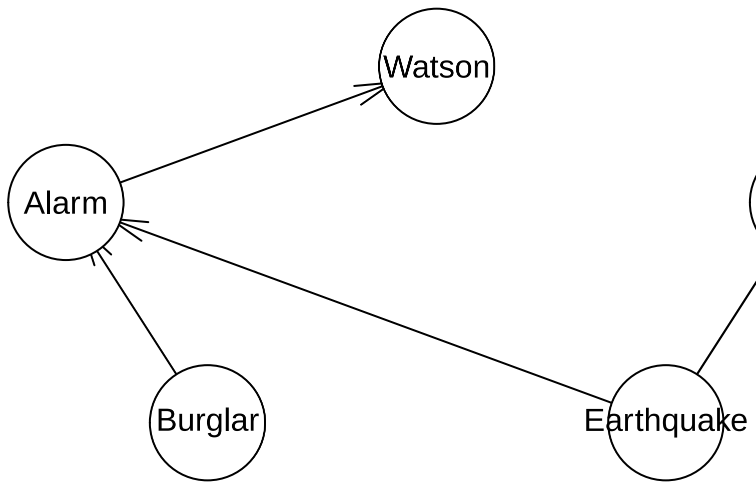
```
dsep(bl.alarm, 'Watson', 'News')
```

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

```
dsep(bl.alarm, 'Watson', 'News', 'Alarm')
```

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

```
plot(cpdag(bl.alarm))
```



Passing a list of local distributions to `custom.fit()` produces a `"bn.fit"`. With discrete variables, we can specify a CPT for each node:

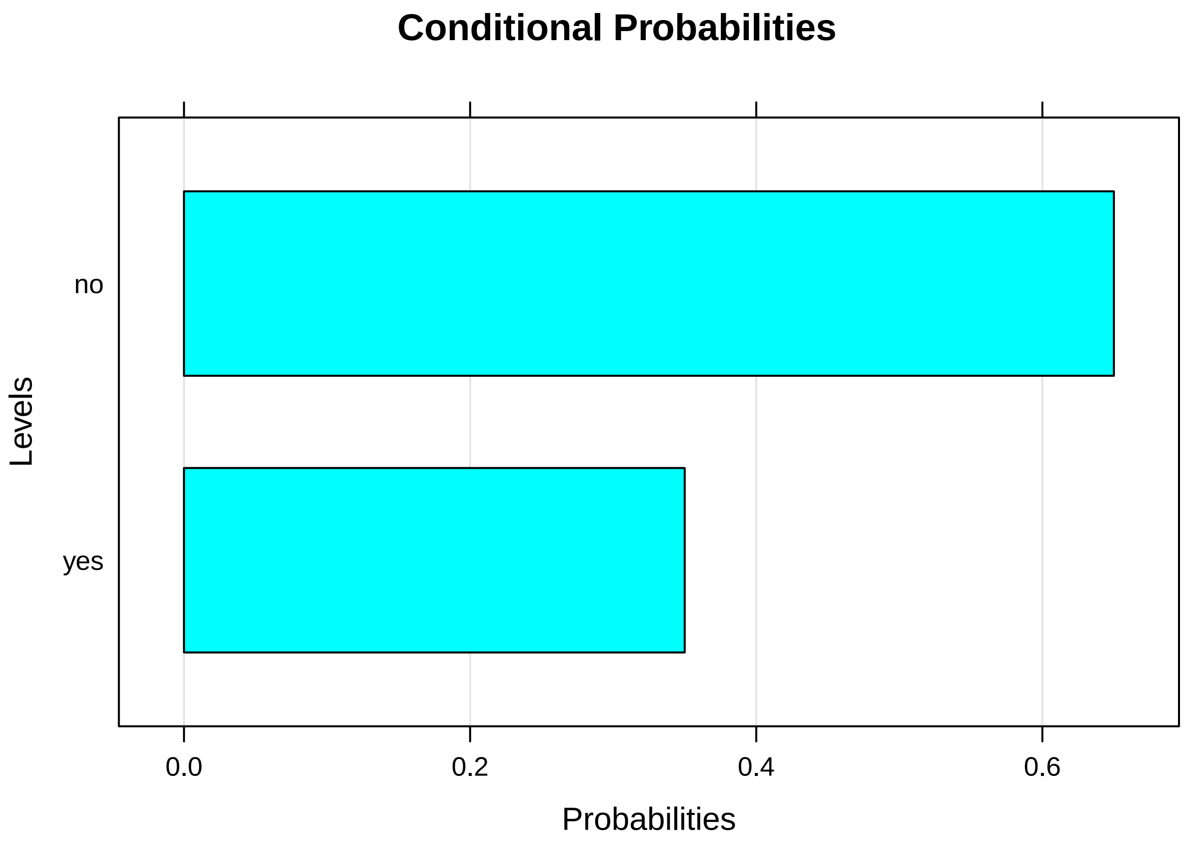
```
yn <- c("yes","no")
B  <- array(dimnames = list(Burglar = yn), dim = 2, c(0.30,0.70))
E  <- array(dimnames = list(Earthquake = yn), dim = 2, c(0.35,0.65))
A  <- array(dimnames = list(Alarm = yn, Earthquake = yn, Burglar = yn), dim = c(2, 2, 2),
            c(0.95,0.05,0.90,0.10,0.60,0.40,0.01,0.99))
W  <- array(dimnames = list(Watson = yn, Alarm = yn), dim = c(2, 2), c(0.80,0.20,0.40,0.60))
N  <- array(dimnames = list(News = yn, Earthquake = yn), dim = c(2, 2), c(0.60,0.40,0.01,0.99))
cpts <- list(Burglar = B, Earthquake = E, Alarm = A, Watson = W, News = N)
bl.alarm.fit = custom.fit(bl.alarm, cpts)
```

```
bl.alarm.fit$Earthquake
```

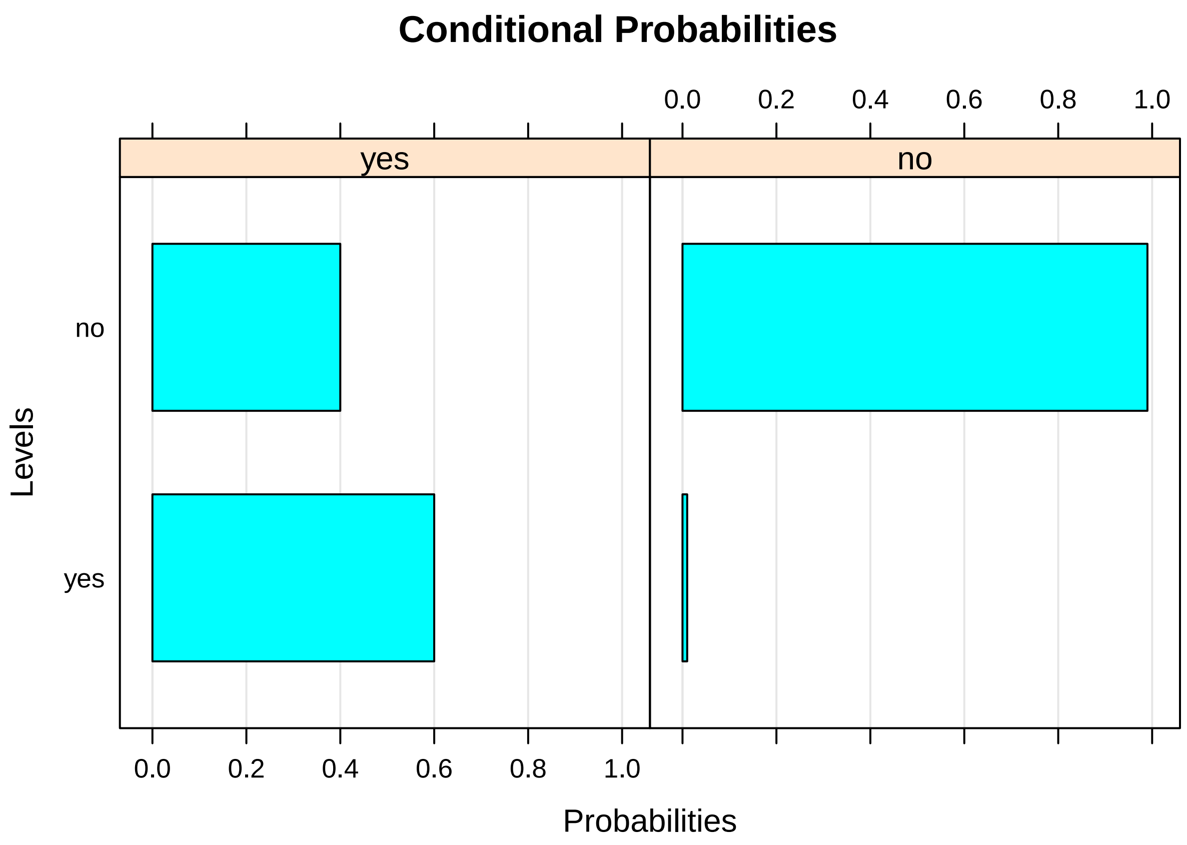
```
##
##   Parameters of node Earthquake (multinomial distribution)
##
## Conditional probability table:
##  Earthquake
##  yes   no
## 0.35 0.65
```

```
bn.fit.barchart(bl.alarm.fit$Earthquake)
```

```
## Loading required namespace: lattice
```



```
bn.fit.barchart(bl.alarm.fit$News)
```



One can load a Bayesian network model from `bnlearn`'s repository:

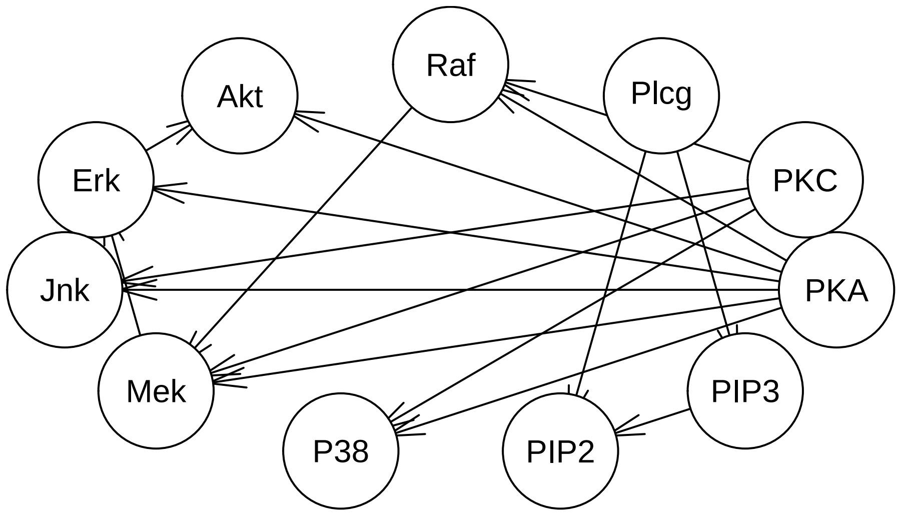
http://www.bnlearn.com/bnrepository/

```
load(url("http://www.bnlearn.com/bnrepository/sachs/sachs.rda"))
sachs.fit <- bn; rm(bn)
```

```
plot(sachs.fit)
```

```
## Error in xy.coords(x, y, xlabel, ylabel, log): 'x' is a list, but does not have components 'x' and '
```

```
plot(bn.net(sachs.fit))
```

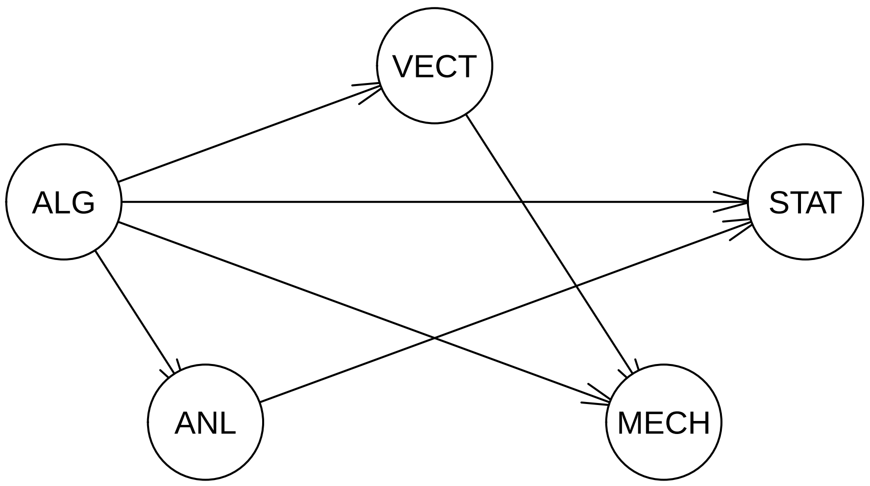


```r
bn.net(sachs.fit)
```

```
##
##   Random/Generated Bayesian network
##
##   model:
##    [PKC][Plcg][PIP3|Plcg][PKA|PKC][Jnk|PKA:PKC][P38|PKA:PKC]
##    [PIP2|PIP3:Plcg][Raf|PKA:PKC][Mek|PKA:PKC:Raf][Erk|Mek:PKA][Akt|Erk:PKA]
##   nodes:                                 11
##   arcs:                                  17
##     undirected arcs:                     0
##     directed arcs:                       17
##   average markov blanket size:           3.09
##   average neighbourhood size:            3.09
##   average branching factor:              1.55
##
##   generation algorithm:                  Empty
```

A Gaussian variable is given by a linear regression. The mean is a function of the parents. Consider the marks network:

```r
marks.dag = model2network("[ALG][ANL|ALG][MECH|ALG:VECT][STAT|ALG:ANL][VECT|ALG]")
plot(marks.dag)
```



Specify the regression coefficients and the standard deviation of the residuals:

```r
ALG.dist = list(coef = c("(Intercept)" = 50.60), sd = 10.62)
ANL.dist = list(coef = c("(Intercept)" = -3.57, ALG = 0.99), sd = 10.5)
MECH.dist = list(coef = c("(Intercept)" = -12.36, ALG = 0.54, VECT = 0.46), sd = 13.97)
STAT.dist = list(coef = c("(Intercept)" = -11.19, ALG = 0.76, ANL = 0.31), sd = 12.61)
VECT.dist = list(coef = c("(Intercept)" = 12.41, ALG = 0.75), sd = 10.48)
ldist = list(ALG = ALG.dist, ANL = ANL.dist, MECH = MECH.dist,
STAT = STAT.dist, VECT = VECT.dist)
marks.fit = custom.fit(marks.dag, ldist)
```

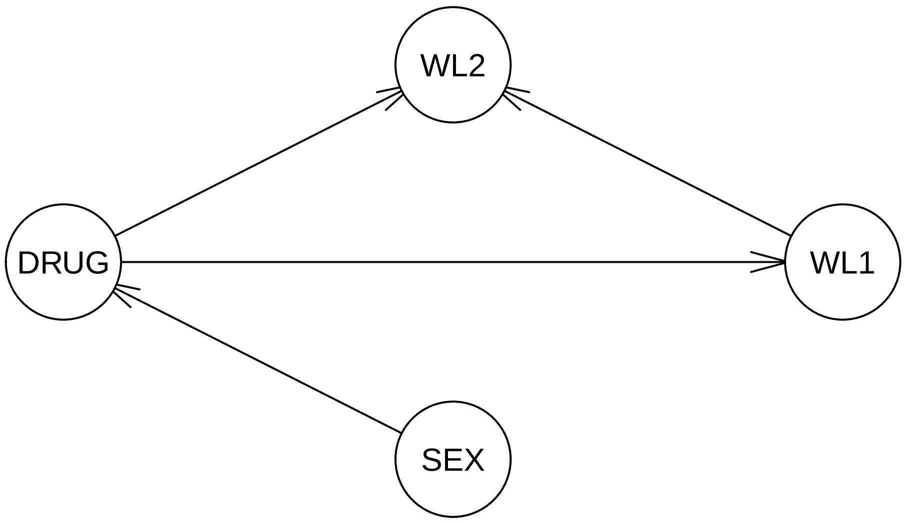
```r
marks.fit[c("MECH", "STAT")]
```

```
## $MECH
##
##   Parameters of node MECH (Gaussian distribution)
##
```

```
## Conditional density: MECH | ALG + VECT
## Coefficients:
## (Intercept)          ALG         VECT
##      -12.36         0.54         0.46
## Standard deviation of the residuals: 13.97
##
## $STAT
##
##   Parameters of node STAT (Gaussian distribution)
##
## Conditional density: STAT | ALG + ANL
## Coefficients:
## (Intercept)          ALG          ANL
##      -11.19         0.76         0.31
## Standard deviation of the residuals: 12.61
```

For conditional linear Gaussian models, consider the rats network, with a discrete variable `DRUG` and `SEX` and real-valued `WL1` and `WL2` (weight loss in week one and week two):

```
rats.dag = model2network("[SEX][DRUG|SEX][WL1|DRUG][WL2|WL1:DRUG]")
plot(rats.dag)
```



For real-valued variables, we have a conditional regression for each combination of discrete parents. For the discrete, we have CPTs.

```
SEX.lv = c("M", "F")
DRUG.lv = c("D1", "D2", "D3")
SEX.prob = array(c(0.5, 0.5), dim = 2, dimnames = list(SEX = SEX.lv))
DRUG.prob = array(c(0.3333, 0.3333, 0.3333, 0.3333, 0.3333, 0.3333),
dim = c(3, 2), dimnames = list(DRUG = DRUG.lv, SEX = SEX.lv))
WL1.coef = matrix(c(7, 7.50, 14.75), nrow = 1, ncol = 3,
dimnames = list("(Intercept)", NULL))
WL1.dist = list(coef = WL1.coef, sd = c(1.58, 0.447, 3.31))
WL2.coef = matrix(c(1.02, 0.89, -1.68, 1.35, -1.83, 0.82), nrow = 2, ncol = 3,
dimnames = list(c("(Intercept)", "WL1")))
WL2.dist = list(coef = WL2.coef, sd = c(1.78, 2, 1.37))
ldist = list(SEX = SEX.prob, DRUG = DRUG.prob, WL1 = WL1.dist, WL2 = WL2.dist)
rats.fit = custom.fit(rats.dag, ldist)
```

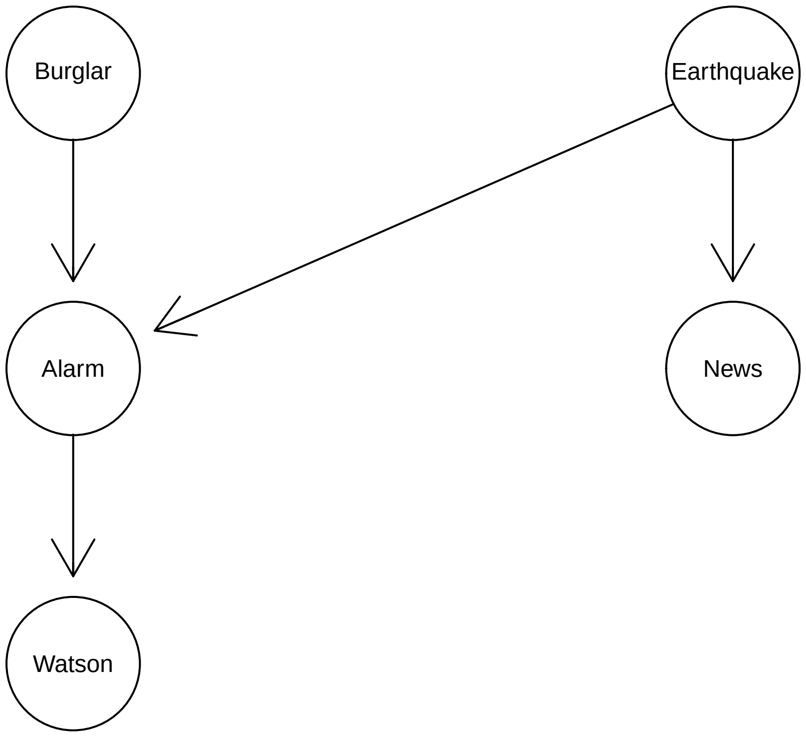
A column corresponds to a configuration of the discrete parents and a row to one of the continuous parents.

```
rats.fit$WL2
```

```
##
##   Parameters of node WL2 (conditional Gaussian distribution)
##
## Conditional density: WL2 | DRUG + WL1
## Coefficients:
##                   0      1      2
## (Intercept)   1.02  -1.68  -1.83
## WL1           0.89   1.35   0.82
## Standard deviation of the residuals:
##    0     1     2
## 1.78  2.00  1.37
## Discrete parents' configurations:
##    DRUG
## 0    D1
## 1    D2
## 2    D3
```

The earthquake network

> Mr. Holmes is working in his office when he receives a phone call (W) from his neighbor, who tells him that Holmes' burglar alarm (A) has gone off. Convinced that a burglar has broken into his house (B), Holmes rushes to his car and heads for home. On his way, he listens to t he radio, and in the news it is reported (N) that there has been a small earthquake (E) in the area. Knowing that earthquakes have a tendency to turn on burglar alarms, he returns to work.



