Bayesian Networks

Alberto Garfagnini

Università di Padova

AA 2020/2021 - Stat Lect. 14

The Truck Driver Example

- a truck driver is due to make a 800 km trip
- we analyze the risk of his falling asleep while driving

- there may be causal relationships between:

1) the driver's sleep (did he sleep well and for more than 7 hours the night before ?)
2) his perceived fatigue (does he feel tired at the beginning of the trip ?)
3) the risk of falling asleep while driving

Current situation

- the truck driver feels tired at the beginning of the trip

➜ weather or not this is due to a bad sleep the night before, or any other reason, is of no use to evaluate the risk

- the driver feels perfectly fit before starting to drive

➜ the quality of sleep the night before has no influence on his current condition

➜➜ the risk of falling asleep is conditionally independent of the quality of his sleep, given the driver's current fatigue

The Truck Driver Example

Our formal model:

- we model it with binary variables which tell us if
  - X_1: the truck driver slept well the night before
  - X_2: he feels tired at the beginning of the trip
  - X_3: he will fall asleep while driving
- but, the quality of sleep the night before has no influence on his current condition

$$P\left(X_3 \mid X_1, X_2\right) = P\left(X_3 \mid X_2\right)$$

therefore:

$$\begin{aligned} P\left(X_1, X_2, X_3\right) &= P\left(X_1 \mid X_2, X_3\right) P\left(X_2, X_3\right) \\ &= P\left(X_3 \mid X_2\right) P\left(X_2 \mid X_1\right) P\left(X_1\right) \end{aligned}$$

The Doped Athlete Example

- during a sports competition, each athlete undergoes two doping tests
  - test A is a blood test
  - test B a urine test
- the two tests are carried out in two different laboratories, no contact between the to labs is possible

Current situation

- the results of the two tests are not independent variables:
  - if test A is positive ➜ the participant is likely to have used a banned product ➜ test B will probably be also positive
  - a participant has has taken and detectable substance

➜ tests A and B can be considered independent, since the two laboratories use different detection methods

- a participant has not taken any prohibited substance

➜ tests A and B can be again considered independent ➜ they may give a negative response according to the test efficacy

➜➜ the results of both tests are conditionally independent, given the status of the tested athlete

The Doped Athlete Example

Our formal model:

- we model it with binary variables which tell us if
  - X_1: the athlete is *clean* or not
  - X_2: the result of test A
  - X_3: the result of test B
- let's write the conditional probability

$$P\left(X_3 \mid X_2, X_1\right) = P\left(X_3 \mid X_1\right)$$

➜ knowing whether the athlete has taken the substance is enough information to estimate the chances of test B being positive

- the symmetric equation holds

$$P\left(X_2 \mid X_3, X_1\right) = P\left(X_2 \mid X_1\right)$$

combining the results:

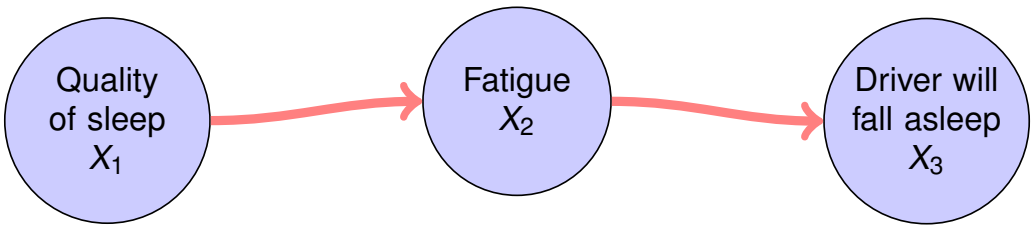
$$\begin{aligned} P\left(X_1, X_2, X_3\right) &= P\left(X_3 \mid X_2, X_1\right) P\left(X_2 \mid X_1\right) P\left(X_1\right) \\ &= P\left(X_3 \mid X_1\right) P\left(X_2 \mid X_1\right) P\left(X_1\right) \end{aligned}$$

Discrete Bayesian Networks

- both examples can be expressed in a Bayesian network

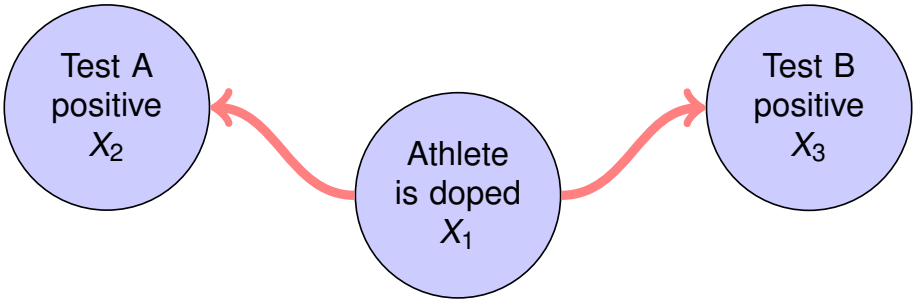
The Truck Driver Example

- that there is an influence of variable X_1 on variable X_2, and of variable X_2 on variable X_3



