Learning Bayesian Networks in R
an Example in Systems Biology

Marco Scutari

m.scutari@ucl.ac.uk
Genetics Institute
University College London

July 9, 2013
Bayesian Networks Essentials
Bayesian Networks

Bayesian networks [21, 27] are defined by:

- a network structure, a directed acyclic graph $\mathcal{G} = (\mathbf{V}, \mathbf{A})$, in which each node $v_i \in \mathbf{V}$ corresponds to a random variable $X_i$;
- a global probability distribution, $\mathbf{X}$, which can be factorised into smaller local probability distributions according to the arcs $a_{ij} \in \mathbf{A}$ present in the graph.

The main role of the network structure is to express the conditional independence relationships among the variables in the model through graphical separation, thus specifying the factorisation of the global distribution:

$$P(\mathbf{X}) = \prod_{i=1}^{p} P(X_i \mid \Pi X_i)$$

where $\Pi X_i = \{\text{parents of } X_i\}$.
A Simple Bayesian Network: Watson’s Lawn

**Bayesian Networks Essentials**

**Marco Scutari**

**University College London**
Graphical Separation

separation (undirected graphs)

\[ A \perp B \mid C \]
\[ P(A, B, C) = P(A \mid C) P(B \mid C) P(C) \]

d-separation (directed acyclic graphs)

\[ A \not\perp B \mid C \]
\[ P(A, B, C) = P(C \mid A, B) P(A) P(B) \]

\[
\begin{align*}
A \perp B \mid C & \quad \Rightarrow \quad P(A, B, C) = P(B \mid C) P(C \mid A) P(A) \\
& = P(A \mid C) P(B \mid C) P(C) 
\end{align*}
\]
Skeletons, Equivalence Classes and Markov Blankets

Some useful quantities in Bayesian network modelling:

- **The skeleton**: the undirected graph underlying a Bayesian network, i.e. the graph we get if we disregard arcs’ directions.

- **The equivalence class**: the graph (CPDAG) in which only arcs that are part of a v-structure (i.e. $A \rightarrow C \leftarrow B$) and/or might result in a v-structure or a cycle are directed. All valid combinations of the other arcs’ directions result in networks representing the same dependence structure $P$.

- **The Markov blanket** of a node $X_i$, the set of nodes that completely separates $X_i$ from the rest of the graph. Generally speaking, it is the set of nodes that includes all the knowledge needed to do inference on $X_i$, from estimation to hypothesis testing to prediction: the parents of $X_i$, the children of $X_i$, and those children’s other parents.
Skeletons, Equivalence Classes and Markov Blankets

- **DAG**
- **Skeleton**
- **CPDAG**
- **Markov blanket of X9**
Model selection and estimation are collectively known as **learning**, and are usually performed as a two-step process:

1. **structure learning**, learning the network structure from the data;
2. **parameter learning**, learning the local distributions implied by the structure learned in the previous step.

This workflow is implicitly Bayesian; given a data set $\mathcal{D}$ and if we denote the parameters of the global distribution as $\mathbf{X}$ with $\Theta$, we have

$$P(\mathcal{M} | \mathcal{D}) = P(\mathcal{G}, \Theta | \mathcal{D}) = P(\mathcal{G} | \mathcal{D}) \cdot P(\Theta | \mathcal{G}, \mathcal{D})$$

and structure learning is done in practice as

$$P(\mathcal{G} | \mathcal{D}) \propto P(\mathcal{G}) P(\mathcal{D} | \mathcal{G}) = P(\mathcal{G}) \int P(\mathcal{D} | \mathcal{G}, \Theta) P(\Theta | \mathcal{G}) d\Theta.$$
Inference on Bayesian networks usually consists of conditional probability (CPQ) or maximum a posteriori (MAP) queries.

Conditional probability queries are concerned with the distribution of a subset of variables $Q = \{X_{j_1}, \ldots, X_{j_l}\}$ given some evidence $E$ on another set $X_{i_1}, \ldots, X_{i_k}$ of variables in $X$:

$$CPQ(Q \mid E, \mathcal{M}) = P(Q \mid E, G, \Theta) = P(X_{j_1}, \ldots, X_{j_l} \mid E, G, \Theta).$$

Maximum a posteriori queries are concerned with finding the configuration $q^*$ of the variables in $Q$ that has the highest posterior probability:

$$MAP(Q \mid E, \mathcal{M}) = q^* = \arg\max_q P(Q = q \mid E, G, \Theta).$$
Causal Protein-Signalling Network from Sachs et al.
What follows reproduces (to the best of my ability, and Karen Sachs’ recollections about the implementation details that did not end up in the Methods section) the statistical analysis in the following paper [29] from my book [25]:

**Causal Protein-Signaling Networks Derived from Multiparameter Single-Cell Data**

Karen Sachs, *et al.*

*Science* **308**, 523 (2005);
DOI: 10.1126/science.1105809

That’s a landmark paper in applying Bayesian Networks because:

- it highlights the use of *observational vs interventional* data;
- results are *validated* using existing literature.
An Overview of the Data

The data consist in the **simultaneous** measurements of 11 phosphorylated proteins and phospholipids derived from thousands of **individual** primary immune system cells:

- 1800 data subject only to **general** stimulatory cues, so that the protein signalling paths are active;
- 600 data with **specific** stimulatory/inhibitory cues for each of the following 4 proteins: pmek, PIP2, pakts473, PKA;
- 1200 data with **specific** cues for PKA.