Load some utility functions that we will use.

```r
source('functions.R')
```

Representation

Compact representation

- Which node has most free parameters associated? How many?
- How many parameters in total (do by hand)?
- How many parameters in a full joint?
- Where does the reduction come from?

Independence

What are the v-structures in the network?

```r
vstructs(bl.alarm)
```

```
##      X         Z       Y
## [1,] "Burglar" "Alarm" "Earthquake"
```

- State two independencies that hold in absence of evidence.
- Which probability queries would let us verify them?

Conditional independence (d-separation)

- Looking at the network, do you think these hold:
  - $I(B; W \mid A)$?
  - $I(B; N \mid A)$?

- Why/why not?

Check the above by querying d-separation. For this, we will use the `bnlearn` package.

Check the above using the `dsep()` function:

- $I(B; W \mid A)$?
- $I(B; N \mid A)$?

```r
dsep(bn=bl.alarm, x='Burglar', y=, z=)
```

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

- Is Watson independent of News given Earthquake $I(N; W \mid E)$?
- And the other way around $I(W; N \mid E)$?
- Why/why not?

Markov blanket

- Can you identify the Markov blanket of Earthquake? Try verifying it by checking d-separation.

```r
mb(x=bl.alarm, node='Earthquake')
mbe <- subgraph(bl.alarm, nodes = mb(x=bl.alarm, node='Earthquake'))
plot(mbe)
```

All the Markov blankets:

```r
par(mfrow=c(2, 3))
print_mblankets(bl.alarm)
```

```
## [[1]]
## A graphNEL graph with directed edges
## Number of Nodes = 3
## Number of Edges = 0
##
## [[2]]
## A graphNEL graph with directed edges
## Number of Nodes = 2
## Number of Edges = 1
##
## [[3]]
## A graphNEL graph with directed edges
## Number of Nodes = 3
## Number of Edges = 1
##
## [[4]]
## A graphNEL graph with directed edges
## Number of Nodes = 1
## Number of Edges = 0
##
## [[5]]
## A graphNEL graph with directed edges
## Number of Nodes = 1
## Number of Edges = 0
```

```r
par(mfrow=c(1, 1))
```

- So, given Burglar, News, and Alarm, is Earthquake independent of Watson?

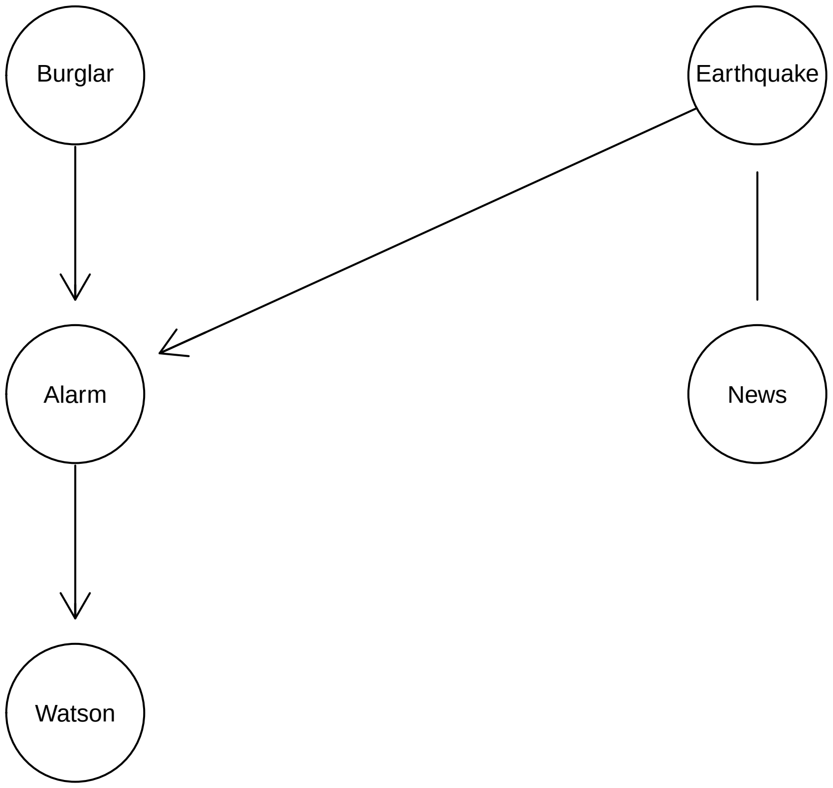
Equivalence classes

Different networks can encode same conditional independencies.

- How could we obtain an equivalent DAG?
- Obtain and plot the equivalent DAG.
- Is News independent of Watson given Earthquake ($I(N; W \mid E)$) in this equivalent DAG?
- How many DAGs in this equivalence class?

We can obtain a representative model of the equivalence class, a complete partially directed acyclic graph, with `cpdag()`

```r
# Equivalence class.
cpdag.alarm <- cpdag(bl.alarm)
plot(cpdag.alarm)
```



Reasoning

We will use the `gRain` package for exact inference. We can convert the `bnlearn` objects to `gRain` ones. Do we need `bl.alarm` or `bl.alarm.fit` to do inference?

```
library(gRain)
```

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

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

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

```
?gRain
```

```
## No documentation for 'gRain' in specified packages and libraries:
## you could try '??gRain'
```

```
gr.alarm <- as.grain(bl.alarm.fit)
```

Similarly to `bnlearn`, we can construct the Earthquake network (specify its DAG and conditional probability tables (CPTs)) with the `grain` package.

```
yn <- c("yes","no")
B  <- cptable(~Burglar, values=c(30,70),levels=yn)
E  <- cptable(~Earthquake, values=c(35,65),levels=yn)
A  <- cptable(~Alarm+Earthquake+Burglar, values=c(95,5,90,10,60,40,1,99),levels=yn)
```

```r
W  <- cptable(~Watson+Alarm, values=c(80,20,40,60),levels=yn)
N  <- cptable(~News+Earthquake, values=c(60,40,1,99), levels=yn)
```

```r
cptlist <- compileCPT(list(B, E, A, W, N))
gr.alarm.orig <- grain(cptlist)
```

Plot the network.

```r
plot(gr.alarm.orig)
```

Probability queries with no evidence

Let us see the marginal distribution over Earthquake ($P(E)$)?

```r
gr.alarm.orig$cptlist$Earthquake
```

```
## Earthquake
##  yes   no
## 0.35 0.65
## attr(,"class")
## [1] "parray" "array"
```

- How does this relate to the CPT specification above?
- What is $P(B)$?
- What is the probability that nothing occurs $P(B = b^0, E = e^0, A = a^0, W = w^0, N = n^0)$? Compute it using the CPTs.

```r
bn <- gr.alarm.orig$cptlist$Burglar['no']
en <- gr.alarm.orig$cptlist$Earthquake['no']
an <- gr.alarm.orig$cptlist$Alarm['no', 'no', 'no']
nn <- gr.alarm.orig$cptlist$News['no', 'no']
wn <- gr.alarm.orig$cptlist$Watson['no', 'no']
bn * en * an * nn * wn
```

Instead of summing and multiplying the probabilities in the CPTs, the `grain` object can do inference for us.

First, we need to *compile* the `grain` network.

```r
gr.alarm <- compile(gr.alarm.orig) # Compile first
gr.alarm
```

```
## Independence network: Compiled: TRUE Propagated: FALSE
##   Nodes: chr [1:5] "Burglar" "Earthquake" "Alarm" "Watson" "News"
```

Now we can query the network:

```r
querygrain(object=gr.alarm, nodes="Earthquake")
```

```
## $Earthquake
## Earthquake
##  yes   no
## 0.35 0.65
```

```r
querygrain(object=gr.alarm, nodes="Burglar")
```

```
## $Burglar
## Burglar
## yes  no
## 0.3 0.7
```

- Use `setEvidence()` and `pEvidence()` to get the probability that nothing occurs. Complete the missing arguments. What is the probability?

```
no <- rep("no", 5)
nodes <- c('Burglar', 'Earthquake', 'Alarm', 'Watson', 'News')
gr.alarm <- setEvidence(object=gr.alarm, nodes=, states=)
pEvidence(gr.alarm)
```

Finally, we will retract the evidence to return the network to initial state:

```
gr.alarm
```

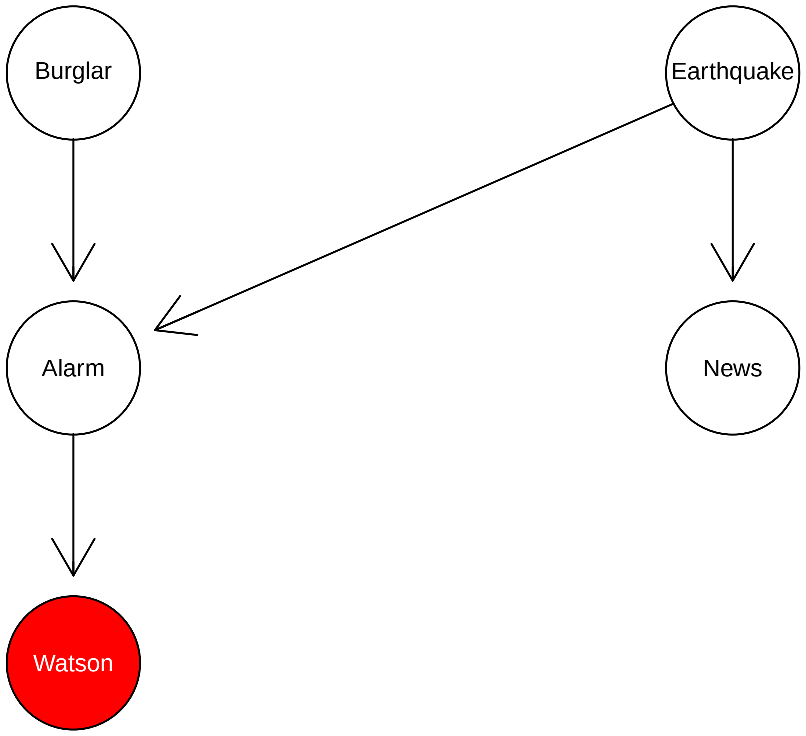
```
## Independence network: Compiled: TRUE Propagated: TRUE
##   Nodes: chr [1:5] "Burglar" "Earthquake" "Alarm" "Watson" "News"
##   Evidence:
##        nodes is.hard.evidence hard.state
## 1    Burglar             TRUE         no
## 2 Earthquake             TRUE         no
## 3      Alarm             TRUE         no
## 4     Watson             TRUE         no
## 5       News             TRUE         no
##   pEvidence: 0.267567
```

```
gr.alarm <- retractEvidence(gr.alarm, nodes)
gr.alarm
```

```
## Independence network: Compiled: TRUE Propagated: TRUE
##   Nodes: chr [1:5] "Burglar" "Earthquake" "Alarm" "Watson" "News"
```

Reasoning patterns: Causal, evidential and intercausal

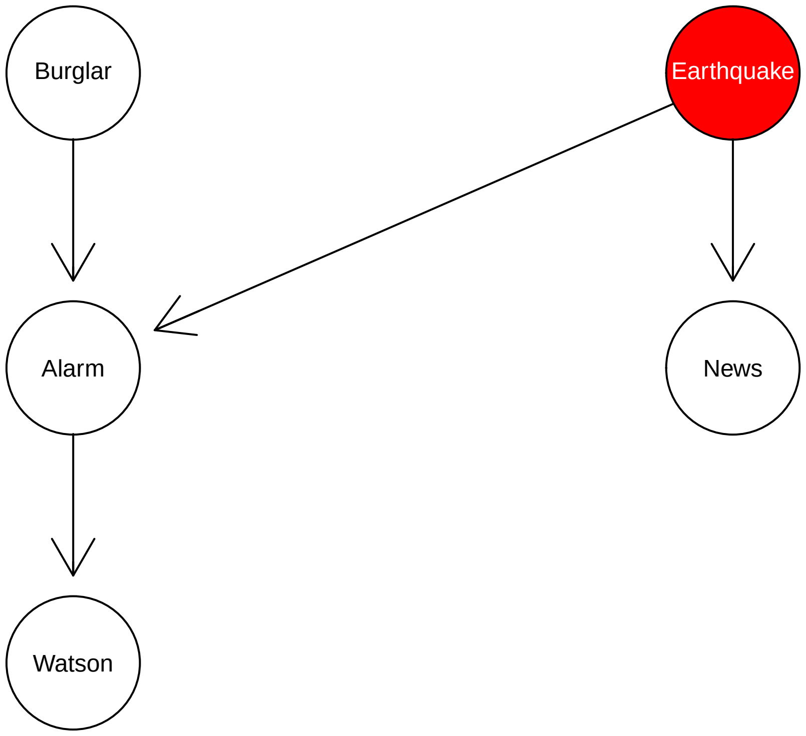
- Watson called. Set that as evidence.



- Is earthquake more likely now $(P(e^1 \mid w^1) > P(e^1))$?
- Is burglary more likely $(P(b^1 \mid w^1) > P(b^1))$?
- Which reasoning pattern was this?

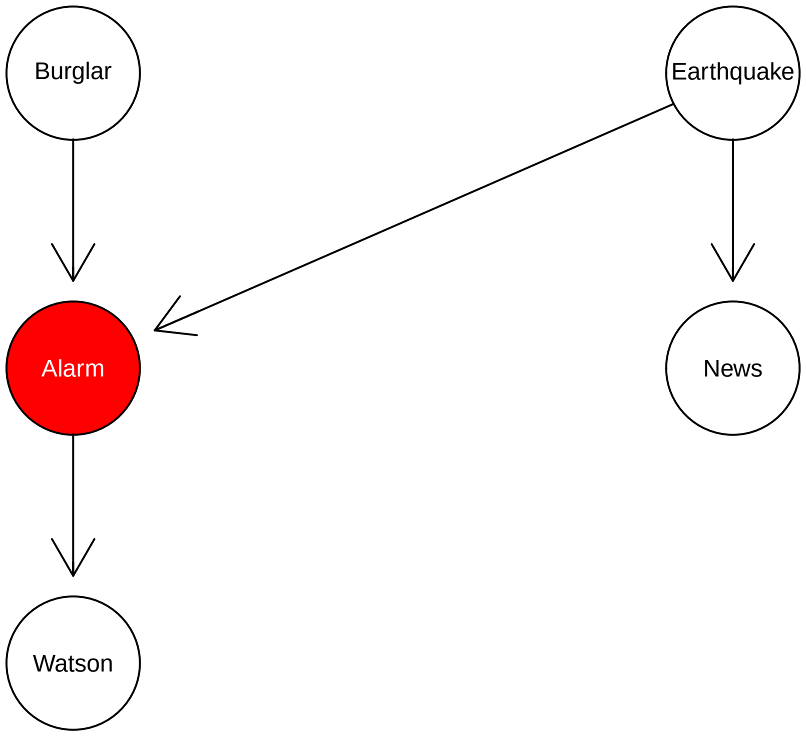
Before continuing, retract the evidence on Watson.

- Now, let us consider that an earthquake has occurred. Set it as evidence.

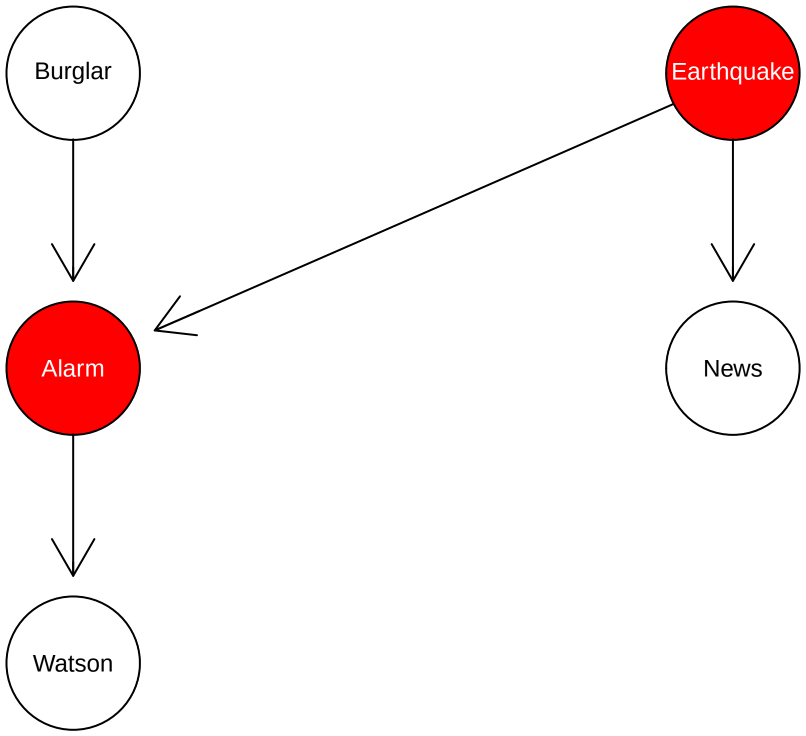


```r
gr.alarm <- setEvidence(object=gr.alarm, nodes="Earthquake", states="yes")
```

- Is Watson more likely to call ($P(w^1 \mid e^1) > P(w^1)$)?
- Which reasoning pattern did we apply?
- Is burglary more likely now ($P(b^1 \mid e^1) > P(b^1)$)? Why/why not?
- Now consider only the alarm went off.