The Doped Athlete Example

- X_2 and X_3 are conditionally independent given X_1



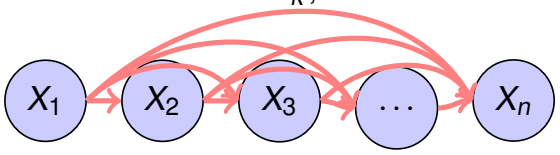
Bayesian Network

Definition

- given
  - n random variables $X_1, X_2, \ldots, X_n$
  - a directed acyclic graph with n numbered nodes
  - suppose node j of the graph associated to the X_j variable
- the graph is a Bayesian network, representing the variables $X_1, X_2, \ldots, X_n$, if

$$P(X_1, X_2, \ldots, X_n) = \prod_{j=1}^{n} P\left(X_j \mid \text{parents}(X_J)\right)$$

where $\text{parents}(X_j)$ denotes the set of all variables X_k, such that there is an arc from node k to node j



Proposition

- any joint probability distribution may be represented by a Bayesian network

$$\begin{aligned} P\left(X_1 \mid X_2, \ldots, X_n\right) &= P(X_1)\, P\left(X_2 \ldots X_n \mid X_1\right) \\ &= P(X_1)\, P(X_2 \ldots X_1)\, P\left(X_3 \ldots X_n \mid X_1 X_2\right) \\ &= \ldots \\ &= P(X_1)\, P\left(X_2 \mid X_1\right) P\left(X_3 \mid X_2, X_1\right) \ldots P\left(X_n \mid X_1, X_2, \ldots, X_{n-1}\right) \end{aligned}$$

Bayesian Networks in R: case study

The transportation means survey

- let's consider an hypothetical survey whose aim is to investigate the usage patterns of different means of transport, with a focus on private cars and public trains or buses
- each regular commuting individual fills a questionnaire on the following six discrete variables

  Age (`A`): below 30 (`young`), between 30 and 60 (`adult`) greater than 60 (`senior`)

  Sex (`S`): male (M) or female (F)

  Education (`E`): highest individual degree between high school (`high`) and university or higher (`uni`)

  Occupation (`O`): weather the individual is an employee (`empl`) or a self-employed worker (`self`)

  Residence (`R`): the size of the city the individual lives in, a (`small`) or (`big`) town

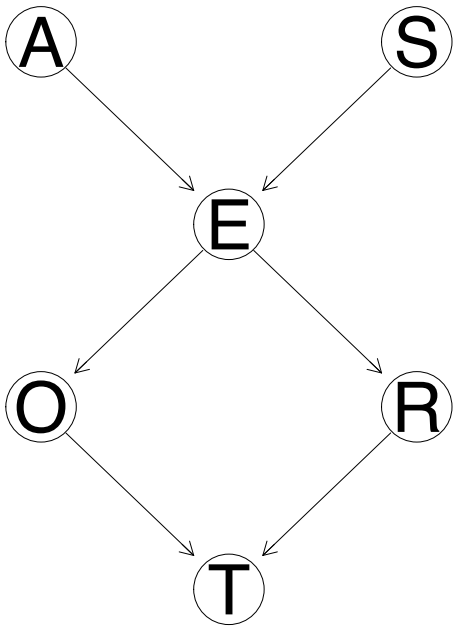
  Travel (`T`): the means of transport flavored by the individual, `car`, `train` or `bus`
- the variables can be grouped into demographic indicators (`Age` and `Sex`), socioeconomic indicators (`Education`, `Occupation` and `Residence`) and the target of the survey (`Travel`)

Travel Survey in bnlearn

- we use the bnlearn R package to build a directed acyclic graphs (DAG) that describes the network

```
library(bnlearn)

tus_dag <- empty.graph(nodes = c("A", "S",
                                 "E", "O", "R", "T"))
tus_dag <- set.arc(tus_dag, from = "A", to = "E")
tus_dag <- set.arc(tus_dag, from = "S", to = "E")
tus_dag <- set.arc(tus_dag, from = "E", to = "O")
tus_dag <- set.arc(tus_dag, from = "E", to = "R")
tus_dag <- set.arc(tus_dag, from = "O", to = "T")
tus_dag <- set.arc(tus_dag, from = "R", to = "T")
```




```
tus_dag
#> Random/Generated Bayesian network
#> model:
#>  [A][S][E|A:S][O|E][R|E][T|O:R]
#> nodes:                                 6
#> arcs:                                  6
#>   undirected arcs:                     0
#>   directed arcs:                       6
#> average markov blanket size:           2.67
#> average neighbourhood size:            2.00
#> average branching factor:              1.00
#> generation algorithm:                  Empty
```