Overall, the data set contains 5400 observations with no missing value.
Network Validated from Literature

(11 nodes, 17 arcs)
The plot in the previous slide requires **bnlearn** [25] and **Rgraphviz** [14] (which is based on **graph** [13] and the Graphviz library).

```r
> library(bnlearn)
> library(Rgraphviz)
> spec =
+   paste("[PKC][PKA|PKC][praf|PKC:PKA][pme|PKC:PKA:pme]",
+   "[p44.42|pme|PKA][paks473|p44.42:PKA][P38|PKC:PKA]",
+   "[pjnk|PKC:PKA][p1cg][PIP3|p1cg][PIP2|p1cg:PIP3]")
> net = model2network(spec)
> class(net)
[1] "bn"
> graphviz.plot(net, shape = "ellipse")
```

The **spec** string specifies the structure of the Bayesian network in a format that recalls the decomposition into local probabilities; the order of the variables is irrelevant.
graphviz.plot() is **simpler** to use (but less flexible) than the functions in **Rgraphviz**; we can only choose the layout and do some limited formatting using `shape` and `highlight`.

```r
> h.nodes = c("praf", "pmek", "p44.42", "pakts473")
> high = list(nodes = h.nodes, arcs = arcs(subgraph(net, h.nodes)),
+ col = "darkred", fill = "orangered", lwd = 2, textCol = "white")
> gr = graphviz.plot(net, shape = "ellipse", highlight = high)
```

`graphviz.plot()` returns a `graphNEL` object, which can be **customised** with the functions in **graph** and **Rgraphviz**.

```r
> nodeRenderInfo(gr)$col[c("PKA", "PKC")]
>  = "darkgreen"
> nodeRenderInfo(gr)$fill[c("PKA", "PKC")]
>  = "limegreen"
> edgeRenderInfo(gr)$col[c("PKA~praf", "PKC~praf")]
>  = "darkgreen"
> edgeRenderInfo(gr)$lwd[c("PKA~praf", "PKC~praf")]
>  = 2
> renderGraph(gr)
```

To achieve a **complete control** on the layout of the network, we can export `gR` to the **igraph** [6] package or use **Rgraphviz** directly.
Plotting Networks, with Formatting

code: graphviz.plot(...) 

Marco Scutari
University College London
Creating a Network Structure in `bnlearn`

- With the network’s **string representation**, using `model2network()` and `modelstring()`.

  ```r
  > model2network(modelstring(net))
  ```

- Creating an empty network and adding arcs one at a time.

  ```r
  > e = empty.graph(nodes(net))
  > e = set.arc(e, from = "PKC", to = "PKA")
  ```

- Creating an empty network and adding all arcs in one batch.

  ```r
  > to.add = matrix(c("PKC", "PKA", "praf", "PKC"), ncol = 2, +                        byrow = TRUE, dimnames = list(NULL, c("from", "to")))
  > to.add
   from  to
  [1,] "PKC" "PKA"
  [2,] "praf" "PKC"
  > arcs(e) = to.add
  ```
Creating a Network Structure in \texttt{bnlearn}

- Creating an empty network and adding all arcs using an \textit{adjacency matrix}.

  \begin{verbatim}
  > n.nodes = length(nodes(e))
  > adj = matrix(0, nrow = n.nodes, ncol = n.nodes)
  > colnames(adj) = rownames(adj) = nodes(e)
  > adj['PKC', 'PKA'] = 1
  > adj['praf', 'PKC'] = 1
  > adj['praf', 'PKA'] = 1
  > amat(e) = adj
  > bnlearn:::fcat(modelstring(e))
  \end{verbatim}

- Creating one or more \textit{random} networks.

