abn: an R package for modelling multivariate data using additive Bayesian networks

Gilles Kratzer, Marta Pittavino, Fraser Lewis, Reinhard Furrer

Abstract

This vignette describes the R package **abn** which provides functionality for identifying statistical dependencies in complex multivariate data using additive Bayesian network (ABN) models. This methodology is ideally suited for both univariate - one response variable, and multiple explanatory variables - and multivariate analysis, where in both cases all statistical dependencies between all variables in the data are sought. ABN models comprise of directed acyclic graphs (DAGs) where each node in the graph comprises a generalized linear model. Model search algorithms are used to identify those DAG structures most supported by the data. Currently implemented are models for data comprising of categorical, count and/or continuous variables. Further relevant information about **abn** can be found at: www.r-bayesian-networks.org.

Keywords: Graphical models, additive models, structure learning, exact order based search, parametric bootstrapping, JAGS, MCMC, parameter learning, heuristic search.

1. Introduction

Bayesian network (BN) modelling (Buntine 1991; Heckerman, Geiger, and Chickering 1995; Lauritzen 1996; Jensen 2001) is a form of graphical modeling which attempts to separate out indirect from direct association in complex multivariate data, a process typically referred to as structure discovery in Friedman and Koller (2003). In the last decades, BN modelling has been widely used in biomedical science/systems biology (Poon, Lewis, Pond, and Frost 2007a; Poon, Lewis, Frost, and Pond 2008; Poon, Lewis, Pond, and Frost 2007b; Needham, Bradford, Bulpitt, and Westhead 2007; Dojer, Gambin, Mizera, Wilczynski, and Tiuryn 2006; Jansen, Yu, Greenbaum, Kluger, Krogan, Chung, Emili, Snyder, Greenblatt, and Gerstein 2003; Djebbari and Quackenbush 2008; Hodges, Dai, Xiang, Woolf, Xi, and He 2010) to analyse multi-dimensional data. However, only in the last few years, ABN models have been applied to veterinary epidemiology field; thanks to their ability to generalize standard regression methodologies. Unlike other widely used multivariate approaches where dimensionality is reduced through exploiting linear combinations of random variables, such as in principal component analysis, graphical modeling does not involve any such dimension reduction.

BN is a generic and well established *data mining/machine learning* methodology, which has been demonstrated in other fields of study to be ideally suited to such analysis. They have been developed for analysing multinomial, multivariate Gaussian or conditionally Gaussian networks (a mix categorical and Gaussian variables), see Heckerman *et al.* (1995); Boettcher (2004); Geiger and Heckerman (1994). Additive Bayesian network (ABN) models are a special type of Bayesian network (BN) models, where each node in the graph comprises a generalized linear model. As the latter, they consist of statistical models which derive a directed acyclic graph (DAG) from empirical data, describing the

dependency structure between random variables. All types of BN models comprise of two reciprocally dependent parts: a qualitative (the structure) and a quantitative (the model parameters) part. The DAG is the graphical representation of the joint probability distribution of all random variables in the data. The model parameters are represented by a local probability distribution for all the variables in the network.

A number of libraries for fitting such BNs are available from CRAN. These types of BN have been constructed to ensure conjugacy, that is, enable posterior distributions for the model parameters and marginal likelihood to be calculated analytically. The purpose of **abn** is to provide a library of functions for more flexible BNs which can also not rely on conjugacy, which opens up an extremely rich modeling framework but at some considerable additional computational cost.

Currently **abn** includes functionality for fitting non-conjugate BN models which are multi-dimensional analogues of combinations of Binomial (logistic) and Gaussian regression. It includes also model with Poisson (log) distribution for count data and generalised linear models with random effects (with the previous distributions).

The objective in BN modeling structure discovery is to perform a model search on the data to identify an optimal model. Recall that BN models have a vast search space - super-exponential in the number of nodes - and it is generally impossible to determine a globally optimal model. How best to summarize a set of locally optimal networks with different structural features is an open question, and there are a number of widely used and intuitively reasonable possibilities. For example, one option is to conduct a series of heuristic searches and then simply select the best model found in Heckerman *et al.* (1995); alternatively, a single summary network can be constructed using results across many different searches (Hodges *et al.* 2010; Poon *et al.* 2007a). Otherwise, an exact search method, as the one presented in Koivisto and Sood (2004) which perform an exhaustive search, can be used. There are obvious pros and cons to either approaches and both are common in the literature and provide a good first exploration of the data.

For a general non-technical review of BN modeling applied in biology see Needham *et al.* (2007). While, a general introduction to BN modelling in veterinary epidemiology is provided by Lewis, Brulisauer, and Gunn (2011). Further applications of BN to veterinary studies were described by Ward and Lewis (2013); Wilson, Ribeiro, and Boinas (2013); Sanchez-Vazquez, Nielen, Edwards, Gunn, and Lewis (2012). Graphical modelling techniques used to analyse epidemiological data were used by Firestone, Lewis, Schemann, Ward, Toribio, and Dhand (2013); Firestone, Lewis, Schemann, Ward, Toribio, Taylor, and Dhand (2014); Lewis and McCormick (2012); Lewis (2012); Lewis and Ward (2013); Schemann, Lewis, Firestone, Ward, Toribio, Taylor, and Dhand (2013); Ludwig, Berthiaume, Boerlin, Gow, Léger, and Lewis (2013); McCormick, Sanchez-Vazquez, and Lewis (2013) resulting in dozens of publications, and references therein.