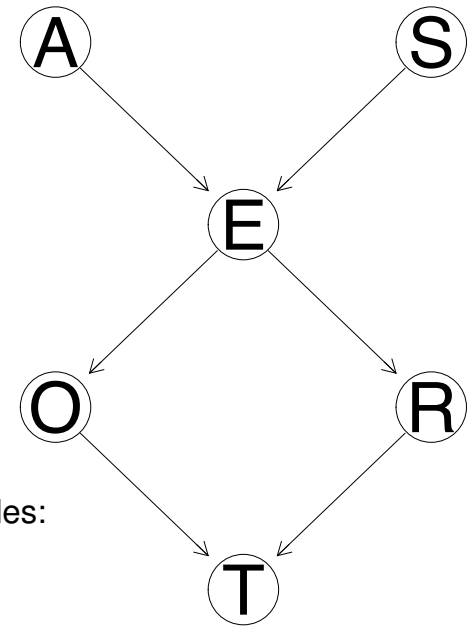

Exploring the Travel Survey DAG bnlearn

- show the nodes of a graph:

```
nodes(tus_dag)
[1] "A" "S" "E" "O" "R" "T"
```

- examine the nodes close to a target

```
# The neighbourhood of 'E'
nbr(tus_dag, "E")
#> [1] "A" "S" "O" "R"
parents(tus_dag, "E")
#> [1] "A" "S"
children(tus_dag, "E")
#> [1] "O" "R"
```



- look for roots (no parents) and leaves (no children) nodes:

```
root.nodes(tus_dag)
#> [1] "A" "S"
leaf.nodes(tus_dag)
#> [1] "T"
```

- and plot the graph

```
library(Rgraphviz)
graphviz.plot(plant)
```

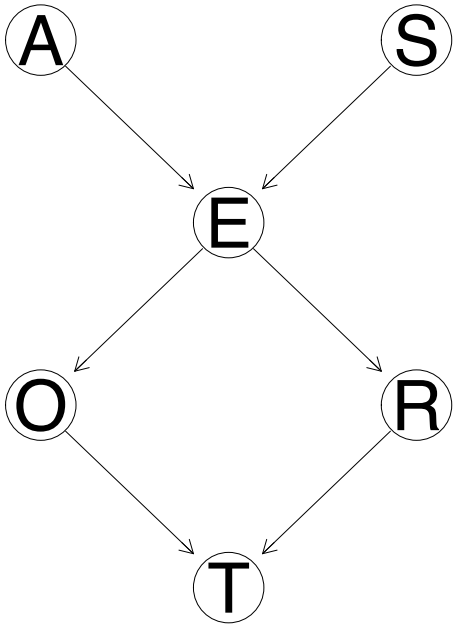
Travel Survey in bnlearn

- another way to create the network is using the model formula interface provided by `modelstring`

```
dag <- empty.graph(nodes = c("A", "S", "E", "O", "R", "T"))

dag2 <- model2network("[A][S][E|A:S][O|E][R|E][T|O:R]")

all.equal(tus_dag, dag2)
[1] TRUE
```



- we now define the levels of the nodes, i.e. the discrete values defined on a non-ordered set

```
A_lvl <- c("young", "adult", "old")
S_lvl <- c("M", "F")
E_lvl <- c("high", "uni")
O_lvl <- c("emp", "self")
R_lvl <- c("small", "big")
T_lvl <- c("car", "train", "other")
```


Travel Survey in bnlearn

- To complete the BN modelling the survey➜ specify the joint probabilities

```
A_prob <- array(c(0.30, 0.50, 0.20), dim = 3,
                dimnames = list(A = A_lvl))
S_prob <- array(c(0.60, 0.40), dim = 2, dimnames = list(S = S_lvl))

O_prob <- array(c(0.96, 0.04, 0.92, 0.08), dim = c(2, 2),
                dimnames = list(O = O_lvl, E = E_lvl))
R_prob <- matrix(c(0.25, 0.75, 0.20, 0.80), ncol = 2,
                 dimnames = list(R = R_lvl, E = E_lvl))

E_prob <- array(c(0.75, 0.25, 0.72, 0.28, 0.88, 0.12,
                  0.64, 0.36, 0.70, 0.30, 0.90, 0.10), dim = c(2, 3, 2),
                dimnames = list(E = E_lvl, A = A_lvl, S = S_lvl))
T_prob <- array(c(0.48, 0.42, 0.10, 0.56, 0.36, 0.08,
                  0.58, 0.24, 0.18, 0.70, 0.21, 0.09), dim = c(3, 2, 2),
                  dimnames = list(T = T_lvl, O = O_lvl, R = R_lvl))
```