  \begin{verbatim}
  > random.graph(nodes(net), num = 5, method = "melancon")
  \end{verbatim}
Gaussian Bayesian Networks
As a first, exploratory analysis, we can try to learn a network from the data that were subject only to general stimolatory cues. Since these cues only ensure the pathways are active, but do not tamper with them in any way, such data are observational (as opposed to interventional).

```r
> sachs = read.table("sachs.data.txt", header = TRUE)
> head(sachs, n = 4)

<table>
<thead>
<tr>
<th>praf</th>
<th>pmek</th>
<th>plcg</th>
<th>PIP2</th>
<th>PIP3</th>
<th>p44.42</th>
<th>pakts473</th>
<th>PKA</th>
<th>PKC</th>
<th>P38</th>
<th>pjnk</th>
</tr>
</thead>
<tbody>
<tr>
<td>26.4</td>
<td>13.2</td>
<td>8.82</td>
<td>18.3</td>
<td>58.80</td>
<td>6.61</td>
<td>17.0</td>
<td>414</td>
<td>17.0</td>
<td>44.9</td>
<td>40.0</td>
</tr>
<tr>
<td>35.9</td>
<td>16.5</td>
<td>12.30</td>
<td>16.8</td>
<td>8.13</td>
<td>18.60</td>
<td>32.5</td>
<td>352</td>
<td>3.37</td>
<td>16.5</td>
<td>61.5</td>
</tr>
<tr>
<td>59.4</td>
<td>44.1</td>
<td>14.60</td>
<td>10.2</td>
<td>13.00</td>
<td>14.90</td>
<td>32.5</td>
<td>403</td>
<td>11.40</td>
<td>31.9</td>
<td>19.5</td>
</tr>
<tr>
<td>73.0</td>
<td>82.8</td>
<td>23.10</td>
<td>13.5</td>
<td>1.29</td>
<td>5.83</td>
<td>11.8</td>
<td>528</td>
<td>13.70</td>
<td>28.6</td>
<td>23.1</td>
</tr>
</tbody>
</table>
```

Most approaches in literature cannot handle interventional data, but work “out of the box” with observational ones.
When dealing with continuous data, we often assume they follow a multivariate normal distribution to fit a Gaussian Bayesian network [12, 26]. The local distribution of each node is a linear model,

\[ X_i = \mu + \Pi X_i \beta + \varepsilon \quad \text{with} \quad \varepsilon \sim N(0, \sigma_i). \]

which can be estimated with any frequentist or Bayesian approach. The same holds for the network structure:

- **Constraint-based algorithms** [24, 31, 32] use statistical tests to learn conditional independence relationships from the data.
- **In score-based algorithms** [23, 28], each candidate network is assigned a goodness-of-fit score, which we want to maximise.
- **Hybrid algorithms** [11, 33] use conditional independence tests are to restrict the search space for a subsequent score-based search.
> library(bnlearn)
> print(iamb(sachs))

Bayesian network learned via Constraint-based methods
model:
   [partially directed graph]

nodes: 11
arcs: 8
  undirected arcs: 6
  directed arcs: 2
average markov blanket size: 1.64
average neighbourhood size: 1.45
average branching factor: 0.18

learning algorithm: Incremental Association
conditional independence test: Pearson's Linear Correlation
alpha threshold: 0.05
tests used in the learning procedure: 259
optimized: TRUE
> print(hc(sachs))

Bayesian network learned via Score-based methods
model:
[praf] [PIP2] [p44.42] [PKC] [pmek|praf] [PIP3|PIP2] [pakts473|p44.42] [P38|PKC] [plcg|PIP3] [PKA|p44.42:pakts473] [pjnk|PKC:P38]
nodes: 11
arcs: 9
  undirected arcs: 0
  directed arcs: 9
average markov blanket size: 1.64
average neighbourhood size: 1.64
average branching factor: 0.82

learning algorithm: Hill-Climbing
score: Bayesian Information Criterion (Gaussian)
penalization coefficient: 3.37438
tests used in the learning procedure: 145
optimized: TRUE
> print(mmhc(sachs))

Bayesian network learned via Hybrid methods

model:

[praf][PIP2][p44.42][PKC][pmek|praf][PIP3|PIP2][pakts473|p44.42][P38|PKC][pjk|PKC][plcg|PIP3][PKA|p44.42:pakts473]

nodes: 11
arcs: 8

[...]
Since defaults are (often) not appropriate, we can tune each structure learning algorithm in \texttt{bnlearn} with several optional arguments.

- **Constraint-based algorithms**: we can pick the test \([8, 26]\), the alpha threshold and the number of permutations, e.g.:

  \[
  \texttt{> inter.iamb(sachs, test = "smc-cor", B = 100, alpha = 0.01)}
  \]

- **Score-based algorithms**: we can pick the score function \([17, 12]\) and its tuning parameters, the number of random restarts, the length of the tabu list, the maximum number of iterations, and more, e.g.:

  \[
  \texttt{> hc(sachs, score = "bge", iss = 3, restart = 5, perturb = 10)}
  \]
  \[
  \texttt{> tabu(sachs, tabu = 15, max.iter = 500)}
  \]

- **Hybrid algorithms**: both the above, e.g.:

  \[
  \texttt{> rsmax2(sachs, restrict = "si.hiton.pc", maximize = "tabu",}
  \]
  \[
  \texttt{+ test = "zf", alpha = 0.01, score = "bic-g"})
  \]

Other useful arguments: debug, whitelist, blacklist.
Parameter Learning: Fitting and Modifying

> net = hc(sachs)
> bn = bn.fit(net, sachs, method = "mle")
> bn$pmek

Parameters of node pmek (Gaussian distribution)
Conditional density: pmek | praf
Coefficients:
(Intercept)      praf
   -0.834129    0.520336
Standard deviation of the residuals: 16.72394