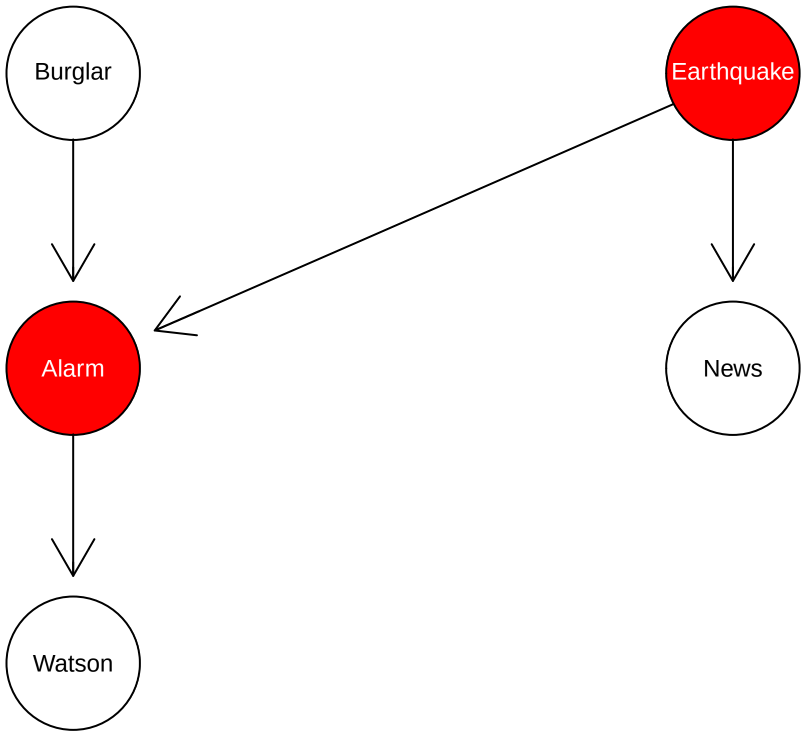
- Is burglary more likely now ($P(b^1 \mid a^1) > P(b^1)$)?
- Now, also an earthquake occurs.



- What about burglary now ($P(b^1 \mid a^1) < P(b^1 \mid a^1, e^1)$)?
- Which reasoning pattern was this?

Maximum a posteriori (MAP) queries

- An earthquake has occurred and alarm went off.



- What is the most likely event in the full space? In other words, what is the most likely state for `W`, `B` and `N` ($\arg\max_{w,b,n} P(W, B, N|e^1, a^1)$)?

Get the joint posterior over the corresponding nodes:

```
q.wnb <- querygrain(gr.alarm, type = "joint", nodes = )
q.wnb
```

```
## , , News = yes
##
##         Watson
## Burglar       yes          no
##     yes 0.1940426 0.04851064
##     no  0.2859574 0.07148936
##
## , , News = no
##
##         Watson
## Burglar       yes          no
##     yes 0.1293617 0.03234043
##     no  0.1906383 0.04765957
##
## attr(,"class")
## [1] "parray" "array"
```

- What is the most likely state? What is its probabillity?

```
map(q.wnb)
```

```
## $state
## Burglar  Watson    News
##    "no"   "yes"   "yes"
##
## $prob
## [1] 0.2859574
```

```
q.wnb["yes", "yes", "no"]
```

```
## [1] 0.1293617
```

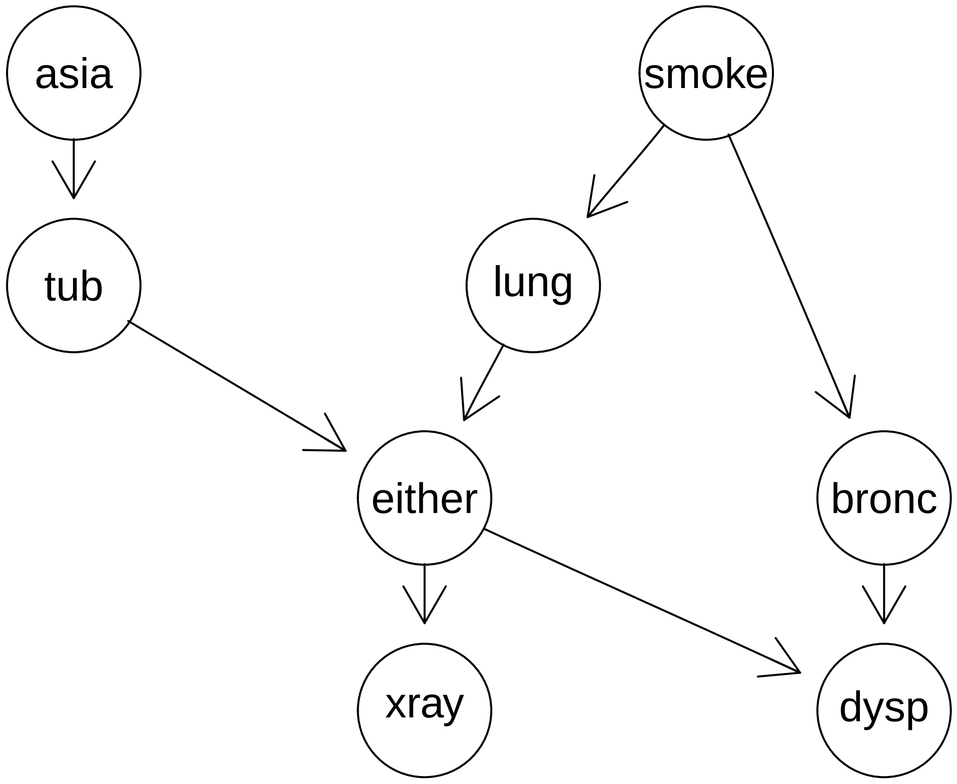
No specialized algorithm in `gRain` to get the MAP assignment; we need to build a joint over the nodes of interest.

- What was the most likely joint state before observing any evidence ($\arg max_{w,b,n,e,a} P(W, B, N, E, A)$)? Call the `map` utility function on the result of `querygrain()`.

Inference

The `gRain` package allows conditional probability queries for discrete Bayesian networks. We can also ask for the probability of the evidence. `bnlearn` provides approximate inference based on sampling for discrete, Gaussian and conditional linear Gaussian networks.

The Asia network



All variables are Boolean (yes/no).

Load the network.

```
asia.fit <- load_asia_fit()
```

Let us have a quick look at the network. Look at, for example, the CPT of either:

```
asia.fit$either
```

```
##
##   Parameters of node either (multinomial distribution)
##
## Conditional probability table:
##
## , , tub = yes
##
##       lung
## either yes no
##    yes   1  1
##    no    0  0
##
## , , tub = no
##
##       lung
## either yes no
##    yes   1  0
##    no    0  1
```

Conditional probability queries

We will look at queries of the type $P(\mathbf{Q}|\mathbf{e})$, where $\mathbf{W} = \mathbf{X} \setminus \mathbf{E} \setminus \mathbf{Q}$ may be non-empty.

What if all variables are observed, i.e., $\mathbf{E} = \mathbf{X}$; how complex is it to compute the probability of evidence?

Consider that have observed all variables except for B and D, and we want the marginal for D. If we were to do inference from the full joint, how many summations do we need to do for each value of D? And what if we observed no evidence?

In the full joint, the number of summations is exponential in number of unobserved variables. However, the summations have many common factors and we can reduce complexity by caching them.

We multiply variables to take them into account. Sum in order to marginalize them out. Computational complexity depends on the largest factor created. The algorithm works on undirected graphs so that is what we will do.

Junction tree message passing

`gRain` steps when performing inference:

- compile: moralize, triangulate, find RIP ordering, form initial clique potentials

- optional: absorb evidence
- propagate: transform clique potentials to clique marginals

`querygrain` performs those steps internally, no need to do them oneself.

Consider the moralized, triangulated graph of asia.

```
asia.gr <- as.grain(asia.fit)
m <- moralize(asia.gr$dag)
t <- triangulate(m)
```

- Which edges were added by moralization?
- And by triangulation?
- What cliques is A part of?
- What about B?
- What is the largest clique?

```
library(gRbase)
getCliques()
```

- Does the $\psi(A, T)$ clique potential correspond to $P(T \mid A)$?

```
asia.gr <- compile(asia.gr)
# origpot contains the clique potentials, i.e., initial cliques
asia.gr$origpot[[1]]
```

```
##      asia
## tub      yes     no
##   yes 0.0005 0.0099
##   no  0.0095 0.9801
```

```
asia.fit$tub$prob
```

```
##      asia
## tub    yes   no
##   yes 0.05 0.01
##   no  0.95 0.99
```

```
asia.fit$asia$prob
```

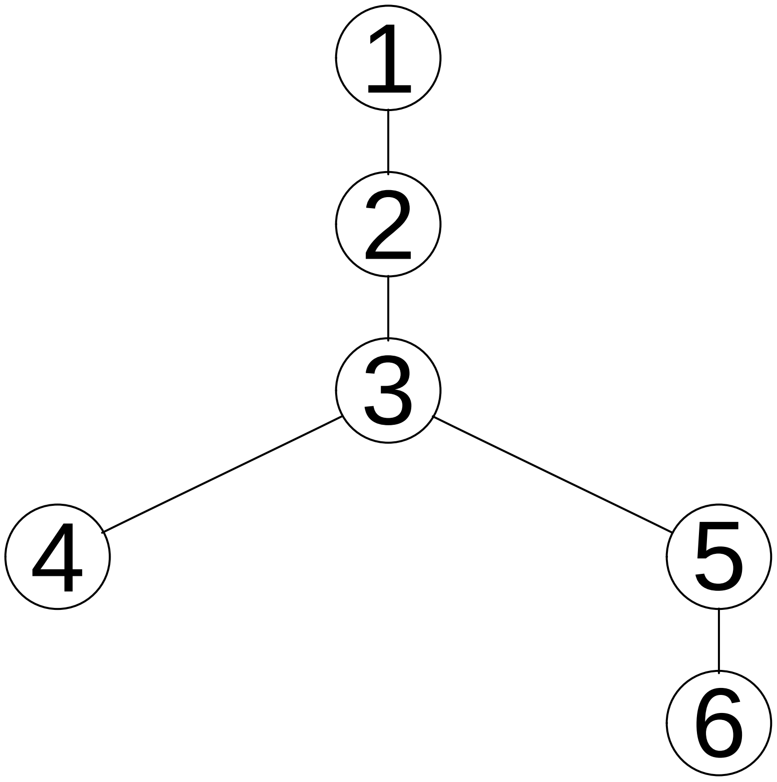
```
##  yes   no
## 0.01 0.99
```

- What do the entries of clique $\psi(B, D, E)$ correspond to?
- Can we get $P(B)$ from clique $\psi(B, D, E)$ alone? Why / why not?

We need a RIP ordering in order to obtain a junction tree.

- What is the 3rd clique?
- Is E on every path between $\psi(E, X)$ and $\psi(E, L, T)$?
- If we query for $P(S)$ will there be a factor with a domain including both A and S? What is the largest clique?

```
asia.gr$rip
```



Now, propagate the tree to calibrate the clique potentials.

```
asia.gr <- propagate(asia.gr)
```

- Did $\psi(A, T)$ change? Why / why not?

```
asia.gr$equipot[[1]] == asia.gr$origpot[[1]]
```

- Can we get $P(B)$ from clique $\psi(B, D, E)$ alone? Why / why not? - Can we get $P(B)$ from clique $\psi(B, S, L)$ alone? Why / why not? - How costly could have the above computation of $P(B)$ been? - What if we query for $P(A, S)$. Will the largest clique have to grow?

```r
# equipot contains the clique marginals
asia.gr$equipot[[5]]
apply(asia.gr$equipot[[5]], 2, sum)
apply(asia.gr$equipot[[4]], , )
```

```r
querygrain(asia.gr, nodes = 'bronc')
```

- Let us introduce evidence. How have the clique marginals changed? Which have changed? (hint: look at RIP)
- Does $\psi(X, E)$ (the 6th clique) correspond to $P(X \mid E)$?

```r
summary(asia.gr)
```

```
## Independence network: Compiled: TRUE Propagated: TRUE
##  Nodes : chr [1:8] "asia" "tub" "smoke" "lung" "bronc" "either" "xray" "dysp"
##  Number of cliques:                 6
##  Maximal clique size:               3
##  Maximal state space in cliques:    8
```

```r
asia.gr <- setEvidence(asia.gr, nodes = 'tub', states = 'yes', propagate = FALSE)
asia.gr <- propagate(asia.gr)
summary(asia.gr)
```

```
## Independence network: Compiled: TRUE Propagated: TRUE
##  Nodes : chr [1:8] "asia" "tub" "smoke" "lung" "bronc" "either" "xray" "dysp"
##  Number of cliques:                 6
##  Maximal clique size:               3
##  Maximal state space in cliques:    8
##   nodes is.hard.evidence hard.state
## 1   tub             TRUE        yes
```

```
##      either
## lung    yes no
##   yes 0.055  0
##   no  0.945  0
```

```
##       lung
## either   yes    no
##    yes 0.055 0.945
##    no  0.000 0.000
```

```r
querygrain(asia.gr, nodes = c('lung', 'either'), type = 'joint')
apply(asia.gr$equipot[[]], c(2, 3), )
```

Logic sampling

`bnlearn` provides two functions for sampling-based inference:

- `cpdist` : create a sample $\mathcal{D}$ of n [weighted] instances
- `cpquery`: compute the probability of an event in $\mathcal{D}$

- Sample smoke, lung, and bronc with the base function `sample()`. Begin with smoke

```r
yn <- c("yes", "no")
set.seed(0)
s <- sample(yn, 1, prob = asia.fit$smoke$prob)
```

- How do we sample lung?
- What values do you get for lung and bronc? Repeat.

```r
a <- sample(yn, 1, prob = asia.fit$asia$prob)
t <- sample(yn, 1, prob = asia.fit$tub$prob[, a])
e <- sample(yn, 1, prob = asia.fit$either$prob[, l, t])
e <- "no"
x <- sample(yn, 1, prob = asia.fit$xray$prob[, e])
d <- sample(yn, 1, prob = asia.fit$dysp$prob[, b, e])
instance <- c(a, s, t, l, e, x, b, d)
```

It is easier to sample with `cpdist()` from `bnlearn`. Let us draw 15 samples.

```r
set.seed(0)
# evidence = TRUE means that we are not conditioning on any evidence
samples.asia <- cpdist(asia.fit, nodes = nodes(asia.fit),
                       evidence = TRUE, n = 15)
summary(samples.asia)
```

```
##   asia     tub     smoke    lung    bronc    either    xray     dysp
##  yes: 1   yes: 0   yes:8   yes: 0   yes:10   yes: 0   yes: 1   yes:9
##  no :14   no :15   no :7   no :15   no : 5   no :15   no :14   no :6
```

- What is $P_{\mathcal{D}}(D)$?
- What is $P_{\mathcal{D}}(S = s^0, D = d^1)$?

```r
t <- table(samples.asia[, c('smoke', 'dysp')])
prop.table(t)
```

```
##      dysp
## smoke       yes        no
##   yes 0.3333333 0.2000000
##   no  0.2666667 0.2000000
```

- What is the most likely assignment to A and S?

We can also use `cpquery` to get the probability of an event directly (avoid the `prop.table(table())` step).

```r
set.seed(0)
ep <- cpquery(asia.fit,  event = (smoke == "no" & dysp == "yes"),
              evidence = TRUE, n = 15)
ep
```

```
## [1] 0.1333333
```

Since we can perform exact inference, we can measure the absolute error ($|P(S = s^0, D = d^1) - P_{\mathcal{D}}(S = s^0, D = d^1)|$). We need `grain` for exact inference.

```r
gr.asia <- as.grain(asia.fit)
q <- querygrain(gr.asia, nodes = c("smoke", "dysp"), type = "joint")
q
```

- What is the absolute error?
- Reduce absolute error below 0.001.

```r
set.seed(0)
ep <- cpquery(asia.fit,  )
```

- Consider we know that $A = a^1$. If we draw 5000 samples to estimate $P(S = s^0, D = d^1 | A = a^1)$, how many non-rejected samples do you expect? Why?

```r
set.seed(0)
samples.asia <- cpdist(asia.fit, nodes = c("smoke", "dysp"),
                       evidence=(asia == "yes"),
                       n=5000)
```

- What is the absolute error?

```r
ep <- prop.table(table(samples.asia))
ep <- ep['no', 'yes']
gray <- setEvidence(gr.asia, nodes = "asia", states = c("yes"))
q <- querygrain(gray, nodes = c("smoke", "dysp"), type = "joint")
tp <- q['no', 'yes']
abs(ep - tp)
```

- Reduce the absolute error below 0.001.
- How could we estimate the probability of the evidence?

- Why not just fix the value of the evidence in the network and then sample from other nodes?

Likelihood weighting

Say that we know $A = a^1$. With likelihood weighting we can fix $A = a^1$ and sample from the rest of the network.

- How many samples will be generated?

- If we had sampled from the prior distribution, without fixing A, how many samples could we have expected?

```r
set.seed(0)
samples.asia <- cpdist(asia.fit, nodes = nodes(asia.fit),
                       evidence=list(asia = "yes"),
                       n=5000, method = "lw")
```

- What are the weights of the samples?

```r
w <- attr(samples.asia, 'weights')
summary(w)
```

Consider instead that the sole evidence is $L = l^1$. Draw 5000 samples in S and D.

```r
set.seed(0)
samples.asia <- cpdist(, nodes = ,
                       evidence=list(lung = 'yes'),
                       n=5000, method = "lw")
w <- attr(samples.asia, 'weights')
```

- What are weights of the first five samples? Do they depend on D?
- Does the distribution of S correspond to that in the prior distribution? Yet, does $P(A)$ change if we know $L = l^1$?

```r
prop.table(table(samples.asia$smoke))
asia.fit$
```

- Why are there weights below 1? Hint: What is $\frac{P(L=l^1|S=s^1)}{P(L=l^1|S=s^0)}$?

So, what is $P_D(S = s^0, D = d^1 \mid L = l^1)$?

```
prop.table(xtabs(w ~ smoke + dysp, data = cbind(samples.asia, w = w)))
```

We can compute it with `cpquery`. What if we removed `set.seed()`?

```
set.seed(0)
epl <- cpquery(,  event = (smoke == "no" & dysp == ),
              evidence = list(lung = 'yes'), n = 5000, method = 'lw')
epl
```

- What is the absolute error?

```
## [1] 0.003619475
```

```
## [1] 0.9502207
```

```
gr.asia <- setEvidence(gr.asia, nodes = 'lung', states = 'yes')
q <- querygrain(gr.asia, nodes = c("smoke", "dysp"), type = "joint")
tpl <- q['no', 'yes']
abs(tpl - )
```

Summing up the weights, an effective sample size would be:

```
sum(w)
```

```
## [1] 2691.5
```

The marks data set

Consider the marks dataset (students' scores in mechanics, vectors, algebra, analysis and statistics):

```
summary(marks)
```

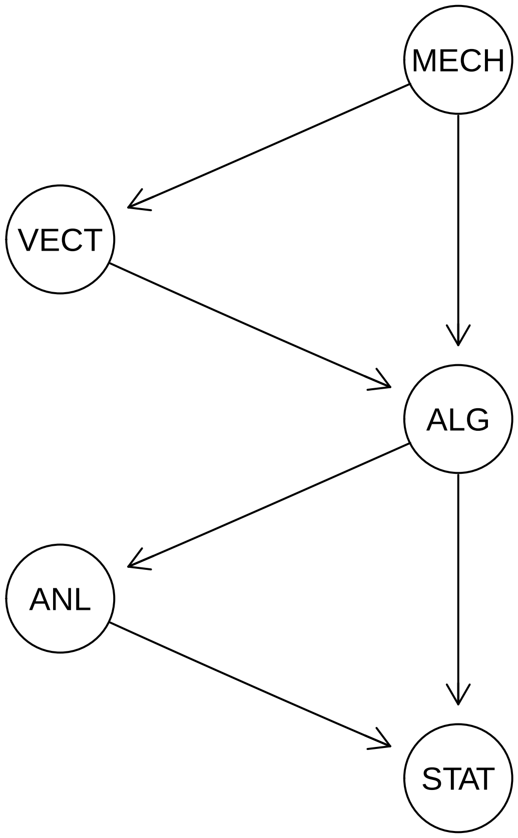
```
##       MECH            VECT            ALG             ANL       
##  Min.   : 0.00   Min.   : 9.00   Min.   :15.00   Min.   : 9.00  
##  1st Qu.:30.00   1st Qu.:42.00   1st Qu.:45.00   1st Qu.:35.75  
##  Median :41.50   Median :51.00   Median :50.00   Median :49.00  
##  Mean   :38.95   Mean   :50.59   Mean   :50.60   Mean   :46.68  
##  3rd Qu.:49.25   3rd Qu.:60.00   3rd Qu.:57.25   3rd Qu.:57.00  
##  Max.   :77.00   Max.   :82.00   Max.   :80.00   Max.   :70.00  
##       STAT      
##  Min.   : 9.00  
##  1st Qu.:31.00  
##  Median :40.00  
##  Mean   :42.31  
##  3rd Qu.:51.50  
##  Max.   :81.00  
```

Gaussian networks

Only particle-based (sampling) inference (`bnlearn`) is available for Gaussian networks.