- and finally associate the probabilities to the BN model

```
cpt <- list(A = A_prob, S = S_prob, E = E_prob, O = O_prob,
            R = R_prob, T = T_prob)
bn <- custom.fit(tus_dag, cpt)
```

- variables that are not linked by an arc are conditionally independent

➜ we can factorise the global distribution

$$P(A,S,E,O,R,T) = P(A)\cdot P(S)\cdot P(E \mid A,S)\cdot P(O \mid E)\cdot P(R \mid E)\cdot P(T \mid O,R)$$

Travel Survey : estimate the probability table

- we knew the DAG and the probabilities defining the BN ➜ BNs are used as expert systems
- but in most cases, the parameters of the local distributions will be inferred (i.e. learned) from the observed sample

```
survey <- read.table("survey.txt", header = TRUE,
                     stringsAsFactors = TRUE)

head(transp_survey, n=3)
#>       A     R    E   O S     T
#> 1 adult   big high emp F   car
#> 2 adult small  uni emp M   car
#> 3 adult   big  uni emp F train

tail(transp_survey, n=3)
#>          A   R    E   O S     T
#> 498   old big high emp M train
#> 499 adult big high emp F other
#> 500 adult big high emp M other
```

- the conditional probabilities can be estimated looking at the corresponding empirical frequencies in the data set

$$P\left(O = \text{emp} \,\middle|\, E = \text{high}\right) = \frac{P(O = \text{emp}, E = \text{high})}{P(E = \text{high})}$$
$$= \frac{\text{number of observations for which O = emp and E = high}}{\text{number of observations for which E = high}}$$

Travel Survey : estimate the probability table

- the `bn.fit()` function computes the classic frequentist and maximum likelihood estimates from the data
- it complements the `custom.fit()` function which constructs a BN using a set of custom parameters specified by the user

```
travel_dag <- model2network("[A][S][E|A:S][O|E][R|E][T|O:R]")

bn.mle <- bn.fit(travel_dag, data = transp_survey,
                 method = "mle")
```

- as an alternative, the same conditional probabilities can be estimated in the Bayesian framework, using their posterior distributions

```
bn.bayes <- bn.fit(travel_dag, data = transp_survey,
                   method = "bayes", iss = 10)
```

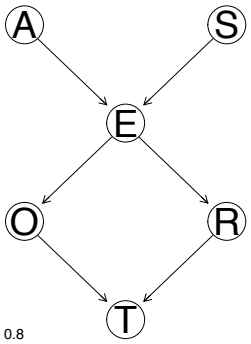
- the `iss` optional argument (imaginary sample size) determines how much weight is assigned to the prior distribution compared to the data when computing the posterior
- `iss` is typically chosen between 1 and 15, to allow the prior distribution to be easily dominated by the data (see bnlearn documentation)