> bn$pmek = list(coef = c(0, 0.5), sd = 20)
> bn$pmek

Parameters of node pmek (Gaussian distribution)
Conditional density: pmek | praf
Coefficients:
(Intercept)      praf
      0.0      0.5
Standard deviation of the residuals: 20
When we learn a CPDAG representing an equivalence class (e.g. with constraint-based algorithms), such as

```r
> pdag = iamb(sachs)
```

we must set the directions of the undirected arcs before learning the parameters. We can do that automatically with `cextend` [7]

```r
> dag = cextend(pdag)
```

by imposing a topological ordering on the nodes,

```r
> dag = pdag2dag(pdag, ordering = node.ordering(net))
```

or by hand for each arc.

```r
> pdag = set.arc(pdag, from = "praf", to = "pme",
+                  check.cycles = FALSE)
```
Parameter Learning: Other Methods

We can use a list containing `coef`, `sd`, `fitted` and `resid` to set each node parameters’ from other models, either replacing parts of a Bayesian network returned by `bn.fit()` or creating a new one with `custom.fit()`.

```r
> dPKA = list(coef = c("(Intercept)" = 1, "PKC" = 1), sd = 2))
> dPKC = list(coef = c("(Intercept)" = 1), sd = 2))
> bn = custom.fit(net, dist = list(PKA = dPKA, PKC = dPKC, ...))
```

There are shortcuts to do that directly for the `penalized` package [15],

```r
> library(penalized)
> bn$pmek = penalized(pmek, penalized = ~ praf, data = sachs,
+                   lambda2 = 0.1, trace = FALSE)
```

and for `lm()` and `glm()`:

```r
> bn.pmek = lm(pmek ~ praf, data = sachs, na.action = "na.exclude",
+               weight = runif(nrow(sachs)))
```
What Kind of Network Structures Did We Learn?

They look nothing like the one validated from literature...
Gaussian Bayesian Networks

Parametric Assumptions: Variables Are Not Normal

They are not even symmetric...
Parametric Assumptions: Dependencies Are Not Linear

The regression line is nearly flat...
Discrete Bayesian Networks
Since we cannot use Gaussian Bayesian networks on the raw data, we must transform them and possibly change our parametric assumptions. For example, we can:

- **transform each variable** using a Box-Cox transform [35] to make it normal (or at least symmetric);
- **discretise each variable** [20, 34] using quantiles or fixed-length intervals;
- **jointly discretise all the variables** [16] into a small number of intervals by iteratively collapsing the intervals defined by their quantiles.

The last choice is the only one that gets rid of both normality and linearity assumptions while trying to preserve the dependence structure of the data.
Hartemink’s method [16] is designed to preserve pairwise dependencies as much as possible, unlike marginal discretisation methods.

```r
> library(bnlearn)
> dsachs = discretize(sachs, method = "hartemink",
+ breaks = 3, ibreaks = 60, idisc = "quantile")
```

Data are first marginalised in 60 intervals, which are subsequently collapsed while reducing the mutual information between the variables as little as possible. The process stops when each variable has 3 levels (i.e. low, average and high expression).
Each variable in the dsachs data frame is now a factor with three levels (low, average and high concentration). The local distribution of each node is a set of conditional distributions, one for each configuration of the levels of the parents.

, , PKC = (1,9.73]

<table>
<thead>
<tr>
<th>PKA</th>
<th>praf</th>
<th>(1.95,547]</th>
<th>(547,777]</th>
<th>(777,4.49e+03]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(1.61,39.5]</td>
<td>0.3383085</td>
<td>0.2040816</td>
<td>0.2352941</td>
</tr>
<tr>
<td></td>
<td>(39.5,62.6]</td>
<td>0.2885572</td>
<td>0.4897959</td>
<td>0.3921569</td>
</tr>
<tr>
<td></td>
<td>(62.6,552]</td>
<td>0.3731343</td>
<td>0.3061224</td>
<td>0.3725490</td>
</tr>
</tbody>
</table>

, , PKC = (9.73,20.2]

<table>
<thead>
<tr>
<th>PKA</th>
<th>praf</th>
<th>(1.95,547]</th>
<th>(547,777]</th>
<th>(777,4.49e+03]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(1.61,39.5]</td>
<td>0.3302326</td>
<td>0.3095238</td>
<td>0.3088235</td>
</tr>
<tr>
<td></td>
<td>(39.5,62.6]</td>
<td>0.3023256</td>
<td>0.2857143</td>
<td>0.3823529</td>
</tr>
<tr>
<td></td>
<td>(62.6,552]</td>
<td>0.3674419</td>
<td>0.4047619</td>
<td>0.3088235</td>
</tr>
</tbody>
</table>
**Posterior Maximisation using **deal**