Let us learn a network with `hc` and plot it.

```r
bn.marks <- hc(marks)
plot(bn.marks)
```



```r
bn.marks
```

```
##
##   Bayesian network learned via Score-based methods
##
##   model:
##    [MECH][VECT|MECH][ALG|MECH:VECT][ANL|ALG][STAT|ALG:ANL]
##   nodes:                                 5
##   arcs:                                  6
##     undirected arcs:                     0
##     directed arcs:                       6
##   average markov blanket size:           2.40
##   average neighbourhood size:            2.40
##   average branching factor:              1.20
##
##   learning algorithm:                    Hill-Climbing
##   score:                                 BIC (Gauss.)
##   penalization coefficient:              2.238668
##   tests used in the learning procedure:  34
##   optimized:                             TRUE
```

- What is the prob of ALG < 60?
- What is the probability of both STAT and MECH above 60?

```r
bn.marks.fit <- bn.fit(bn.marks, marks)
cpquery(bn.marks.fit, event = , evidence = )
```

```r
bn.marks.fit <- bn.fit(bn.marks, marks)
cpquery(bn.marks.fit, event = ((ALG < 60) & (MECH >= 60)), evidence = TRUE)
```

```
## [1] 0.05275
```

```r
cpquery(bn.marks.fit, event = ((STAT >= 60) & (MECH >= 60)), evidence = TRUE)
```

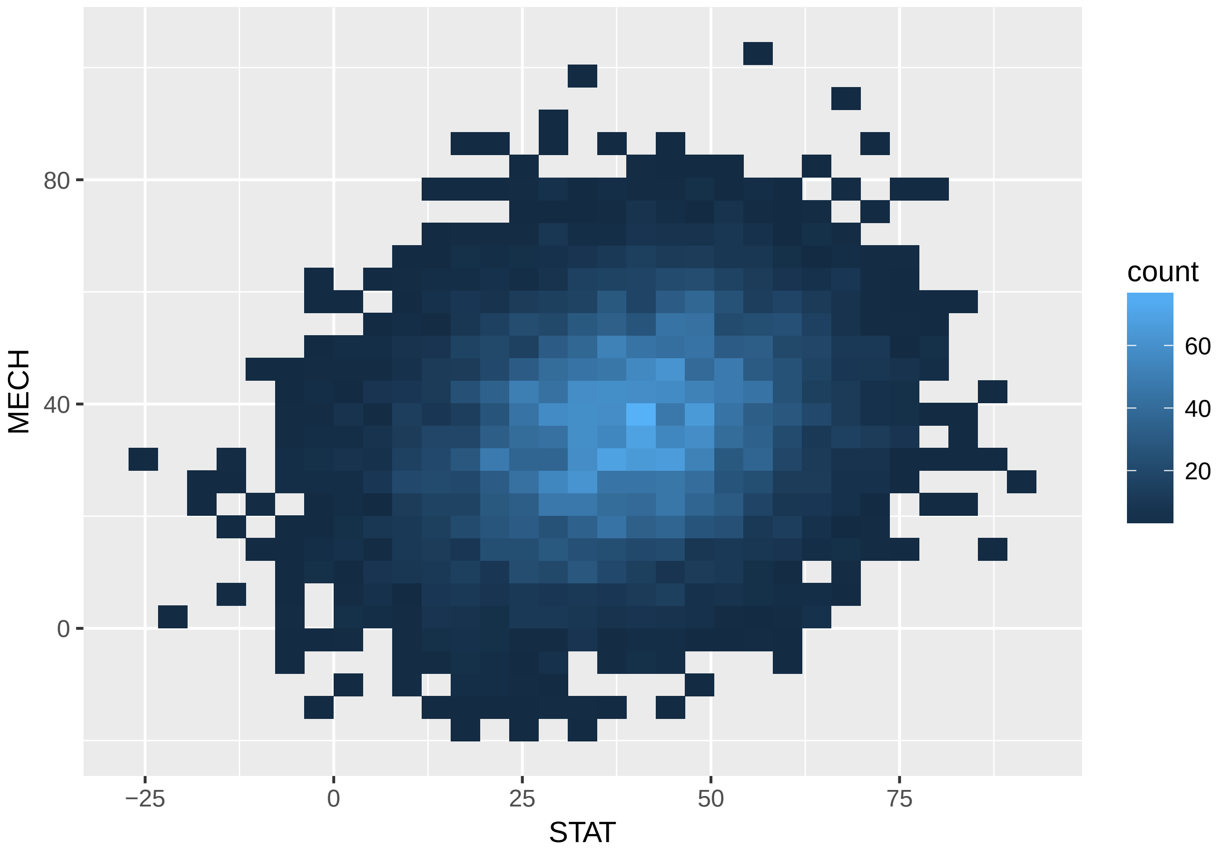
```
## [1] 0.035375
```

```r
cpquery(bn.marks.fit, event = ((STAT >= 60) & (MECH >= 60)), evidence = (ALG >= 60))
```

```
## [1] 0.1472149
```

- Is STAT independent of MECH? Let use `cpdist` and check this visually.

```r
library(ggplot2)
samples.marks <- cpdist(bn.marks.fit, nodes = c('STAT', 'MECH'), evidence = (ALG < 60))
ggplot(samples.marks, aes(x = STAT, y = MECH)) + geom_bin2d()
```



```r
cor(samples.marks$STAT, samples.marks$MECH)
```

```
## [1] 0.2364236
```

```r
cor(marks$STAT, marks$MECH)
```

```
## [1] 0.3890993
```

- What about conditional on ALG? How could we check it?

- What is the probability of STAT ≥ 60 conditional on MECH ≥ 60? And what about MECH ≥ 60 and ALG ≥ 60?. Use `cpquery`. Does this confirm they are independent?

```
cpquery(bn.marks.fit, event = ((STAT >= 60)), evidence = ((MECH >= 60) & ALG >= 60))
```

```
## [1] 0.490566
```

```
cpquery(bn.marks.fit, event = ((STAT >= 60)), evidence = (ALG >= 60))
```

```
## [1] 0.4413932
```

- What is the probability of the evidence? How many samples were kept with logic sampling?

```
samples.marks <- cpdist(bn.marks.fit, nodes = c('STAT', 'MECH'), evidence = (ALG < 60))
samples.marks.noevidence <- cpdist(bn.marks.fit, nodes = c('STAT', 'MECH'), evidence = TRUE)
nrow(samples.marks )
```

```
## [1] 6472
```

```
nrow(samples.marks.noevidence)
```

```
## [1] 8000
```

```
library(bnlearn)
```

Learning from data

Different objectives:

- Learn $P(\mathbf{Y} \mid \mathbf{X})$
- Discover associations
- Learn the joint

Breast cancer data

Data on recurrence of breast cancer within five years of surgery.

- 286 patients; 85 recurring whereas 201 not (we have removed 9 patients with incomplete information)
- 9 measured variables
  - Patient age (`age`)
  - Menopause (non, pre, post) (`menopause`)
  - Tumour malignancy degree (`deg_malig`)
  - Tumour location quadrant (`breast-quad`)
  - Perforated lymph node capsule (`node\_caps`)
  - Involved lymph nodes (`inv-nodes`)
  - Max. tumour diameter (`size`)
  - Patient irradiated (`irradiated`)
  - Left or right breast (`breast`)
  - Recurring or not (`Class`)

We have discretized `size} into groups 0--19, 20--39, and 40--54; andinv-nodes}` into groups 0–8, 9–17, and 8–26.

This is basically a classification problem where we are interested in recurrence. Yet, here we will consider learning the entire distribution.

Load the data and have a look at it:

```
breast <- foreign::read.arff('data/dbreast-cancer.arff')
summary(breast)
```

```
##     age        menopause    tumor_size   inv_nodes    node_caps deg_malig
##  20-29: 1   ge40   :123   0-19 : 69   0-8   :260   no :221   1: 66
##  30-39:36   lt40   :  5   20-39:175   18-26:  1   yes: 56   2:129
##  40-49:89   premeno:149   40-54: 33   9-17 : 16             3: 82
##  50-59:91
##  60-69:55
##  70-79: 5
##    breast       breast_quad  irradiat                   Class
##  left :145   central   : 21   no :215   no-recurrence-events:196
##  right:132   left_low  :106   yes: 62   recurrence-events   : 81
##              left_up   : 94
##              right_low: 23
##              right_up : 33
##
```

No missing values (i.e., we can apply `bnlearn`).

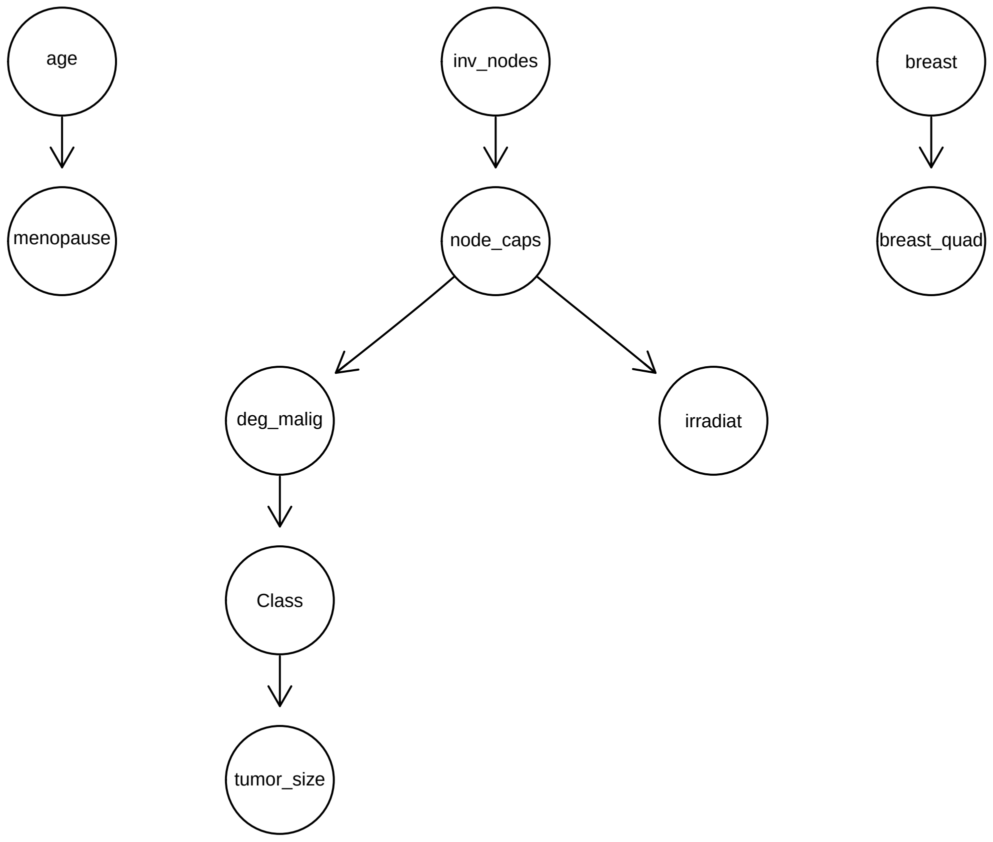
```
anyNA(breast)
```

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

Score + search algorithms

Let us use the hill-climbing (HC) algorithm with the BIC score to learn network structure

```
hc.breast <- hc(breast, score = "bic")
plot(hc.breast)
```



- Do the (in)dependencies make sense?
- Read three conditional independencies off the structure. If in doubt, verify with `dsep()`.
- Which nodes suffice to know $P(C)$? Which function can tell that?
- What are the parameters of the network? Could we compute the likelihood of the network?

Scores

Two types of scores:

- Information theoretic
- Bayesian

Compute the log-likelihood on the full data set. (Note that the parameters are learned from the second argument)

```r
score(hc.breast, breast, type="loglik")
```

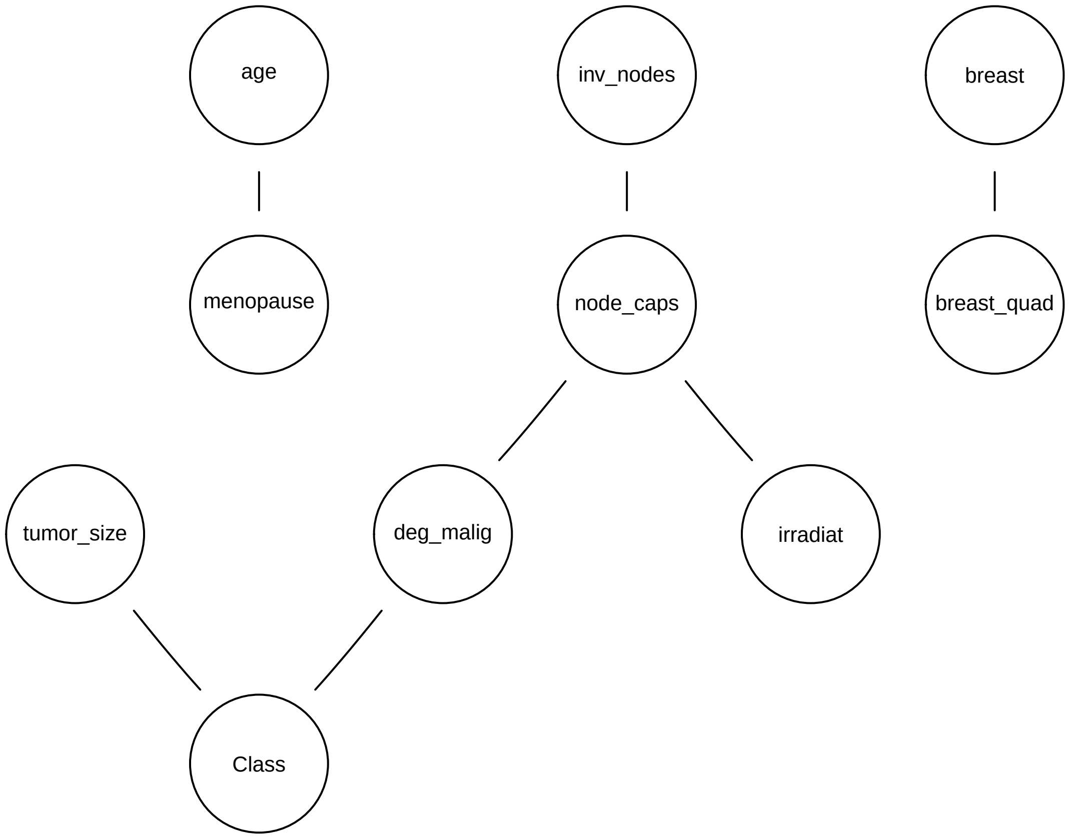
```
## [1] -2044.865
```

- Compute two other scores. Use `help("bnlearn-package")` to see which scores are available.
- Think how you could get an equivalent DAG to `hc.breast`. Once you obtain such a DAG, compute its BIC, BDe and K2. Compare to those of `hc.breast`.

```r
eq_dag <-
score(eq_dag, breast, type="bic")
score(eq_dag, breast, type="bde")
score(eq_dag, breast, type="k2")
```

- What is the complete partially directed acyclic graph? Are there any v-structures in this network? (see `vstruct`)

```r
pcdag <- skeleton()
plot(pcdag )
```



- Now use the log-likelihood score with hill-climbing. How many free parameters in the network?

```r
loglik.hc.breast <- hc(breast, score='loglik')
nparams(loglik.hc.breast, breast)
```

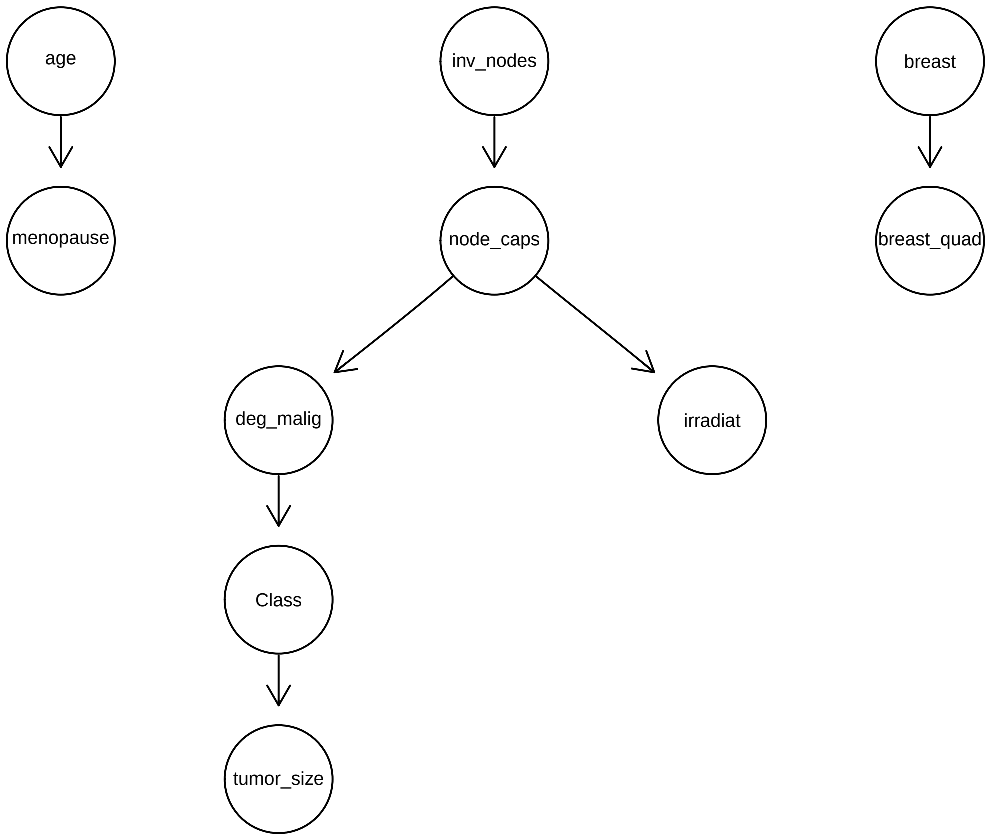
- What is the penalization with the BIC score? Print out the learned network.