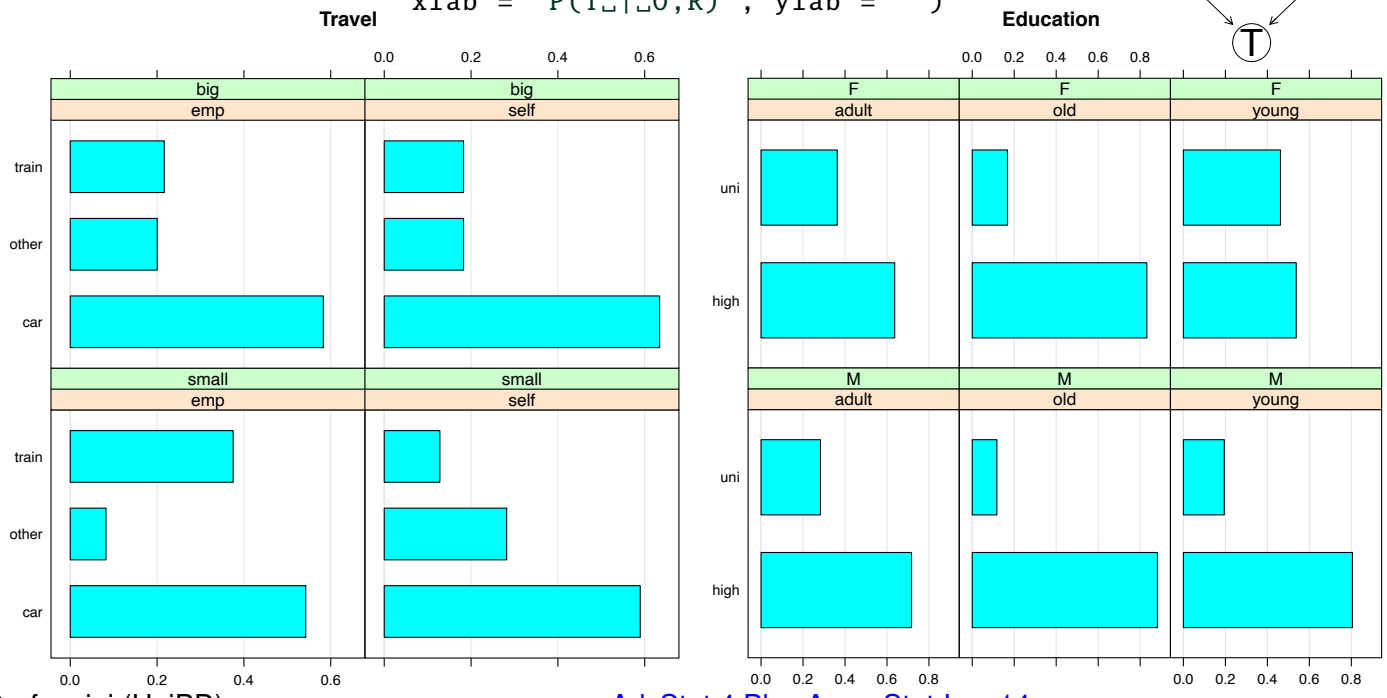
Plotting the probability between links

- we can plot the probabilities associated to each link of the network

```
bn.fit.barchart(bn.bayes$E, main = "Education",
                xlab = "P(E␣|␣A,S)", ylab = "")

bn.fit.barchart(bn.bayes$T, main = "Travel",
                xlab = "P(T␣|␣O,R)", ylab = "")
```





Investigating a DAG structure from data

- so far we have assumed that the DAG underlying the BN is known

➜ we rely on prior knowledge on the phenomenon to decide which arcs are present in the graph and which are not ➜ expert system

- the structure of the DAG itself may be the object of our investigation

➜ in genetics and systems biology it is common to reconstruct the molecular pathways and networks underlying complex diseases and metabolic processes

- learning the DAG of a BN is a complex task

1) the space of the possible DAGs is very big ➜ it grows exponentially with the number of nodes
2) this space is very different from real spaces : it is not continuous and has a finite number of elements ➜ ad-hoc algorithms are required to explore it

- two classes of statistical criteria used to evaluate DAGs

1) conditional independence tests
2) network scores

Conditional Independence Test

- arcs encode a probabilistic dependence ➜ conditional independence tests can be used to verify if that probabilistic dependence is supported by the data

E ➜ T

- H$_\circ$: Travel is independent of Education
- use the log-likelihood ratio G^2

$$P\left(T, E \,\middle|\, O, R\right) = \sum_{t \in T} \sum_{e \in E} \sum_{k \in O \times R} n_{tek} \log \frac{n_{tek}\, n_{++k}}{n_{t+k}\, n_{+ek}},$$

  - the use of a "+" subscript denotes the sum over that index :
  - n_{t+k} = sum over the second index, $e \in E$
  - n_{++k} = sum over the first ($t \in T$) and second ($e \in E$) indexes, respectively
- or Pearson's X^2:

$$P\left(T, E \,\middle|\, O, R\right) = \sum_{t \in T} \sum_{e \in E} \sum_{k \in O \times R} \frac{(n_{tek} - m_{tek})^2}{m_{tek}} \quad \text{with } m_{tek} = \frac{n_{t+k}\, n_{+ek}}{n_{++k}}$$

- both tests have an asymptotic χ^2 distribution under H$_\circ$

Travel Survey: evaluate DAG arcs

Mutual Information, log-likelihood ratio G^2

```
transp_survey <- read.table("survey.txt", header = TRUE,
                            stringsAsFactors = TRUE)

ci.test("T", "E", c("O", "R"), test = "mi", data = transp_survey)

#>       Mutual Information (disc.)
#>
#> data:  T ~ E | O + R
#> mi = 9.8836, df = 8, p-value = 0.2733
#> alternative hypothesis: true value is greater than 0
```