**deal** implements learning using a Bayesian approach that supports discrete and mixed data assuming a conditional Gaussian distribution [2]. Structure learning is done with a hill-climbing search maximising the posterior density of the network (as in \texttt{hc(..., score = "bde")} in \texttt{bnlearn}).

```r
> library(deal)
> deal.net = network(dsachs)
> prior = jointprior(deal.net, N = 5)
> deal.net = learn(deal.net, dsachs, prior)$nw
> deal.best = autosearch(deal.net, dsachs, prior)
> bnlearn:::fcat(deal::modelstring(deal.best$nw))

[praf|pmek][pmek][plcg|PIP3][PIP2|plcg:PIP3][PIP3]
[p44.42][pakts473|p44.42][PKA|p44.42:pakts473][PKC|P38]
[P38][pjk|PKC:P38]
```
catnet [1] learns the network structure in two steps. First, it learns the node ordering from the data using simulated annealing [3],

```r
> library(catnet)
> netlist1 = cnSearchSA(dsachs)
```

unless provided by the user.

```r
> netlist2 = cnSearchOrder(dsachs, maxParentSet = 5,
+   nodeOrder = sample(names(dsachs)))
```

Then it performs an exhaustive search among the networks with the given node ordering and returns the maximum likelihood estimate.

```r
> catnet.best = cnFindBIC(netlist1, nrow(dsachs))
> catnet.best
```

A catNetwork object with 11 nodes, 1 parents, 3 categories, Likelihood = -9.904293 , Complexity = 50 .  

Marco Scutari University College London
**PC Algorithm using pcalg**

*pcalg* [19] implements the PC algorithm [31], and it is specifically designed to learn causal effects from both discrete and continuous data. *pcalg* can also account for the effects of latent variables through a modified PC algorithm known as Fast Causal Inference (FCI) [31, 4].

```r
> library(pcalg)
> suffStat = list(dm = dsachs, nlev = sapply(dsachs, nlevels),
+ adaptDF = FALSE)
> pcalg.net = pc(suffStat, indepTest = disCItest, p = ncol(dsachs),
+ alpha = 0.05)
> pcalg.net@graph

A graphNEL graph with undirected edges
Number of Nodes = 11
Number of Edges = 0
```

From the code above, we can also see how to implement custom conditional independence tests and pass them to `pc()` and `fci()` via the `indepTest` argument.
The categories in the discretised variables are ordered, so we are discarding information if we assume they come from a multinomial distribution. An appropriate test is the Jonckheere-Terpstra test \cite{8} which will be available soon\textsuperscript{TM} in the next release of \texttt{bnlearn}.

\begin{verbatim}
> library(bnlearn)
> print(iamb(dsachs, test = "jt"))
Bayesian network learned via Constraint-based methods
model:
    [partially directed graph]
nodes: 11
arcs: 8
[...]

learning algorithm: Incremental Association
conditional independence test: Jonckheere-Terpstra Test
alpha threshold: 0.05
tests used in the learning procedure: 223
\end{verbatim}
What Kind of Network Structures Did We Learn?

Again, they look nothing like the one validated from literature...
Moving Network Structures Between Packages

- From **bnlearn** to **deal** (and back).

  ```
  > mstring = bnlearn::modelstring(net)
  > dnet = deal::network(dsachs[, bnlearn::node.ordering(net)])
  > dnet = deal::as.network(bnlearn::modelstring(net), dnet)
  > net = bnlearn::model2network(deal::modelstring(dnet))
  ```

- From **bnlearn** to **pcalg** through **graph** (and back).

  ```
  > pnet = new("pcAlgo", graph = as.graphNEL(net))
  > net = bnlearn::as.bn(pnet@graph)
  ```

- From **bnlearn** to **catnet** (and back).

  ```
  > cnet = cnCatnetFromEdges(nodes = names(dsachs),
  +                    edges = edges(as.graphNEL(net)))
  > net = bnlearn::empty.graph(names(dsachs))
  > arcs(net, ignore.cycles = TRUE) = cnMatEdges(cnet)
  ```
All the packages we covered, with the exception of \texttt{bnlearn}, fit the parameters of the network when they learn its structure. As was the case for Gaussian Bayesian networks, in \texttt{bnlearn} we can compute the \textbf{maximum likelihood} estimates with

\begin{verbatim}
> fitted = bn.fit(net, dsachs, method = "mle")
\end{verbatim}

and, in addition, the \textbf{Bayesian posterior} estimates with

\begin{verbatim}
> fitted = bn.fit(net, dsachs, method = "bayes", iss = 5)
\end{verbatim}

while controlling the relative weight of the (flat) prior distribution with the \texttt{iss} argument. And we can also modify \texttt{fitted} or create it from scratch.

\begin{verbatim}
> new.cpt = matrix(c(0.1, 0.2, 0.3, 0.2, 0.5, 0.6, 0.7, 0.3, 0.1),
+                  dimnames = list(pmek = levels(dsachs$pmek),
+                                  pjnk = levels(dsachs$pjnk)),
+                  byrow = TRUE, ncol = 3)
> fitted$pmek = as.table(new.cpt)
\end{verbatim}
bnlearn can export discrete Bayesian networks to software packages such as Hugin, GeNIe or Netica by writing BIF, DSC and NET files.