In this manual we first consider a series of examples illustrating how to fit different types of models and run different searches and summary analysis to a (synthetic) data set comprising of 250 observations from a joint distribution comprising of 17 categorical and 16 continuous variables which is included as part of the **abn** library. This data set is a single realization from a network of the same structure as that presented in Lewis *et al.* (2011), which is based on real data and sufficiently complex to provide a realistic example of data mining using Bayesian Network modeling. Then, a fully explained Case Study is provided, where all the important steps to conduct a data analysis using additive Bayesian networks are illustrated. Another detailed introduction and further relevant case studies about **abn** can be found at: www.r-bayesian-networks.org.

2. Pigs Case Study

We present data on disease occurrence in pigs provided by the industry body the 'British Pig Health Scheme' (BPHS). The main objective of BPHS is to improve the productivity of pig production in the UK, and reducing disease occurrence is a significant part of this process. The data we consider here comprise of a randomly chosen batch of 50 pigs from each of 500 randomly chosen pig producers in the UK. These are 'finishing pigs', animals about to enter the human food chain at an abattoir. Each animal is assessed for the presence of a range of different disease conditions by a specialist swine veterinarian. We consider here the following nine disease conditions: enzootic-pneumonia (EPcat); pleurisy (plbinary); milk spots (MS); hepatic scarring (HS); pericarditis (PC); peritonitis (PT); lung abscess (Abscess); tail damage (TAIL); and papular dermatitis (PDcat). The presence of any of these conditions results in an economic loss to the producer. Either directly due to the relevant infected part of the animal being removed from the food chain, or indirectly in cases such as enzooticpneumonia, which may potentially indicate poor herd health and efficiency losses on the farm. An additional loss, though not directly monetary, is the presence of tail damage which may be suggestive of welfare concerns, which may also be linked to sub-optimal production efficiency. Milk spots and hepatic scarring result from infestation with Ascaris suum which is particularly important as this is a zoonotic helminth parasite.

2.1. Deciding on a search method

As a very rough rule of thumb if there are less than 20 variables (and no random effects) then probably the most robust model search option is an exact search (as opposed to a heuristic) which will identify a globally best DAG. Followed then by parametric bootstrapping in order to assess whether the identified model contains excess structure (this is an adjustment for over-modelling). Although, the parametric bootstrapping might require access to a cluster computer to make this computationally feasible. This is arguably one of the most comprehensive and reliable statistical modelling approaches for identifying an empirically justified best guess (DAG) at 'nature's true model', the unknown mechanisms and processes which generated the study data.

Order Based Searches

It is generally not feasible to iterate over all possible DAG structures when dealing with more than a handful of variables, hence the reliance on heuristic searches. It is also extremely difficult to construct efficient Monte Carlo Markov chain samplers across BN structures. A solution to this was proposed in Friedman and Koller (2003) where rather than sample across DAGs, it was proposed to sample across node orderings. A node order is simply the set of integers 1 through n, where n is the number of variables in the data. A DAG is consistent with an ordering if for each node in the order its parents come before it. For example a DAG with only an arc from $1 \rightarrow 2$ is consistent with ordering $1, 2, 3, 4$ as the parent 1 comes before 2, but a DAG with an arc from $2 \rightarrow 1$ is not consistent with this ordering. In effect, each order is a collection of DAGs, and note that each DAG may be consistent with multiple orders, i.e. the empty DAG is consistent with every possible ordering. This introduces bias, in that averaging across orders need not give the same results as averaging across DAGs, if the latter were possible. This is relevant when estimating posterior probabilities of individual structural features, and is baised towards more parsimonious features as they are consistent with more orders. Note that this bias does not apply to maximising across orders, as in finding most probable structures (see later). The big advantage of searching across orders is that there are $n!$ different orders compared to a reasonably tight upper bound of $2^{\binom{n}{2}}$ for different DAGs.

There are (at least) two approaches for searching across orders. The first is to construct a Markov chain which samples from the posterior distribution of all orders, and is the approach presented in Friedman and Koller (2003). Alternatively, in Koivisto and Sood (2004) an exact method is proposed which rather than sample across orders, performs an exhaustive search. This has the advantage that it can also be used to find the globally optimal DAG of the data - the most probable structure - as well as posterior probabilities for structural features, such as individual arcs. The drawback is that this exact approach is only feasible with smaller number of variables e.g. up to 12 or 13 when dealing with additive models. For the code provided in **abn** this exact approach is readily feasible up to 20 variables using typical desktop computing, and potentially up to 25 variable with access to a shared memory cluster computer.

Most Probable Structure

Using the exact order based method due to Koivisto and Sood (2004) it is also possible to identify the DAG with globally best network score. Identification of a most probable structure is split into two parts. Firstly we calculate a cache of individual node scores, for example using `buildscorecache`. Next, an exhaustive order based structural search is carried out using the function `mostprobable` which relies on the information in the node cache.