Pearson's X^2

```
ci.test("T", "E", c("O", "R"), test = "x2", data = transp_survey)

#>       Pearson's X^2
#>
#> data:  T ~ E | O + R
#> x2 = 8.2375, df = 8, p-value = 0.4106
#> alternative hypothesis: true value is greater than 0
```

- both tests return very large p-values ➜ the dependence relationship encoded by $E \times T$ is not significant given the current DAG structure

Network Scores

- network scores focus on the DAG as a whole
- they provide a statistical measurement of how well the DAG mirrors the dependence structure of the data

Bayesian Information criterion (BIC)

$$
\begin{aligned}
\text{BIC} &= \log P(A,S,E,O,R,T) - \frac{d}{2}\log n \\
&= \log P(A) - \frac{d_A}{2}\log n + \log P(S) - \frac{d_S}{2}\log n \\
&+ \log P(E \mid A,S) - \frac{d_E}{2}\log n + \log P(O \mid E) - \frac{d_O}{2}\log n \\
&+ \log P(R \mid E) - \frac{d_R}{2}\log n + \log P(T \mid O,R) - \frac{d_T}{2}\log n
\end{aligned}
$$

- with n the sample size
- d the number of parameters of the network
- d_A, d_S, d_E, d_O, d_R and d_T the number of parameters associated with each node

Travel Survey: evaluate scores

several scores are available in `bnlearn`:

- Bayesian Information criterion (BIC)

```
score(travel_dag, data = transp_survey, type = "bic")
#> [1] -2012.687
```

- Bayesian Dirichlet equivalent uniform (BDe)

```
score(travel_dag, data = transp_survey, type = "bde")
#> [1] -2015.647
```

- using such scores it is possible to compare different DAGs and investigate which of them fits the data better

```
nparams(travel_dag, transp_survey)
[1] 21

dag_new <- set.arc(travel_dag, from = "E", to = "T")
nparams(dag_new, transp_survey)
[1] 29

score(dag_new, data = transp_survey, type = "bic")
[1] -2032.603
```

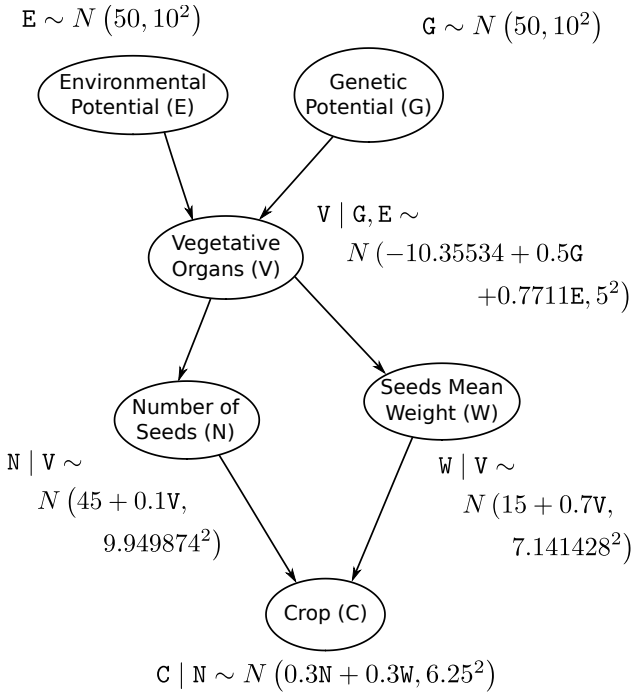
➜ adding E➜ T is not beneficial
the increase in $\log P(A,S,E,O,R,T)$ is not sufficient to offset the heavier penalty from the additional parameters

Continuous Bayesian Networks

- focus on modelling continuous data under a multivariate Normal distribution hypothesis
- we are interested in the analysis of a particular plant and study
  - the potential of the plant and the environment
  - the production of vegetative mass
  - the harvested grain mass, i.e. the *crop*

The plant network

- environmental potential `(E)`
- genetic potential `(G)`
- vegetative organs `(V)`
- number of seeds `(N)`
- seeds mean weight `(W)`
- crop `(C)`