> write.dsc(fitted, file = "bnlearn.dsachs.dsc")

Conversely, bnlearn can import discrete Bayesian networks created with those software packages by reading the BIF, DSC and NET files they create.

> fitted = read.dsc("bnlearn.dsachs.dsc")

As an example, that’s how a node looks like in a DSC file.

node pmek {
}

probability ( pmek | pjnk ) {
    (0) : 0.3622590, 0.2506887, 0.3870523;
    (1) : 0.3828909, 0.1811209, 0.4359882;
    (2) : 0.3763309, 0.2547093, 0.3689599;
}
Model Averaging and Interventional Data
The results of both structure and parameter learning are noisy in most real-world settings, due to limitations in the data and in our knowledge of the processes that control them. Since parameters are learned conditional on the results of structure learning, it's a good idea to use model averaging to obtain a stable network structure from the data. We can generate the networks to average in a few different ways:

- **Frequentist:** using nonparametric bootstrap and learning one network from each bootstrap sample (aka bootstrap aggregation or bagging) [9].

- **Full Bayesian:** using Markov Chain Monte Carlo sampling. from the posterior \( P(\mathcal{G} | \mathcal{D}) \) [10].

- **MAP Bayesian:** learning a set of network structures with high posterior probability from the original data.
Frequentist Model Averaging: Bootstrap Aggregation

```r
> library(bnlearn)
> boot = boot.strength(data = dsachs, R = 200, algorithm = "hc",
+ algorithm.args = list(score = "bde", iss = 10))
> boot[(boot$strength > 0.85) & (boot$direction >= 0.5), ]

from   to     strength direction
1   praf   pmeke  1.000  0.5675000
23  plcg   PIP2   0.990  0.5959596
24  plcg   PIP3   1.000  0.9900000
34  PIP2   PIP3   1.000  0.9950000
56  p44.42 pakts473 1.000  0.6175000
57  p44.42 PKA    0.995  1.0000000
67  pakts473 PKA    1.000  1.0000000
89  PKC    P38    1.000  0.5325000
90  PKC    pjnk   1.000  0.9850000
100 P38    pjnk   0.965  1.0000000

> avg.boot = averaged.network(boot, threshold = 0.85)
```
We can use the threshold we learn from the data [30] instead of specifying it with `threshold`, and investigate it with `plot(boot)`.
> nodes = names(dsachs)
> start = random.graph(nodes = nodes, method = "ic-dag", num = 200)
> netlist = lapply(start, function(net) {
+   hc(dsachs, score = "bde", iss = 10, start = net)
+ })
> rnd = custom.strength(netlist, nodes = nodes)
> head(rnd[(rnd$strength > 0.85) & (rnd$direction >= 0.5), ], n = 9)

<table>
<thead>
<tr>
<th>from</th>
<th>to</th>
<th>strength</th>
<th>direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>praf</td>
<td>pmek</td>
<td>1</td>
<td>0.5000</td>
</tr>
<tr>
<td>pmek</td>
<td>praf</td>
<td>1</td>
<td>0.5000</td>
</tr>
<tr>
<td>plcg</td>
<td>PIP2</td>
<td>1</td>
<td>0.5000</td>
</tr>
<tr>
<td>plcg</td>
<td>PIP3</td>
<td>1</td>
<td>1.0000</td>
</tr>
<tr>
<td>PIP2</td>
<td>plcg</td>
<td>1</td>
<td>0.5000</td>
</tr>
<tr>
<td>PIP2</td>
<td>PIP3</td>
<td>1</td>
<td>1.0000</td>
</tr>
<tr>
<td>pakts473</td>
<td>p44.42</td>
<td>1</td>
<td>0.7975</td>
</tr>
<tr>
<td>PKA</td>
<td>p44.42</td>
<td>1</td>
<td>0.8875</td>
</tr>
<tr>
<td>PKA</td>
<td>pakts473</td>
<td>1</td>
<td>0.5900</td>
</tr>
</tbody>
</table>

> avg.start = averaged.network(rnd, threshold = 0.85)
Structure learning algorithms implemented in other packages can be used for model averaging with `custom.strength()`; the only requirement is that `netlist` must be a list of `bn` objects or a list of arc sets stored in 2-columns matrices (like the ones returned by the `arcs()` function).

```r
> library(catnet)
> netlist = vector(200, mode = "list")
> ndata = nrow(dsachs)
> netlist = lapply(netlist, function(net) {
+   boot = dsachs[sample(ndata, replace = TRUE), ]
+   nets = cnSearchOrder(boot)
+   best = cnFindBIC(nets, ndata)
+   cnMatEdges(best)
+ })
> sa = custom.strength(netlist, nodes = nodes)
> avg.catnet = averaged.network(sa, threshold = 0.85)
```
The structure is stable overall, but the networks are still quite different from the validated network...
In most data sets, all observations are collected under the same general conditions and can be modelled with a single Bayesian network, because they follow the same probability distribution.