- If AIC penalization is 1, which score yields sparser networks?

```r
hc.breast
```

```
##
##   Bayesian network learned via Score-based methods
##
##   model:
##    [age][inv_nodes][breast][menopause|age][node_caps|inv_nodes]
##    [breast_quad|breast][deg_malig|node_caps][irradiat|node_caps]
##    [Class|deg_malig][tumor_size|Class]
##   nodes:                                 10
##   arcs:                                  7
##     undirected arcs:                     0
##     directed arcs:                       7
##   average markov blanket size:           1.40
##   average neighbourhood size:            1.40
##   average branching factor:             0.70
##
##   learning algorithm:                   Hill-Climbing
##   score:                                BIC (disc.)
##   penalization coefficient:             2.812009
##   tests used in the learning procedure: 108
##   optimized:                            TRUE
```

- Use the BDe score and plot the network.



- Did BIC or BDe yield a more complex model?
- Use `compare` to find common arcs for the BDe and BIC structures.

```
compare(hc.breast, bde.hc.breast, arcs = TRUE)
```

Search

- Identify the first three steps of HC (hint: use `plot()` the `max.iter` argument to `hc()`).
- How did the BIC evolve?

```
## [1] -2226.295 -2211.158 -2199.784
```

- Which search operators does HC use? Why did it stop? (hint: run `hc()` with `debug = TRUE`)

We can also see that, in this case, random restarts do not seem to improve the BIC. Thus, gives us some confidence that we are not being trapped in a local optimum.

```
set.seed(100)
hc.breast.restart <- hc(breast, restart = 1000, perturb = 10)
BIC(hc.breast.restart, breast)
```

```
## [1] -2168.594
```

```
all.equal(cpdag(hc.breast.restart), cpdag(hc.breast))
```

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

Which other search + score algorithms are available in `bnlearn`?

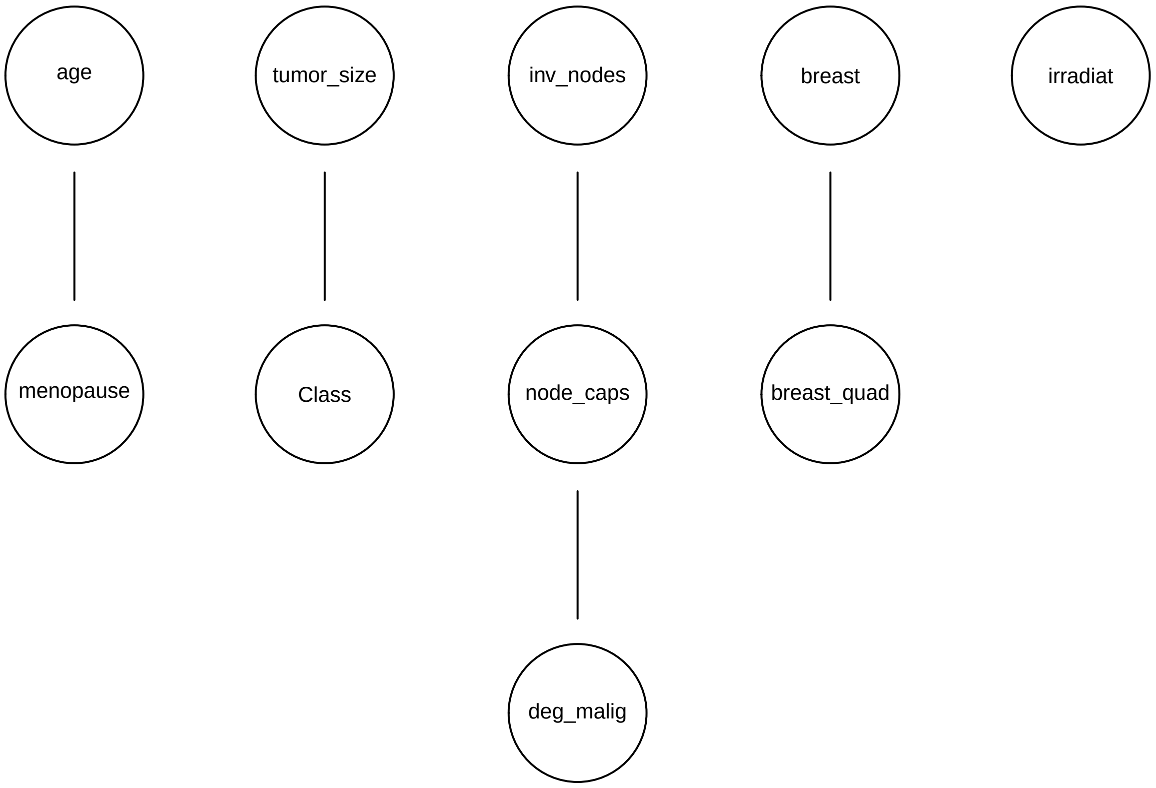
Constraint-based structure learning

- Learn a network with the Grow-shrink algorithm

```
gs.breast <- gs(x = breast)
```

Plot it. Is it a DAG?

```
plot(gs.breast)
```



- Try to compute the log-likelihood. What is the problem?

```
score(gs.breast, breast, type = "loglik")
```

Get a consistent extension first.

```
gs.breast <- cextend(gs.breast)
score(gs.breast, breast)
```

```
## [1] -2190.743
```

- Find how many arcs match those obtained with HC.

(Structural) Hamming distance indicates how similar the (DAGs) equivalence classes are:

```r
shd(gs.breast, hc.breast)
```

```
## [1] 2
```

```r
hamming(gs.breast, hc.breast)
```

```
## [1] 2
```

```r
compare(skeleton(hc.breast), skeleton(gs.breast), arcs = TRUE)
```

```
## $tp
##       from          to
##  [1,] "age"         "menopause"
##  [2,] "menopause"   "age"
##  [3,] "tumor_size"  "Class"
##  [4,] "inv_nodes"   "node_caps"
##  [5,] "node_caps"   "inv_nodes"
##  [6,] "node_caps"   "deg_malig"
##  [7,] "deg_malig"   "node_caps"
##  [8,] "breast"      "breast_quad"
##  [9,] "breast_quad" "breast"
## [10,] "Class"       "tumor_size"
##
## $fp
##      from to
##
## $fn
##      from        to
## [1,] "node_caps" "irradiat"
## [2,] "deg_malig" "Class"
## [3,] "irradiat"  "node_caps"
## [4,] "Class"     "deg_malig"
```

- Which one has higher BIC? Which one has more parameters? Which one would you prefer?

By printing the object we can find how many conditional independence (CI) tests were performed to obtain the structure.

```r
gs.breast
```

```
##
##   Bayesian network learned via Constraint-based methods
##
##   model:
##    [menopause][deg_malig][breast_quad][irradiat][Class][age|menopause]
##    [tumor_size|Class][node_caps|deg_malig][breast|breast_quad]
##    [inv_nodes|node_caps]
##   nodes:                                 10
##   arcs:                                  5
##     undirected arcs:                     0
##     directed arcs:                       5
##   average markov blanket size:           1.00
##   average neighbourhood size:            1.00
##   average branching factor:              0.50
```

```
##
##   learning algorithm:                    Grow-Shrink
##   conditional independence test:         Mutual Information (disc.)
##   alpha threshold:                       0.05
##   tests used in the learning procedure:  174
##   optimized:                             FALSE
```

- Change the alpha argument so that we only get 5 arcs

```
gs.breast.string <- gs(breast, alpha=)
narcs(gs.breast.string)
```

- Use a different test of conditional independence. Use `help("bnlearn-package")` to see the available tests. Plot the network.

```
gs.breast.2 <- gs(breast, test = )
gs.breast.2 <- cextend(gs.breast.2)
plot(gs.breast.2)
```

- What are the steps of GS? Run with the debug option. Unlike the PC algorithm, contraint-based algorithms in `bnlearn` learn local Markov blankets and then complete the full network.

```
gr.breast <- gs(breast, debug = TRUE)
```

- Consider another constraint-based algorithm from `bnlearn`. See `?gs` Does loglik improve with respect to GS?

Search + score is a more global approach to network learning, whereas constraint-based provides a more local approach.

Do hill-climbing or iamb generalize better with respect to log-likelihood?

```
bn.cv(breast, 'iamb', loss = "logl")
bn.cv(breast, 'hc', loss = "logl")
```

Mixed data

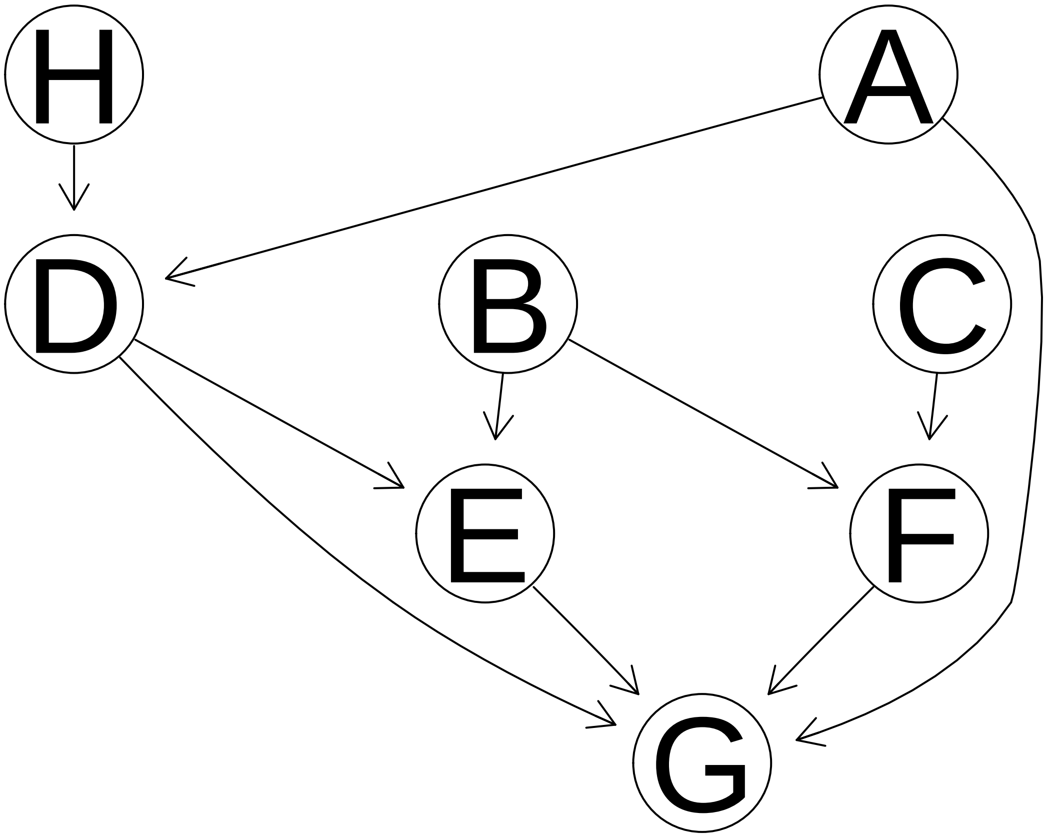
Consider a mixed discrete and real-valued data set:

```
data("clgaussian.test")
head(clgaussian.test)
```

```
##   A B C         D        E F         G        H
## 1 a b d  6.460721 11.98657 b  34.84246 2.334846
## 2 b a a 12.758389 30.43674 b 106.63596 2.359112
## 3 b c c 12.175140 17.21532 a  68.92951 2.319435
## 4 b c d 12.006609 14.41646 b  86.17521 2.417494
## 5 b a a 12.328071 30.39631 b 103.58541 2.268150
## 6 b c c 12.613419 15.19344 b  90.84664 2.308369
```

Let us learn a conditional linear Gaussian network. Real-valued variables cannot be parents of discrete ones.

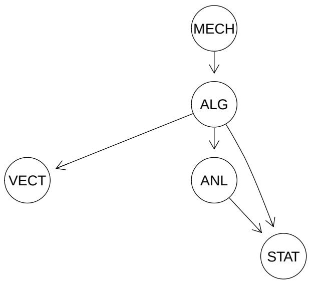
```
plot(hc(clgaussian.test))
```



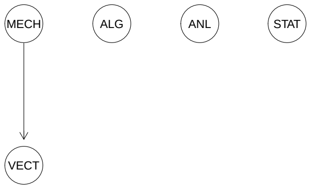
Latent variables

It may be useful to consider the presence of latent variables. Edwards ("Introduction to Graphical Modelling") assigned the students in the marks data set into two groups using an EM algortihm. The networks for the two groups are completely different, and both differ from the overall network.

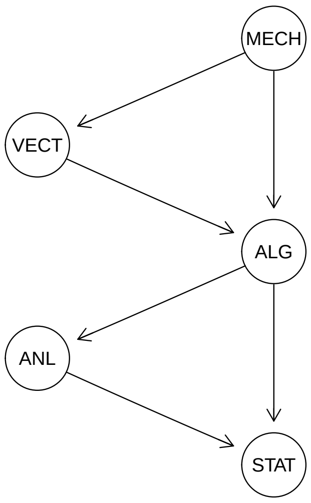
```r
data("marks")
latent <- factor(c(rep("A", 44), "B", rep("A", 7), rep("B", 36)))
plot(hc(marks[latent == "A", ]))
```



```r
plot(hc(marks[latent == "B", ]))
```

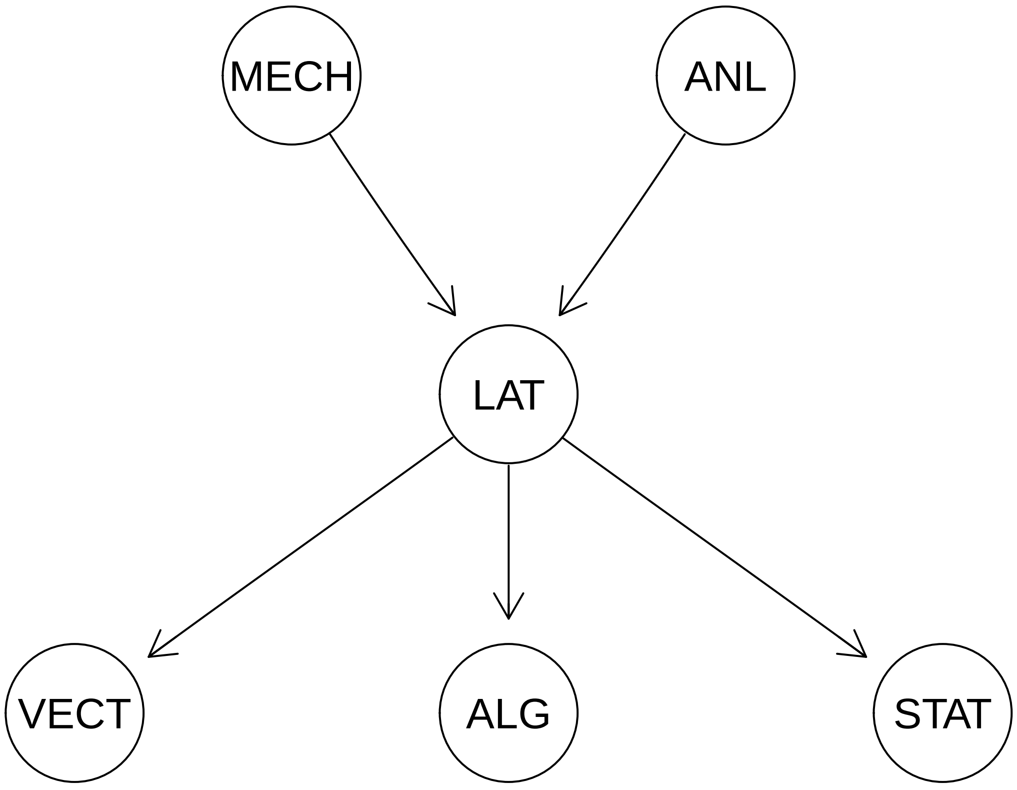


```r
plot(hc(marks))
```



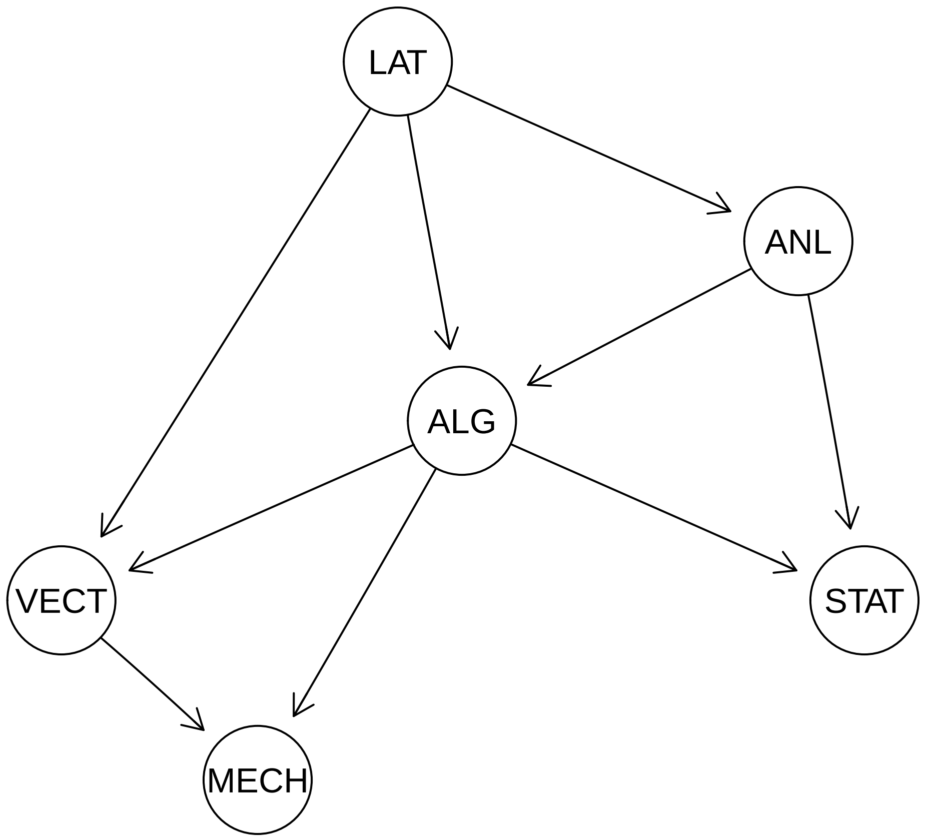
If we include the latent variable in will necessarily be a root of the network.

```r
dmarks = discretize(marks, breaks = 2, method = "interval")
plot(hc(data.frame(dmarks, LAT = latent)))
```



If we discretize the network we get yet a different model.

```r
lmarks <- data.frame(marks, LAT = latent)
plot(hc(lmarks))
```

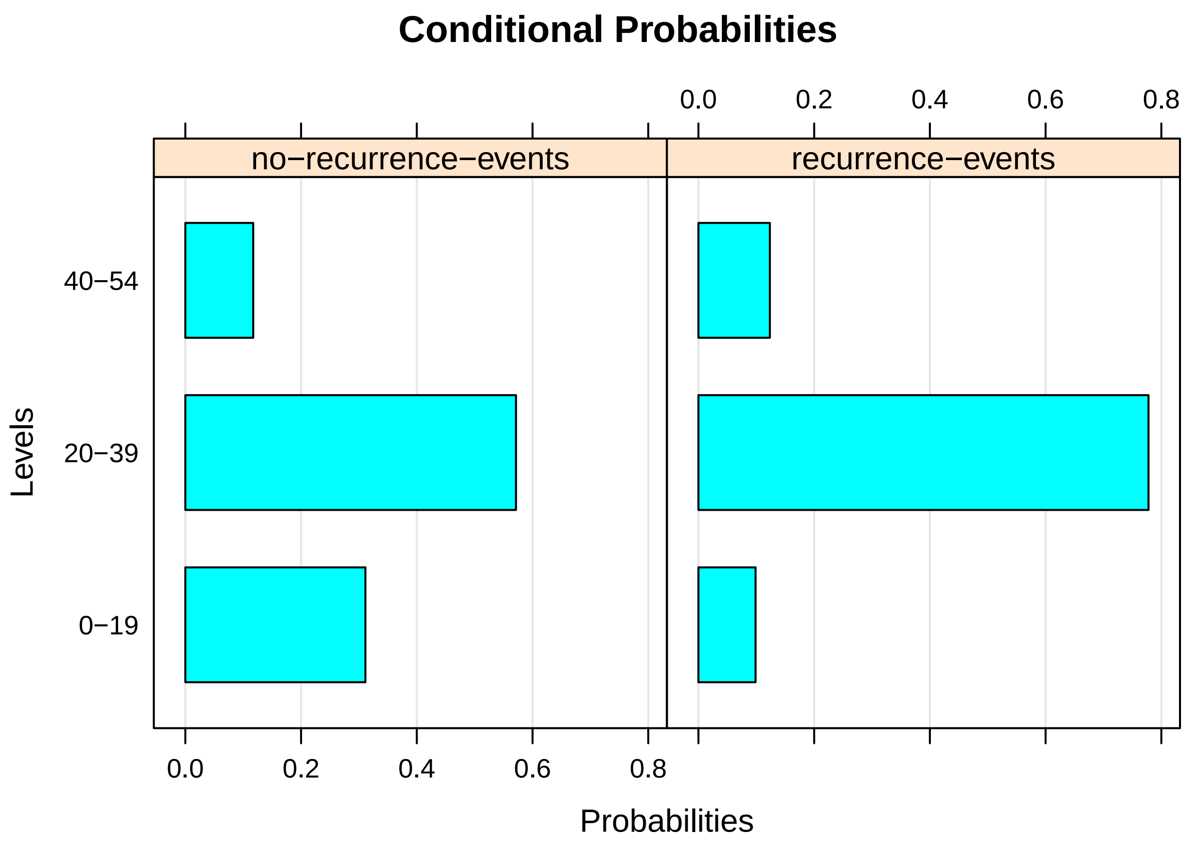


bnlearn uses interval, quantile and Hartemink's discretization. The latter tries to maximize mutual information between variables.