The Plant example: network structure

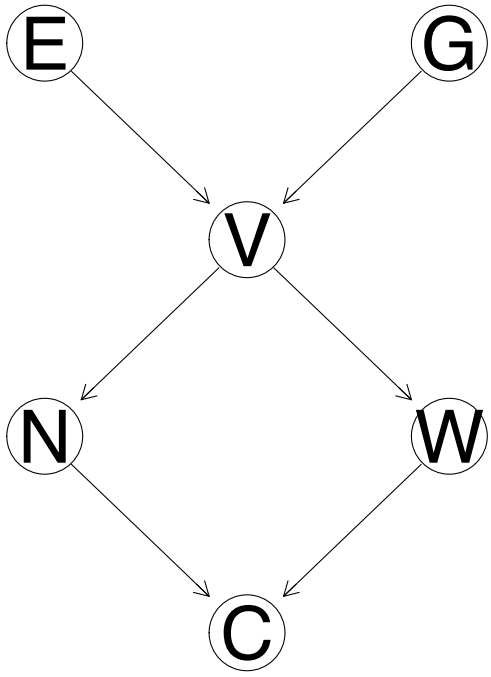
- let's build the DAG bayesian network:

```
plant_dag <- model2network("[G][E][V|G:E][N|V][W|V][C|N:W]")
plant_dag

#>   Random/Generated Bayesian network

#>   model:
#>    [E][G][V|E:G][N|V][W|V][C|N:W]
#>   nodes:                                6
#>   arcs:                                 6
#>     undirected arcs:                    0
#>     directed arcs:                      6
#>   average markov blanket size:          2.67
#>   average neighbourhood size:           2.00
#>   average branching factor:             1.00
#>
#>   generation algorithm:                 Empty

nparams(plant)
#> [1] 18
```



The Plant example: connection probabilities

- to make quantitative statements about the behavior of the variables in the BN, we need to completely specify their joint probability distribution
- if we are modelling n variables we must specify n means, n variances and $n(n-1)/2$ correlation coefficients

```
disE <- list(coef = c("(Intercept)" = 50), sd = 10)
disG <- list(coef = c("(Intercept)" = 50), sd = 10)
disV <- list(coef = c("(Intercept)" = -10.35534,
             E = 0.70711, G = 0.5), sd = 5)
disN <- list(coef = c("(Intercept)" = 45,
             V = 0.1), sd = 9.949874)
disW <- list(coef = c("(Intercept)" = 15,
             V = 0.7), sd = 7.141428)
disC <- list(coef = c("(Intercept)" = 0,
             N = 0.3, W = 0.7), sd = 6.25)
dis.list = list(E = disE, G = disG, V = disV,
                N = disN, W = disW, C = disC)

plant <- custom.fit(plant_dag, dist = dis.list)
```

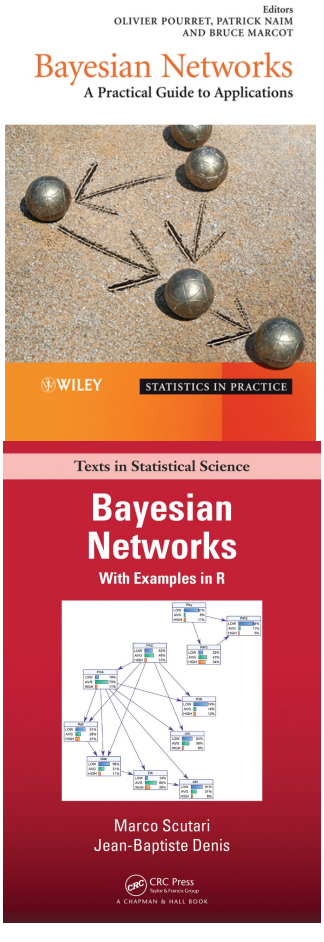

References

R packages

- bnlearn https://cran.r-project.org/web/packages/bnlearn/, https://www.bnlearn.com/
- BayesNetBP https://cran.r-project.org/web/packages/BayesNetBP/index.html
- bnstruct https://github.com/sambofra/bnstruct

Books

- O. Pourret et al, *Bayesian Networks, A Practical Guide to Applications*, J.Wiley and Sons, 2008, ISBN 978-0-470-06030-8
- M. Scutari, J.B. Denis, *Bayesian Networks with Examples in R*, CRC Press, 2015, ISBN 978-1-4822-2559-4



New R release 4.1.0

- released on May 18, 2021:
  https://cran.r-project.org/src/base/R-4/R-4.1.0.tar.gz

```
From: Peter Dalgaard
Subject: [Rd] R 4.1.0 is released
Date: Tue May 18 10:06:24 CEST 2021
To: r-announce@r-project.org, Cc: r-devel@r-project.org

The build system rolled up R-4.1.0.tar.gz (codename "Camp Pontanezen")
this morning.

This is a major update, notably containing the new native pipe operator
"|>" and shorthand inline functions "\(x) x+1".

The list below details the changes in this release.

You can get the source code from

https://cran.r-project.org/src/base/R-4/R-4.1.0.tar.gz

or wait for it to be mirrored at a CRAN site nearer to you.

Binaries for various platforms will appear in due course.


For the R Core Team,
Peter Dalgaard
```