However, this is not the case when samples from different experiments are analysed together with a single, encompassing model. In addition to the data set we have analysed so far, which is subject only to a general stimulus meant to activate the desired paths, we have 9 other data sets subject to different targeted stimulatory cues and inhibitory interventions.

```r
> isachs = read.table("sachs.interventional.txt", header = TRUE, 
+                     colClasses = "factor")
```

Such data are often called interventional, because the values of specific variables in the model are set by an external intervention of the investigator.
Modelling an Intervention on $p_{mek}$

- **Model without interventions**
  - PKC
    - PKA
      - P38
      - praf
      - pjmk
      - $p_{mek}$
      - $p_{44.42}$
      - pakts473
  - $p_{44.42}$

- **Model with interventions**
  - PKC
    - $plcg$
      - PIP3
    - PKA
      - P38
      - praf
      - pjmk
      - $p_{mek}$
      - $p_{44.42}$
  - pakts473
One intuitive way to model these data sets with a single, encompassing Bayesian network is to include the intervention INT in the network and to make all variables depend on it with a whitelist.

```r
> wh = matrix(c(rep("INT", 11), names(isachs)[1:11]), ncol = 2)
> bn.wh = tabu(isachs, whitelist = wh, score = "bde",
              + iss = 10, tabu = 50)
```

We can also let the structure learning algorithm decide which arcs connecting INT to the other nodes should be included in the network, and blacklist all the arcs towards INT.

```r
> tiers = list("INT", names(isachs)[1:11])
> bl = tiers2blacklist(nodes = tiers)
> bn.tiers = tabu(isachs, blacklist = bl,
                  + score = "bde", iss = 10, tabu = 50)
```

tiers2blacklist() builds a blacklist such that all arcs going from a node in a particular element of the nodes argument to a node in one of the previous elements are blacklisted.
Modelling Intervention with an Extra Node

All nodes depend on INT

Relevant nodes depend on INT

Marco Scutari University College London
A better solution is to remove INT from the data,

```r
> INT = sapply(1:11, function(x) {
+     which(isachs$INT == x) })
> isachs = isachs[, 1:11]
> nodes = names(isachs)
> names(INT) = nodes

and incorporate it into structure learning using a modified posterior probability score (mbde) that takes its effect into account [5].

```r
> start = random.graph(nodes = nodes,
+    method = "melancon", num = 200, burn.in = 10^5,
+    every = 100)
> netlist = lapply(start, function(net) {
+    tabu(isachs, score = "mbde", exp = INT,
+        iss = 1, start = net, tabu = 50) })
> arcs = custom.strength(netlist, nodes = nodes, cpdag = FALSE)
> bn.mbde = averaged.network(arcs, threshold = 0.85)
```
Modelling Intervention with an Extra Node

bn.mbde

Validated network
Comparing Network Structures

When we compare two Bayesian networks, it is important to compare their equivalence classes through the respective CPDAGs instead of the networks themselves.

```r
> learned.spec = paste("[plcg][PKC][praf|PKC][PIP3|plcg:PKC]",
+ "[PKA|PKC][pme|praf:PKA:PKC][PIP2|plcg:PIP3][p44.42|PKA:pmek]",
+ "[pakts473|praf:p44.42:PKA][pjnk|pme:PKA:PKC][P38|PKA:PKC:pjnk]"
> true.spec = paste("[PKC][PKA|PKC][praf|PKC:PKA][pme|PKC:PKA:praf]",
+ "[p44.42|pme:PKA][pakts473|p44.42:PKA][P38|PKC:PKA]",
+ "[pjnk|PKC:PKA][plcg][PIP3|plcg][PIP2|plcg:PIP3]"
> true = model2network(true.spec)
> learned = model2network(learned.spec)
> unlist(compare(true, learned))

<table>
<thead>
<tr>
<th>tp</th>
<th>fp</th>
<th>fn</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

> unlist(compare(cpdag(true), cpdag(learned)))

<table>
<thead>
<tr>
<th>tp</th>
<th>fp</th>
<th>fn</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>
Inference
In their paper, Sachs et al. used the validated network to substantiate two claims:

1. a direct perturbation of \( p44.42 \) should influence \( pakts473 \);
2. a direct perturbation of \( p44.42 \) should not influence PKA.

The probability distributions of \( p44.42 \), \( pakts473 \) and PKA were then compared with the results of two ad-hoc experiments to confirm the validity and the direction of the inferred causal influences.

```r
> for (i in names(isachs))
+   levels(isachs[, i]) = c("LOW", "AVG", "HIGH")
> fitted = bn.fit(true, isachs, method = "bayes")
```

For convenience, we rename the levels of each variable to LOW, AVERAGE and HIGH.
gRain [18] implements exact inference for discrete Bayesian networks via junction tree belief propagation [21]. We can export a network fitted with bnlearn,

> library(gRain)
> jtree = compile(as.grain(fitted))

set the evidence (i.e. the event we condition on),

> jprop = setFinding(jtree, nodes = "p44.42", states = "LOW")

and compare conditional and unconditional probabilities.