Parameter estimation

We use bn.fit to learn parameters with bnlearn

```
breast.bn.fit <- bn.fit(hc.breast, breast)
bn.fit.barchart(breast.bn.fit$tumor_size)
```



```
graph.breast <- as.graphNEL(hc.breast)
gr.breast <- grain(graph.breast, data = breast, smooth = 0)
gr.breast$cptlist$menopause
```

```
##          age
## menopause 20-29      30-39     40-49      50-59      60-69 70-79
##   ge40        0 0.00000000 0.1011236 0.61538462 0.96363636     1
##   lt40        0 0.02777778 0.0000000 0.02197802 0.03636364     0
##   premeno     1 0.97222222 0.8988764 0.36263736 0.00000000     0
## attr(,"class")
## [1] "parray" "array"
```

0 probabilities can produce undesirable results. In the Pathfinder study, 10% percent of cases were incorrectly diagnosed due to 0 probabilities —the correct disease was ruled out by a finding that had been given 0 probability (Koller and Friedman, 2006, pp. 67)

Avoid 0 probabilities by using Bayesian parameter estimation:

```
gr.breast.bpe <- grain(graph.breast, data = breast, smooth = 1)
gr.breast.bpe$cptlist$menopause
```

```
##          age
## menopause 20-29      30-39      40-49      50-59      60-69 70-79
##   ge40     0.25 0.02564103 0.10869565 0.60638298 0.93103448 0.750
##   lt40     0.25 0.05128205 0.01086957 0.03191489 0.05172414 0.125
##   premeno  0.50 0.92307692 0.88043478 0.36170213 0.01724138 0.125
## attr(,"class")
## [1] "parray" "array"
```

- Which of the two models has a higher likelihood score?

```r
logLik(as.bn.fit(gr.breast.bpe), breast)
logLik(as.bn.fit(gr.breast), breast)
```

- How could we make all local probability distributions uniform (just as an exercise; otherwise it does not make sense)?
- Does higher `deg\_malign` mean more likely recurrence?

Continuous variables

For continuous variables, `bnlearn` implements Gaussian networks. Each variable is normally distributed with its mean a linear function of its parents and a standard deviation σ.

Learn a structure with `hc` and then learn the parameters.

```r
data(marks)
bn.marks <- hc(marks)
bn.marks <- bn.fit(bn.marks, marks)
```

What are the coefficients?

```r
coefficients(bn.marks)
```

```
## $MECH
## (Intercept)
##    38.95455
##
## $VECT
## (Intercept)        MECH
##  34.3828788   0.4160755
##
## $ALG
## (Intercept)        MECH        VECT
##  25.3619809   0.1833755   0.3577122
##
## $ANL
## (Intercept)         ALG
##   -3.574130    0.993156
##
## $STAT
## (Intercept)         ALG         ANL
## -11.1920114   0.7653499   0.3164056
```

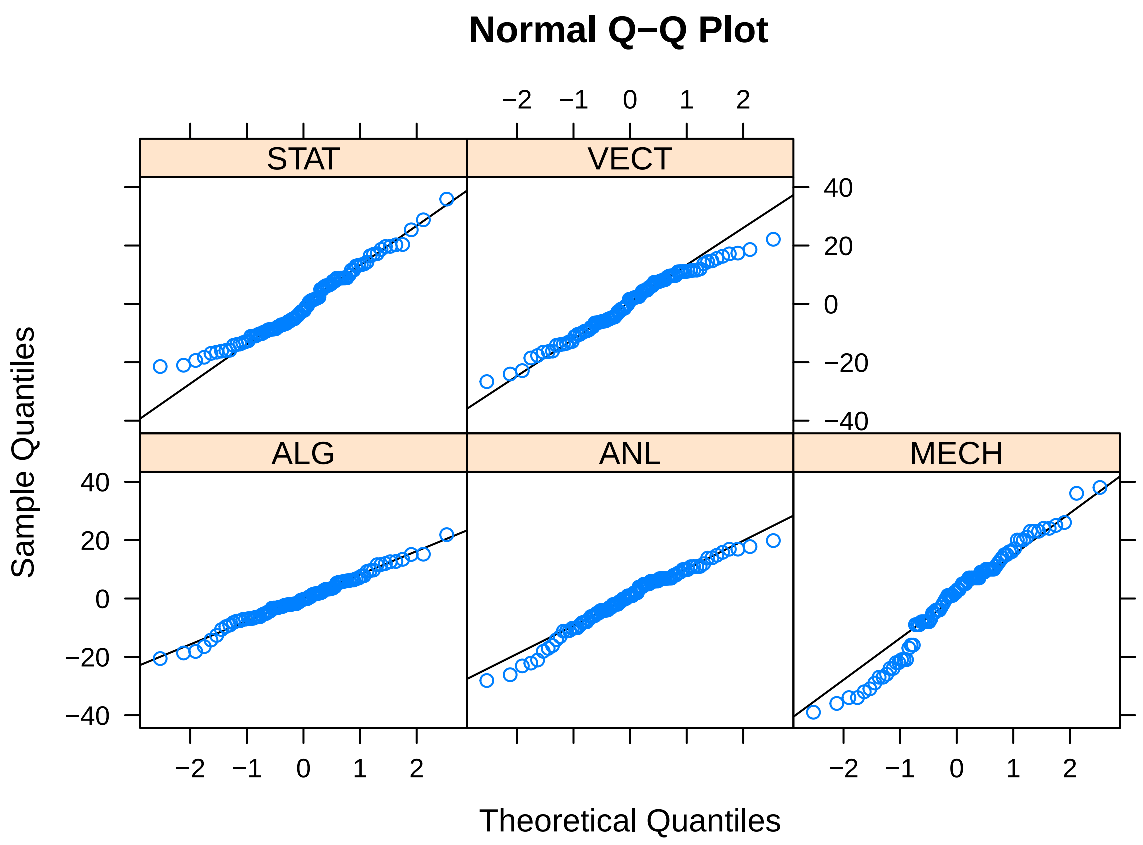
```r
sigma(bn.marks$MECH)
```

```
## [1] 17.48622
```

```r
sigma(bn.marks$VECT)
```

```
## [1] 11.01373
```

```r
bn.fit.qqplot(bn.marks)
```



There is no Bayesian parameter estimation for Gaussian networks. One can, however, fit the coefficients with ridge or elastic net regression instead of ordinary least squares, and then replace it in the network.

```
m <- as.matrix(marks)
# You will need the glmnet package for this
library(glmnet)
```

```
## Loading required package: Matrix

## Loading required package: foreach

## Loaded glmnet 2.0-16
```

```
gnet <- glmnet(m[ , c('VECT', 'MECH')], m[, 'ALG'], alpha =0, family = "gaussian")
coefs <- coef(gnet, s = 0.1)[, 1]
bn.marks$ALG = list(coef = coefs, sd = sigma(bn.marks$ALG))
```

Classification

While `bnlearn` implements discrete naive Bayes and TAN, we will use the `bnclassify` package for discrete classifiers.

The car data set

Car evaluation data set

- PRICE

    - buying: buying price
    - maint: price of the maintenance
- TECH
    - COMFORT
        * doors: number of doors
        * persons: capacity in terms of persons to carry
        * lug_boot: the size of luggage boot
    - safety: estimated safety of the car
- class: car acceptability

We are interested in car acceptability. It is a function of price and tech, which in turn are functions of the observed variables.

- Load the data. All variables discrete.

```
load('data/car.rda')
summary(car)
```

```
##    buying      maint       doors     persons     lug_boot    safety  
##  low  :432   high :432   2    :432   2   :576   big  :576   high:576  
##  med  :432   low  :432   3    :432   4   :576   med  :576   low :576  
##  high :432   med  :432   4    :432   more:576   small:576   med :576  
##  vhigh:432   vhigh:432   5more:432                                    
##    class     
##  unacc:1210  
##  acc  : 384  
##  good :  69  
##  vgood:  65
```

Naive Bayes

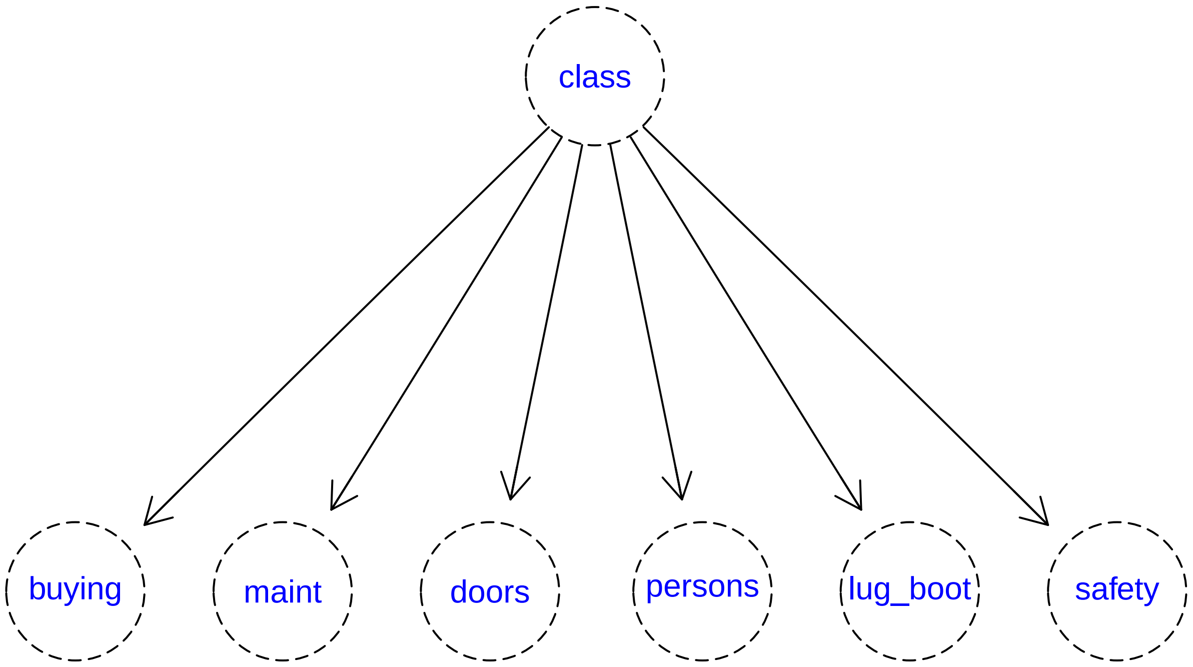
```
library(bnclassify)
```

```
## 
## Attaching package: 'bnclassify'
```

```
## The following objects are masked from 'package:bnlearn':
## 
##     modelstring, narcs, nparams
```

- Learn a naive Bayes structure from car data. The target variable is 'class'. Plot it.

```
nb.car <- nb(class = 'class', dataset =  )
```



- Learn network parameters. What is the class prior probability?

```
nb.car <- lp(nb.car, car, smooth = 0)
params(nb.car)[['class']]
```

```
## class
##      unacc        acc       good      vgood 
## 0.70023148 0.22222222 0.03993056 0.03761574
```

Predicting

When only one variable is unobserved, inference amounts to multiplying the entries for the observed ones.

- How could we compute class posterior for first instance using just the CPTs? The features values for the first instance are:

```
car[1, -7]
```

```
##   buying maint doors persons lug_boot safety
## 1  vhigh vhigh     2       2    small    low
```

We need to multiply CPT entries. Complete the code below to get the class-conditional probabilities for `lug_boot` and `safety`.

```
b <- params(nb.car)[['buying']]['vhigh', ]
m <- params(nb.car)[['maint']]['vhigh', ]
d <- params(nb.car)[['doors']]['2', ]
p <- params(nb.car)[['persons']]['2', ]
l <- params(nb.car)[['lug_boot']]['', ]
s <- params(nb.car)[['safety']]['', ]
cp <- params(nb.car)[['class']]
cpost <- cp * b * m *  * p *  *
```

- Now, multiply the CPT entries accordingly. You should get the following:

```
## class
##       unacc         acc        good       vgood
## 0.001407374 0.000000000 0.000000000 0.000000000
```

- Is this the class posterior distribution? How do we get the class posterior.
- Why are there 0's?
- Avoid zero's by using Bayesian parameter estimation (use a Dirichlet prior with hyperparameter `alpha != 0`)

```
nb.car <-
```

Get the class posterior for the last five instances:

- What would the predicted class labels be?

```
p <- predict(nb.car, car, prob = TRUE)
tail(p)
```

```
##              unacc         acc       good       vgood
## [1723,] 0.94459821 0.004153620 0.02865175 0.022596424
## [1724,] 0.23381820 0.299936481 0.45723037 0.009014958
## [1725,] 0.12346412 0.230996424 0.24095583 0.404583624
## [1726,] 0.92809628 0.004634127 0.02998218 0.037287418
## [1727,] 0.21719983 0.316377610 0.45235815 0.014064411
## [1728,] 0.09339996 0.198429598 0.19413755 0.514032894
```

Get them by dropping the `prob` the argument (or setting it to `FALSE`).

```
p <- predict(nb.car, car)
tail(p)
```

How well does naive Bayes learn the training data? Compute accuracy from a confusion matrix.

```
cm <- table(predicted=p, true=car$class)
cm
```

```
##          true
## predicted unacc  acc good vgood
##     unacc  1162   87    0     0
##     acc      46  287   46    30
##     good      2   10   21     0
##     vgood     0    0    2    35
```

```
sum(cm * diag(1, nrow(cm), ncol(cm))) / sum(cm)
```

```
## [1] 0.8709491
```

or use `accuracy` from `bnclassify`.

```
bnclassify:::accuracy(p, car$class)
```

```
## [1] 0.8709491
```

Irrelevant features

Is the `doors` feature independent of the class? Use `bnlearn::ci.test` to test.

```
ci.test(...)
```

If we remove it, do you expect naive Bayes to perform better or worse on the training data?

```r
car_wo_doors <- car[, -3]
nb.car.wod <- bnc('nb', 'class', dataset = car, smooth = 1)
p <- predict(nb.car.wod, car )
p <- predict(nb.car, car )
bnclassify::accuracy(p, car$class)
```

Correlated features

Is any of the class-conditional independece assumptions violated? E.g., check for `buying` and `maint` given class (use bnlearn):

```r
ci.test('maint', 'buying', 'class', car)
```

How does this violation affect classification performance? Let us look at an extreme example: perfect correlation among variables; let us add a copy of the `safety` feature to the data set.

- Should accuracy stay the same/degrade/improve? Run the code below.

```r
car2 <- cbind(car, safety2=car$safety)
nb.car2 <- bnc('nb', 'class', car2, smooth = 1)
p <- predict(nb.car2, car2)
bnclassify:::accuracy(p, car2$class)
```

Such assumptions are often violated and degrade the probability estimates, yet that does not necessarily lead to poor classification, as long as the true class has the highest probability.

- What happens if we add 10 copies of `safety` to the data set?

```r
sft <- car[, rep(6,10)]
colnames(sft) <- paste0('safety', 1:10)
car2 <- cbind(car, sft)
nb.car2 <- bnc('nb', 'class', car2, smooth = 1)
p <- predict(nb.car2, car2)
bnclassify:::accuracy(p, car2$class)
```

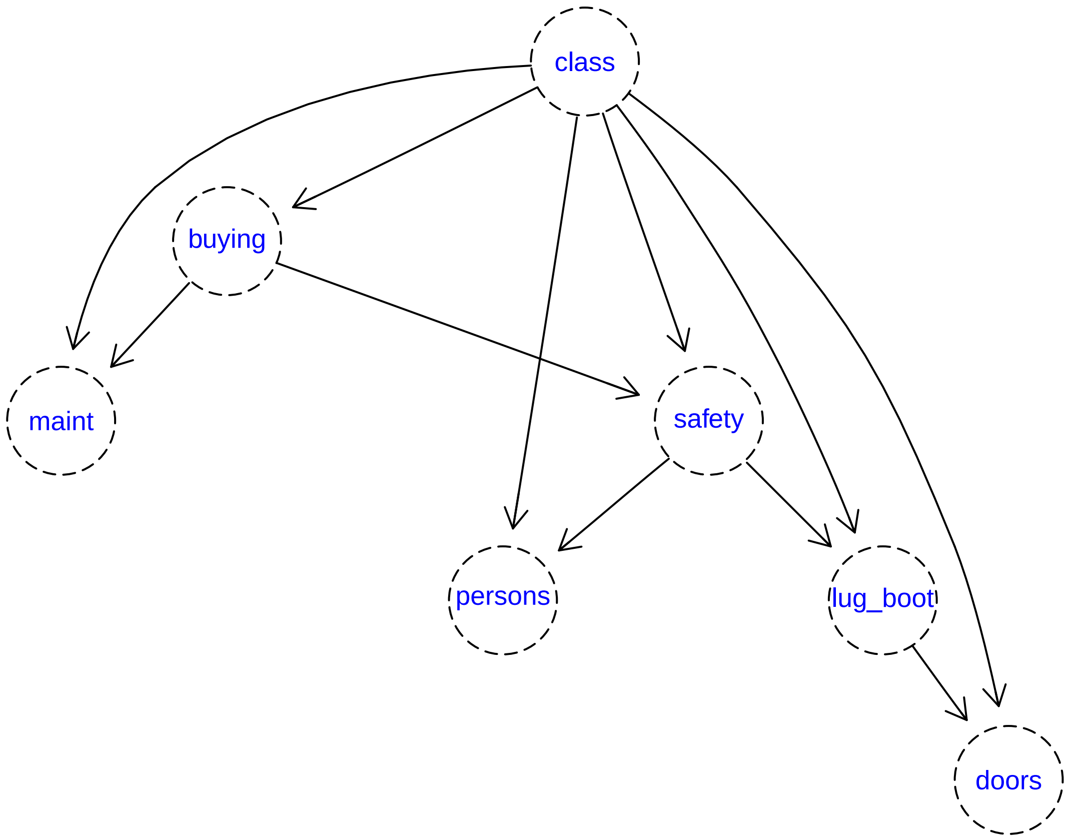
What are the possible approaches for handling feature correlation?