New R 4.1.0 features: the pipe operator, |>

- introduced with the `magrittr` package in 2014, and building upon Unix / linux scripting languages syntax and F# functional programming languages (like F#)
- decrease development time and improve code readability and maintainability
- Example:

- with separate functions:

```
raw_data = rnorm(100, mean = 4, sd = 1)
density_summary = density(raw_data);  plot(density_summary)
```

- with nested function calls:

```
plot(
  density(
    rnorm(100, mean = 4, sd = 1)
  )
)
```

- with the `magrittr` package

```
# R version up to 4.0.5
library("magrittr")
rnorm(100, mean = 4, sd = 1) %>% density() %>% plot()
```

- with the new R pipe operator, | >:

```
# R version 4.1.0
rnorm(100, mean = 4, sd = 1) |> density() |> plot()
```

New R 4.1.0 features: %>% versus |>

- Note: |> is not a drop-in replacement for all uses of %>%
- The %>% allowed function calls to be written with or without parentheses

```
letters %>% head()
# [1] "a" "b" "c" "d" "e" "f"

letters %>% head
# [1] "a" "b" "c" "d" "e" "f"
```

- with the native pipe the parentheses must be present

```
# R 4.1.0: Make sure your parentheses are present:
letters |> head()
# [1] "a" "b" "c" "d" "e" "f"
```

- %>% allowed the caller to use the "piped-in" values anywhere in the function call, simply by using a dot (.) as a place holder

```
c("dogs", "cats", "rats") %>% grepl("at", .)
# [1] FALSE  TRUE  TRUE
```

- no place holder is provided with the native pipe, a function must be defined and used

```
find_at = function(x) grepl("at", x)
c("dogs", "cats", "rats") |> find_at()
# [1] FALSE  TRUE  TRUE

c("dogs", "cats", "rats") |>
  {function(x) grepl("at", x)}()
# [1] FALSE  TRUE  TRUE
```


New R 4.1.0 features: anonymous functions

- it is common practice to define anonymous function when using higher-order functions: in the purrr package (map(), reduce(), keep() and in base-R: Map(), lapply()

```
# For each letter, find the name of each dataset
# in the {datasets} package that starts with that letter
purrr::map(
  letters[1:3],
  function(x) {
    ds = ls("package:datasets")
    ds[stringr::str_starts(tolower(ds), x)]
  }
)
#[[1]]
#[1] "ability.cov"   "airmiles"      "AirPassengers" "airquality"
...
#[[3]]
#[1] "cars" "ChickWeight" "chickwts"  "co2"  "CO2" "crimtab"
```

- or in base-R:

```
# In base R
Map(
  function(x) {
    pattern = paste0("^", x) # eg "^a" to match a leading 'a'
    grep(pattern, ls("package:datasets"), value = TRUE, ignore.case = TRUE)
  },
  letters[1:3]
)
```

New R 4.1.0 features: anonymous functions

- a new syntax has been introduced into R that may make these anonymous function declarations more succinct:

```
# R 4.1.0
Map(
  \(x) {
    pattern = paste0("^", x)
    grep(pattern, ls("package:datasets"), value = TRUE, ignore.case = TRUE)
  },
  letters[1:3]
)
```

- this syntax can help in accessing single values with the native pipe:

```
c("dogs", "cats", "rats") |> {\(x) grepl("at", x)}()
```

- the `\(x)` syntax can be used to write functions anywhere is needed

```
my_func <- \(x) mean(x, na.rm = TRUE)
my_func(c(1, 4, 7))
```

- and multiple arguments are allowed:

```
three_args <- \(a, b, c) a * b / c
three_args(1.5, 4, 3)
```


New R 4.1.0: concatenating factors

- in earlier R versions, combining factors with the syntax `c(factor1, factor2)` brings to a vector of integers

```
# with different levels:
factor("a")
# [1] a
# Levels: a
factor("b")
# [1] b
# Levels: b

c(factor("a"), factor("b"))
# [1] 1 1

# with identical levels:
c(factor("a"), factor("a"))
# [1] 1 1
```

- this happens because factors are stored internally as integers
- in R $\geq$ 4.1.0, combining two factors together generates a new factor with levels that are a combination of the levels in the original factors

```
fac1 = factor(c("a", "b", "d"))
fac2 = factor(c("b", "c"))
c(fac1, fac2)

# [1] a b d b c
# Levels: a b d c
```