> querygrain(jtree, nodes = "pakts473")$pakts473

<table>
<thead>
<tr>
<th>pakts473</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOW</td>
</tr>
<tr>
<td>0.60893407</td>
</tr>
</tbody>
</table>

> querygrain(jprop, nodes = "pakts473")$pakts473

<table>
<thead>
<tr>
<th>pakts473</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOW</td>
</tr>
<tr>
<td>0.665161776</td>
</tr>
</tbody>
</table>
Graphical Comparison of Probability Distributions

Causal and non-causal use of Bayesian networks are different...
bnlearn implements approximate inference via rejection sampling (called logic sampling in this setting), and soon™ via importance sampling (likelihood weighting) [22]. cpdist generates random observations from fitted for the nodes nodes conditional on the evidence evidence.

```r
> particles = cpdist(fitted, nodes = "pakts473",
+                    evidence = (p44.42 == "LOW"))
> prop.table(table(particles))

particles
          LOW     AVG    HIGH
0.665686790 0.332884451 0.001428759
```

On the other hand, cpquery returns the probability of a specific event.

```r
> cpquery(fitted, event = (pakts473 == "AVG"),
+         evidence = (p44.42 == "LOW"))
[1] 0.3319946
```

Both can be configured to generate samples in parallel (using the snow or parallel packages) and/or in small batches to fit available memory.
Approximate Inference with \textbf{bnlearn}

Compared to exact inference in \textbf{gRain}, approximate inference in \textbf{bnlearn} often requires more memory and is much \textit{slower}. However, it is \textit{more flexible} and allows much more complicated queries.

\begin{verbatim}
> cpquery(fitted,
+   event = (pakts473 == "LOW") & (PKA != "HIGH"),
+   evidence = (p44.42 == "LOW") | (praf == "LOW"))
[1] 0.5593692

> cpdist(fitted, n = 6, nodes = nodes(fitted),
+   evidence = (p44.42 == "LOW") | (praf == "LOW") &
+   (pakts473 %in% c("LOW", "HIGH")))

     P38 p44.42 pakts473 PIP2 PIP3 pjnk PKA PKC plcg pmek praf
    1 AVG AVG LOW LOW AVG LOW HIGH LOW LOW LOW LOW
    2 LOW AVG LOW LOW AVG AVG AVG AVG LOW LOW LOW
    3 HIGH LOW LOW LOW LOW AVG LOW LOW LOW LOW HIGH
\end{verbatim}

We can also generate random observations from the \textit{unconditional distribution} with \texttt{cpdist(fitted, n = 6, TRUE, TRUE)} or \texttt{rbn(fitted, n = 6)} as a term of comparison.
Thanks for attending!
References
*catnet: Categorical Bayesian Network Inference*, 2012.  
R package version 1.13.4.

deal: A Package for Learning Bayesian Networks.  

Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm.  

Learning High-Dimensional Directed Acyclic Graphs with Latent and Selection Variables.  

Causal Discovery from a Mixture of Experimental and Observational Data.  
The igraph Software Package for Complex Network Research.

A Simple Algorithm to Construct a Consistent Extension of a Partially Oriented Graph.

*Introduction to Graphical Modelling*.

Data Analysis with Bayesian Networks: A Bootstrap Approach.


Rgraphviz: Provides plotting capabilities for R graph objects.
R package version 2.3.8.

R package version 0.9-41.

Principled Computational Methods for the Validation and Discovery of Genetic Regulatory Networks.

Learning Bayesian Networks: The Combination of Knowledge and Statistical Data.
[18] Søren Højsgaard.  
Graphical Independence Networks with the gRain Package for R.  

Causal Inference Using Graphical Models with the R Package pcalg.  

Error-Based and Entropy-Based Discretization of Continuous Features.  
In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD ’96)*, pages 114–119. AAAI Press, 1996.

*Probabilistic Graphical Models: Principles and Techniques*.  

*Bayesian Artificial Intelligence*.  


*Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.*

*Artificial Intelligence: A Modern Approach.*

Causal Protein-Signaling Networks Derived from Multiparameter Single-Cell Data.

On Identifying Significant Edges in Graphical Models of Molecular Networks.
*Artificial Intelligence in Medicine, 57(3):207–217, 2013.*
Special Issue containing the Proceedings of the Workshop “Probabilistic Problem Solving in Biomedicine” of the 13th Artificial Intelligence in Medicine (AIME) Conference, Bled (Slovenia), July 2, 2011.
Causation, Prediction, and Search. 

Algorithms for Large Scale Markov Blanket Discovery. 

The Max-Min Hill-Climbing Bayesian Network Structure Learning Algorithm. 

[34] W. N. Venables and B. D. Ripley. 
Modern Applied Statistics with S. 

Model-Based Clustering and Data Transformations for Gene Expression Data. 