One-dependence augmented naive Bayes

The `tan_cl` function produces a tree augmented naive Bayes (TAN). It uses the Chow-Liu algorithm to maximize (penalized) likelihood in time quadratic in the number of features.

Let us call `tan_cl` without a score argument. The yields the maximum likelihood TAN.

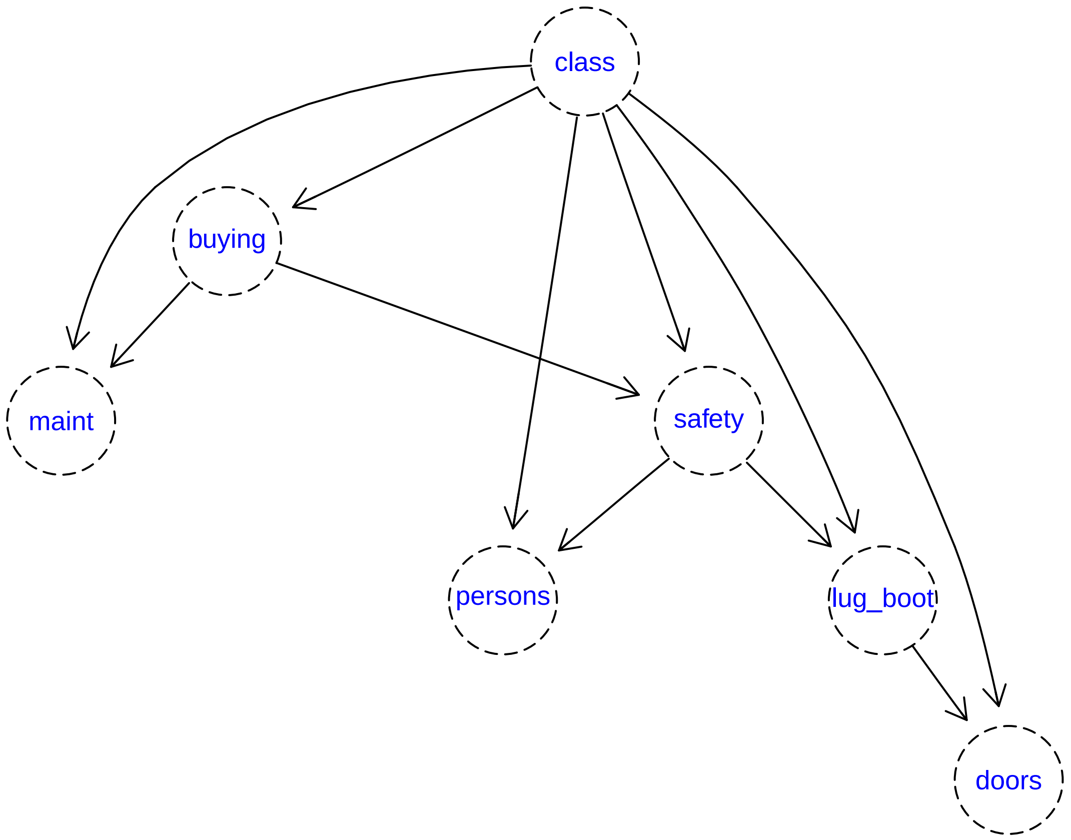
```r
tn <- tan_cl('class', car)
tn <- lp(tn, car, smooth = 1)
plot(tn)
```



- doors is independent of lug_boot given the class. Why is there an arc between these nodes? How many arcs added?
- Which feature is the root of the predictors tree?

Changing the root produces an equivalent DAG.

```
tns <- tan_cl('class', car, root = 'safety')
plot(tns)
```



```r
tns <- lp(tns, car, smooth = 1)
p <- predict(tn, car, prob = TRUE)
ps <- predict(tns, car, prob = TRUE)
identical(p, ps)
```

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

- What is the accuracy on the training data?

```r
p <- predict(tn, car )
bnclassify::accuracy(p, car$class)
```

Does this model have more free parameters than the naive Bayes? It its likelihood score higher?

```r
nparams(...)
nparams(...)
logLik(, car)
logLik(, car)
```

–>

Bayesian information criterion (BIC) and Akaike information criterion (AIC) scores penalize log-likelihood by a factor of model size

AIC $= LL(D) - |\theta|$

BIC $= LL(D) - \frac{\log(N)}{2}|\theta|$

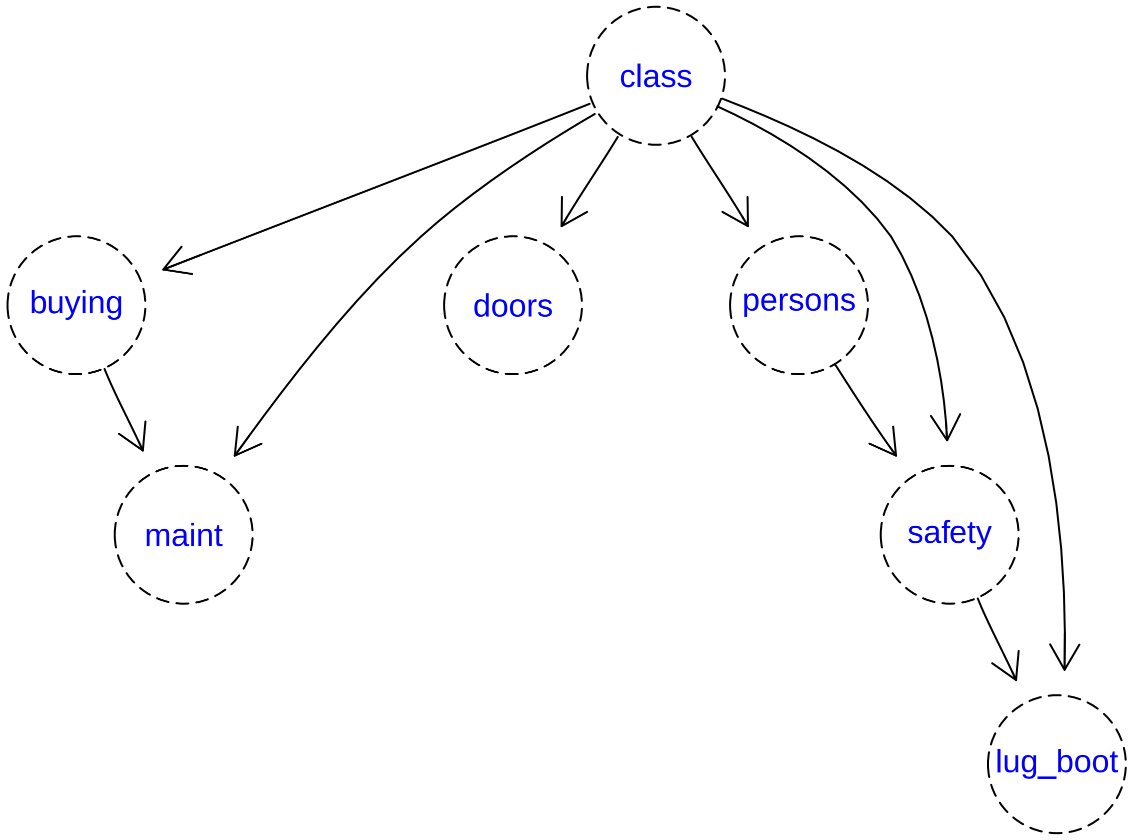
- Which is more restrictive?

An arc between X_i and X_j contributes $NI(X_i; X_j \mid C) - (r_i - 1)(r_j - 1)r_c$ to AIC and $NI(X_i; X_j \mid C) - \frac{\log(N)}{2}(r_i - 1)(r_j - 1)r_c$ to BIC.

- Can BIC and AIC omit spurious arcs from structure?

Let us call `tan_cl` with the aic score.

```
tn.aic <- tan_cl('class', car, score = 'aic')
tn.aic <- lp(tn.aic, car, smooth = 1)
plot(tn.aic)
```



- Is the predictor subgraph a tree?
- Which arcs are added?
- Is the AIC score of this network higher than that of the maximum likelihood TAN?

```
AIC(object = tn, car)
```

```
## [1] -13461.59
```

```
AIC(object = tn.aic, car)
```

```
## [1] -13438.59
```

- What quantity does the arc between lug_boot and safety contribute to AIC? Compute it using conditional mutual information and the number of added parameters.

```
sl.cmi <- bnclassify:::cmi(, , car, 'class')
# The number of parameters added is:
ap <- (3 - 1) * (3 - 1) * 4
N <-
N * sl.cmi - ap
```

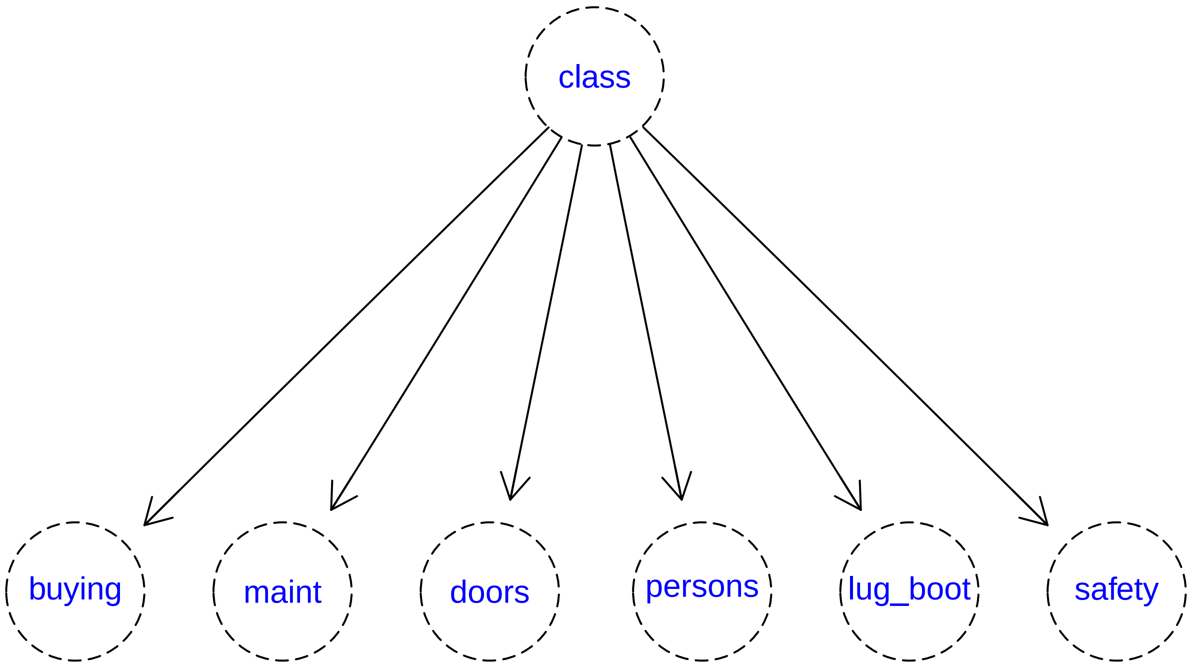
- Would BIC include this arc (between lug_boot and safety)?

We can get the score contribution of each arc with `local_ode_score`:

```
bnclassify:::local_ode_score_contrib('safety', 'lug_boot', 'class', car)
```

On the car data set, BIC adds no arcs.

```
tn.bic <- tan_cl('class', car, score = 'bic')
plot(tn.bic)
```



Comparing classifiers

Which of the above models would you chose? If TAN is more accurate on the training data, is it the better model?

- We can perform cross-validation with `cv`. Before we can use `tn.bic` we need to. . . ?

```
tn.bic <- ...
bnclassify::cv(list(nb.car, tn, tn.aic, tn.bic),  car, k = 10)
```

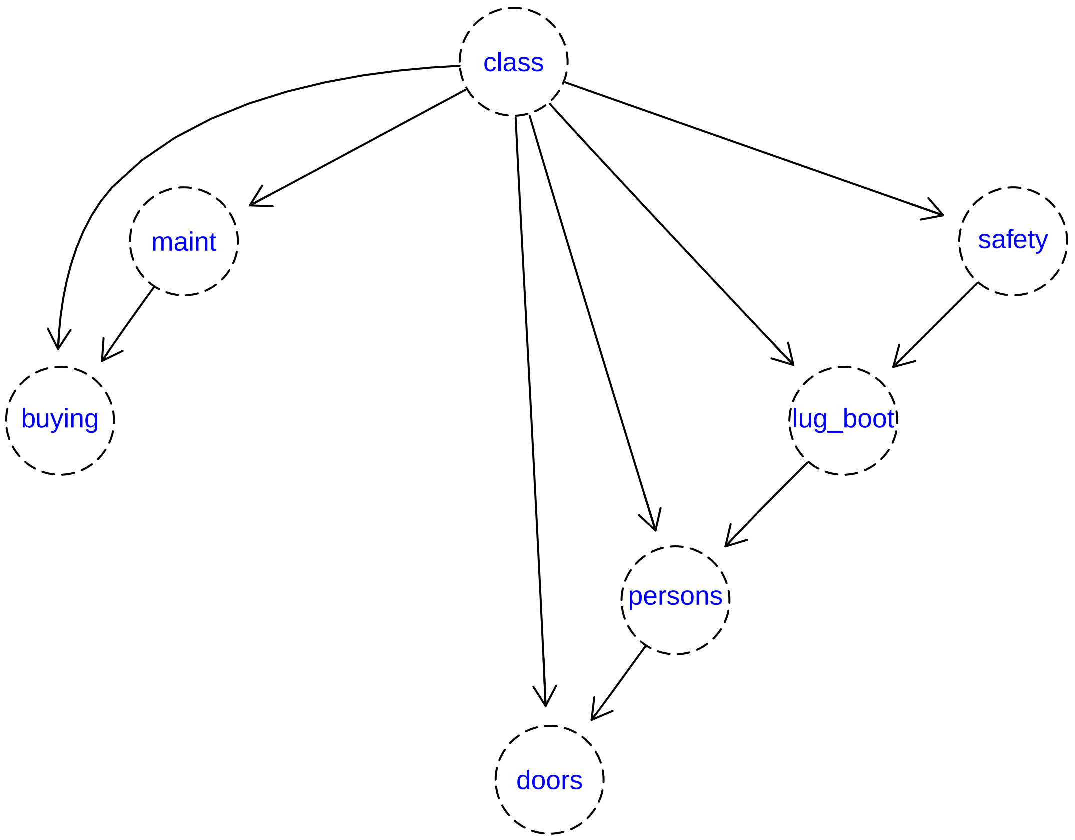
Passing multiple models to `cv` uses the same CV partitions to learn and test the classifiers. This allows for paired tests when comparing performance.

Wrapper structure learning

We can learn one-dependence estimators by maximizing structure accuracy. `tan_hc` greedily adds arcs starting from the naive Bayes until no arc improves accuracy by more than `epsilon = 0`.

```
set.seed(0)
tn.hc <- tan_hc('class', car, k = 10, epsilon = 0, smooth = 1)
```

- Plot the structure. Similar to Chow-Liu ODE structures?



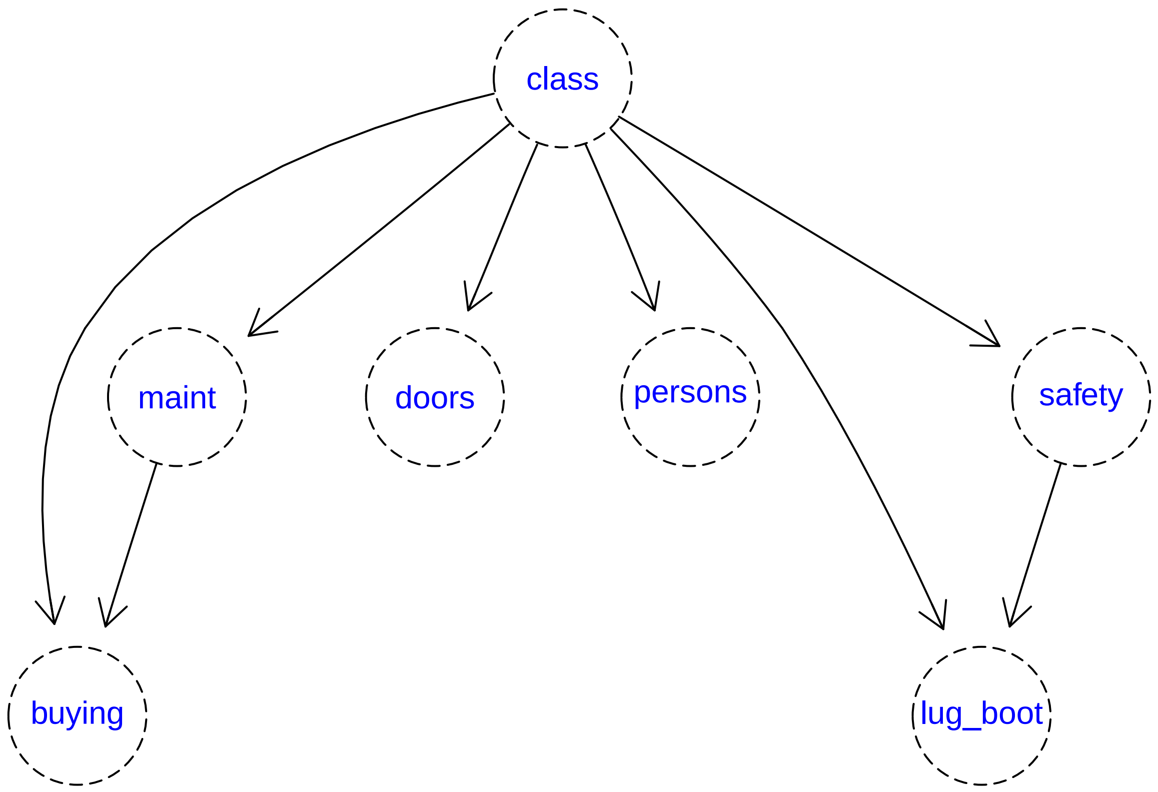
This is how structure score evolved.

```r
tn.hc$.greedy_scores_log
```

```
## [1] 0.8582183 0.9050902 0.9357734 0.9398197 0.9450322
```

- What if we increased `epsilon`?

```r
set.seed(0)
tn.hc2 <- tan_hc('class', car, k = 10, epsilon = , smooth = 1)
```



```
tn.hc2$.greedy_scores_log
```

```
## [1] 0.8582183 0.9050902 0.9357734
```

- What is the k argument for?

Wrappes are slower than Chow-Liu.

```
system.time(tan_hc('class', car, k = 10, epsilon = 0, smooth = 1))
```

```
##    user  system elapsed
##   0.325   0.008   0.333
```

```
system.time(tan_cl('class', car))
```

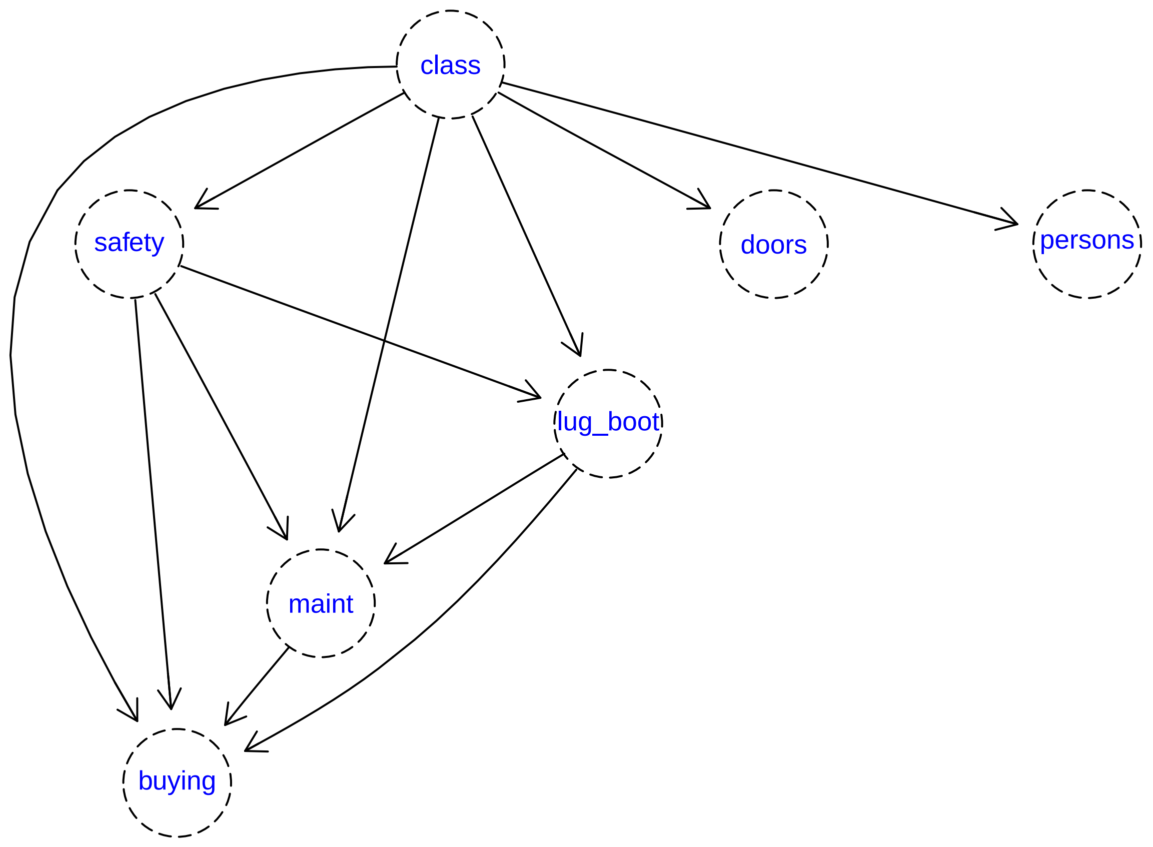
```
##    user  system elapsed
##   0.023   0.000   0.022
```

- How could get lower runtimes?

The semi-naive Bayes `bsej` starts from a naive Bayes and at each step it tries removing features or conditioning them on other features.

```
set.seed(0)
bs.car <- bsej('class', car, k = 10, epsilon = 0, smooth = 1)
```

- Plot the structure. How many arcs?



- What are the supernodes? Do they correspond to the hidden PRICE and TECH variables?

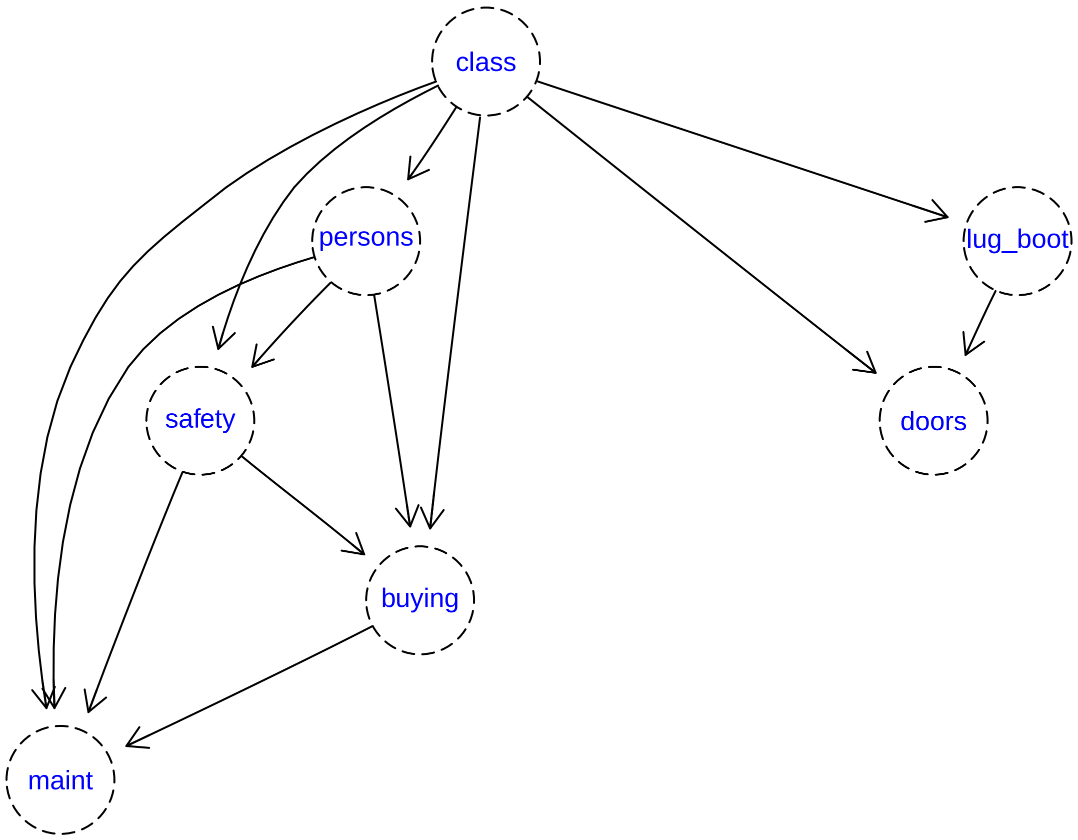
```
bnclassify:::is_supernode(c('maint', 'buying'), bs.car)
```

`fssj` to learns a semi-naive Bayes starting from an empty feature set.

- Call `fssj` (analogously to `bsej`).

```
set.seed(0)
fs.car <- fssj()
```

- Plot the structure.



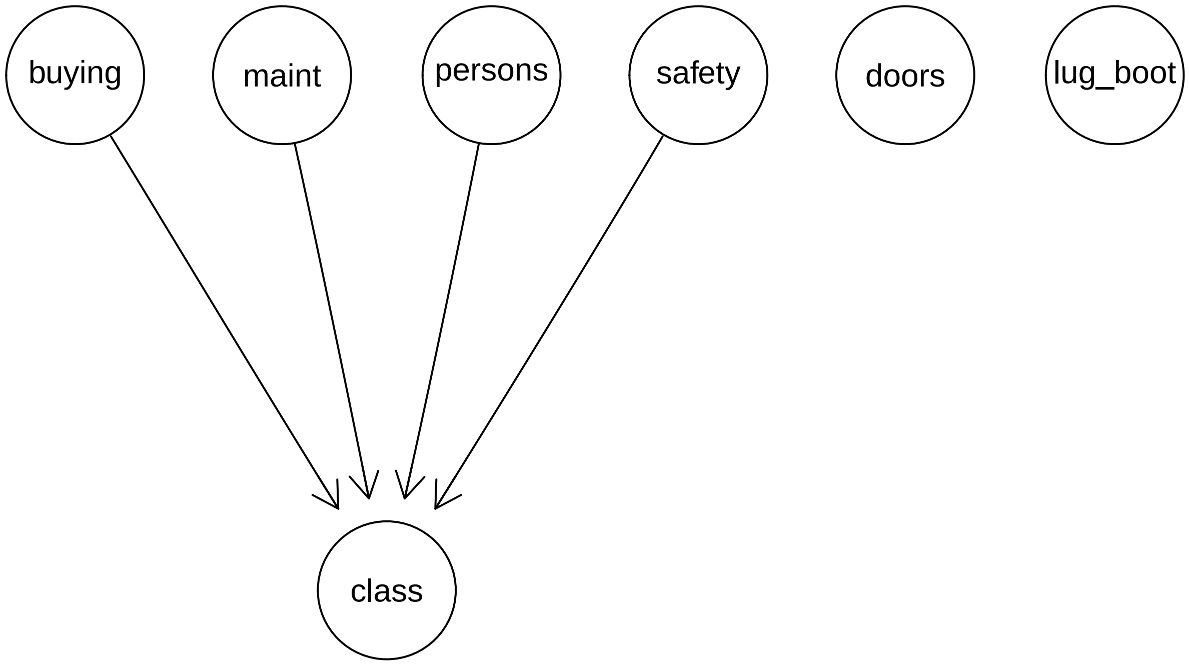
- Were any variables omitted?

Markov blanket

Call `iamb` to get the equivalence class of the entire network.

- What is the Markov blanket of the class?

```
ia.car <- iamb(car)
plot(ia.car)
```



We can predict using `gRain`'s exact inference. First, get the class plus its Markov blanket subgraph.

```
mb.class <- mb(ia.car, 'class')
mb.car <- bnlearn::subgraph(ia.car, c(mb.class, 'class'))
mb.car <- bn.fit(mb.car, car[, nodes(mb.car)])
mb.grain <- as.grain(mb.car)
```

- Is it more accurate on the training data than naive Bayes?

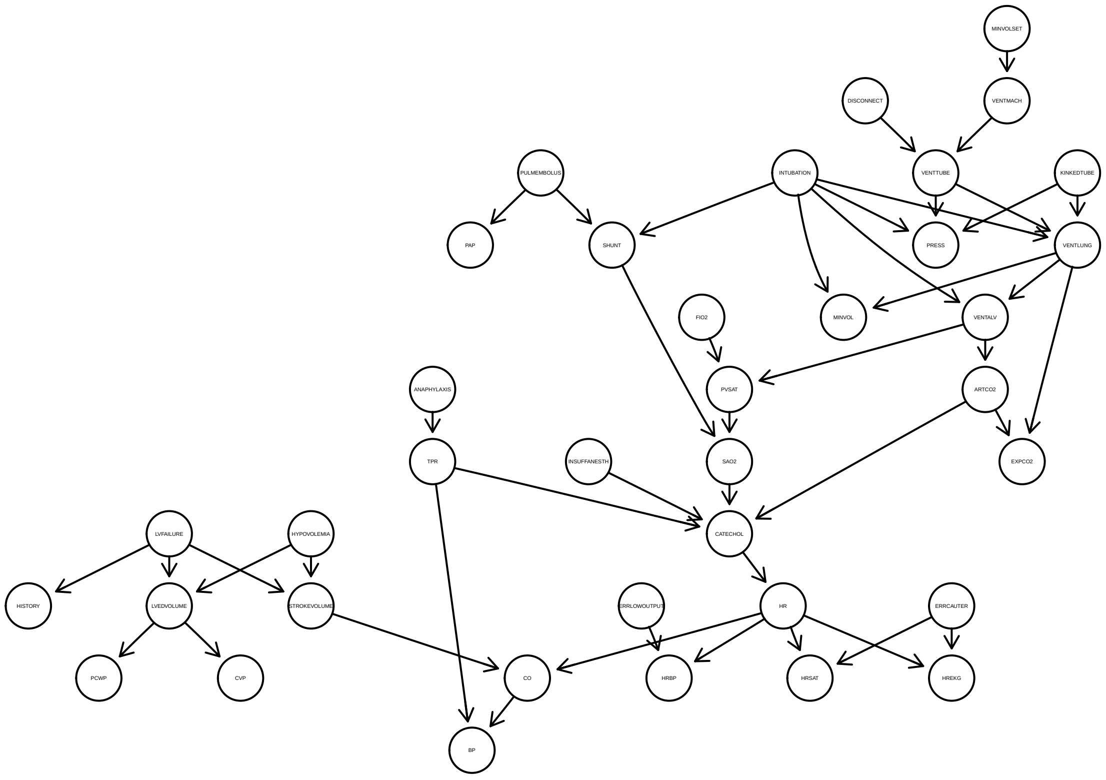
```
p <- gRain::predict.grain(mb.grain,  'class', newdata  = car)
fp <- factor(p$pred$class, levels = levels(car$class))
bnclassify:::accuracy(fp, car$class)
```

```
## [1] 0.8923611
```

The ALARM network

Consider the alarm network

```
load('data/alarm.rda')
alarm.fit <- bn
bn <- as.bn(as.grain(alarm.fit))
plot(as.grain(alarm.fit))
```



Inspect the network

- How many nodes?
- How many arcs?
- What is the average neighborhood size?
- What are the v-structures in the graph? See `vstruct` in `bnlearn`.
- What are the parents of HISTORY?
- What is the Markov blanket of LVED VOLUME?
- What is the neighborhood of STROKEVOLUME?
- Are the local distributions discrete or Gaussian?

Recover network structure

1. Sample from the original network
2. Run a structure learning algorithm of your choice
3. Compare its structural distance to the original network
4. is it an I-map of the original distribution (i.e., network?)?
5. Is its likelihood score higher than that of the original network?

Improve learning

1. Try different options of the algorithm you just used (e.g,. independence tests)
2. Try other algorithms (score + search, constraint-based, hybrid)
3. Which algorithms' models score similarly to the original network?
4. Which has the best cross-validated CV score?

Inference

Go back to the original network

- What is the marginal probability of HYPOVOLEMIA?
- What is the map assignment to INSUFF ANESTH and CATECHOL?
- What is the joint probability of SHUNT and CATECHOL?
- Which is the largest clique in the junction tree?
- Find the marginal probability of HYPOVOLEMIA with sampling-based inference. Is it close to the exact probability?
- Find the MAP assignment to INSUFF ANESTH and CATECHOL with sampling-based inference

Final notes

Some useful functions

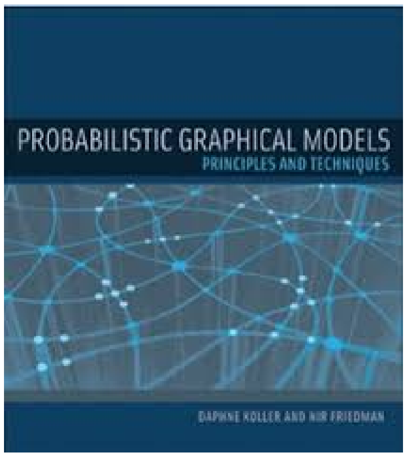
For analysing network structures

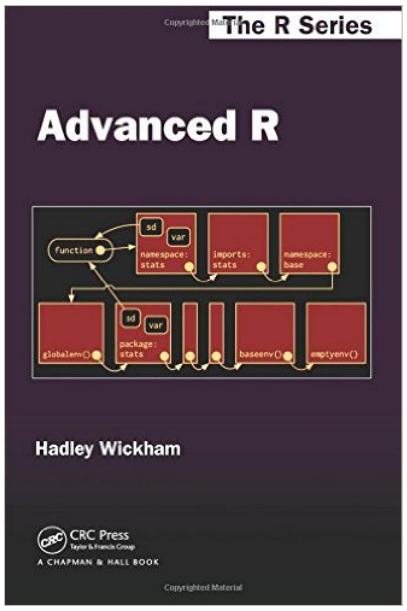
```
bnlearn::mb
bnlearn::ancestors
bnlearn::descendants
bnlearn::children
bnlearn::nparams
bnlearn::model2network
bnlearn::skeleton
bnlearn::compare
bnlearn::cpdag
bnlearn::node.ordering
bnlearn::nodes
bnlearn::dsep
```

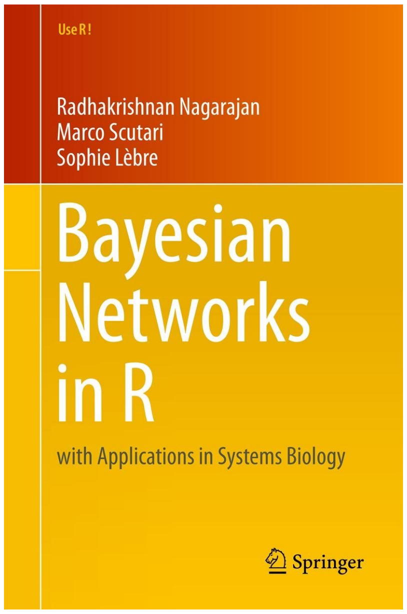
For manipulating network structures

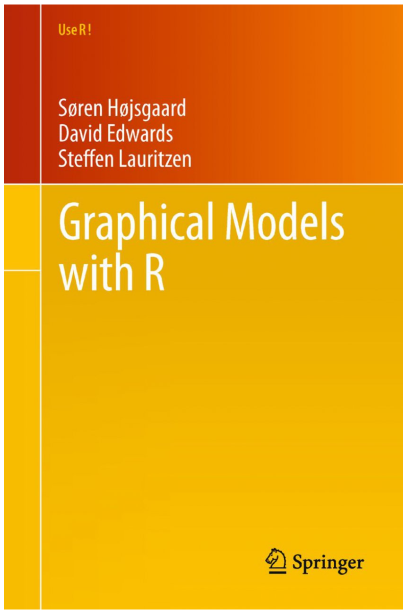
```
gRbase::random_dag
bnlearn::random.graph
bnlearn::empty.graph

bnlearn::set.arc
bnlearn::cextend
gRbase::moralize

gRbase::triangulate
gRbase::getCliques
```

Books


PROBABILISTIC GRAPHICAL MODELS
PRINCIPLES AND TECHNIQUES
DAPHNE KOLLER AND NIR FRIEDMAN


The R Series
Advanced R
Hadley Wickham
CRC Press
A CHAPMAN & HALL BOOK


Use R!
Radhakrishnan Nagarajan
Marco Scutari
Sophie Lèbre
Bayesian Networks in R
with Applications in Systems Biology
Springer


Use R!
Søren Højsgaard
David Edwards
Steffen Lauritzen
Graphical Models with R
Springer

Other Bayesian networks software

- Free: Banjo, Genie, Elvira, Open Markov, . . .
- Commercial: BayesiaLab, Hugin Expert, Netica, . . .
- A list (last updated 2014): http://www.cs.ubc.ca/~murphyk/Software/bnsoft.html
- A survey (2004): http://www.csse.monash.edu.au/bai/book1e/appendix_b.pdf