As in the heuristic searches it is possible to ban or retain certain arcs, for example when splitting multinomial variables. This is done in the node cache computation step. There are two different structural priors implemented in this search, the first in the uniform prior where all structures (all parent combinations) are equally likely. This is the default `prior.choice=1` in `mostprobable` and the other functions. Also implemented is the prior used in Koivisto and Sood (2004) where all parent combinations of equal cardinality are equally weighted, this is `prior.choice=2`. The latter does give the same prior weight to a parent combination with no parents and a parent combination comprising off all possible parents (since there is only one choice of each, $n-1$ choose 0 and n-1 choose $n-1$). This may not be desirable but is included as a prior for completness. Note that the order based search is exact in the sense that it will identify a DAG who score is equal to the best possible score if it was possible to exhaustive search across all DAGs. For example, if using `prior.choice=1` then the network found should have a score greater than or equal to that found using the previously described heuristic searches. The structure found need not be unique in that others may exist with the same globally optimal score, the order based search is only guaranteed to find one such structure.

To calculate the most probable structure we again use `buildscorecache()` to calculate a cache of individual node scores. Next, the function `mostprobable()` does the actual exhaustive order based search, and works for both conjugate and additive models since as with calculating the posterior probabilities this step only involves structural searching and is not concerned with the precise parameterisation of each BN model.

2.2. Preparing the data

There are two main things which need to be checked in the data before it can be used with any of the abn model fitting functions.

- All records with missing variables must be either removed or else the values completed. There are a range of libraries available from CRAN for completing missing values using approaches such as multiple imputation. Ideally, marginalising over the missing values is preferable (as

opposed completing them as this then results in essentially dealing with models of models), but this is far from trivial here and not yet (and may never be) implemented in abn. To remove all records with one or more missing values then code similar to the following probably suffices:

```
library( abn)     # Load the library
mydat <- pigs.vienna[,-11]  # Get data, drop batch variable
mydat <- mydat[ complete.cases(mydat),]
```

N.b. this is not actually needed with pigs.vienna.

- All variables which are to be treated as binary must be coerced to factors. To coerce an existing variable into a factor then:

```
mydat[,1] <- as.factor(mydat[,1])
```

coerces the first variable in data.frame pigs.vienna. The levels (labels of the factor) can be anything provided there are only two and a “success" here is take to be the second level. For example, the second value in the vector returned by:

```
levels( mydat[,1])
```

To include additional variables in the modeling, for example interaction terms or polynomials, then these must be created manually and included into the data.frame just like any other variable.

2.3. Initial searches for a optimal model

Below is some R code which will perform an exact search. We want to find the DAG with the best goodness of fit (network score - log marginal likelihood) and ideally we would search for this without any apriori complexity limit (max number of parents). However, this may be both not computationally feasible and also highly inefficient. For example, with 25000 observation is it really realistic to consider models with up to 9 covariates per node.

One approach is to start off with an apriori limit of one parent per node, find the best DAG, and then repeat an identical search process (again using functions buildscorecache and mostprobably) with a parent limit of 2. And so on, stopping when the maximal DAG found no longer changes when the complexity limit is relaxed (increased). The parent limits `max.par` are not inserted, in the code below, the `max.par` command can vary from one to an higher parent limits. These initial searches can be easily automatized, in the library subdirectory `system.file('bootstrapping_example', package='abn')` is possible to find a script file `initsearch.bash` that performs the initial search explained before in an automated way, increasing progressively the number of parents, from 1 to 5 (for this specific example). The run time for the mostprobable function increases very considerably with the number of parents.

```
ban <- matrix( 0, nrow=dim(mydat)[2], ncol=dim(mydat)[2])
```

The ban and retain matrix must have the names set:

```
colnames( ban) <- rownames(ban) <- names(mydat)

retain <- matrix( 0, nrow=dim(mydat)[2], ncol=dim(mydat)[2])
colnames( retain) <- rownames(retain) <- names(mydat)
```

Setup the distribution list for each node:

```
mydists <- list( PC="binomial", PT="binomial", MS="binomial",
                 HS="binomial", TAIL="binomial",
                 Abscess="binomial", Pyaemia="binomial",
                 EPcat="binomial", PDcat="binomial",
                 plbinary="binomial")
```

Build a cache of all the local computations:

```
mycache <- buildscorecache( data.df=mydat,
           data.dists=mydists, dag.banned=ban,
           dag.retained=retain, max.parents=4)
```

Run the actual exact search:

```
mp.dag <- mostprobable( score.cache=mycache)
fabn <- fitabn( mp.dag, data.df=mydat, data.dists=mydists)

datadir <- tempdir()
```

Save the results obtained:

```
save( mycache, mp.dag, fabn, file=
      paste0(datadir,"mp",max.par,".RData"))
```

2.4. Results from the initial search

Searches across parent limits 1 through 5 were run and we now examine the results. What we are looking for is simply the model with the best score (the largest, the least negative, mlik value), checking that this does not improve when more parents are permitted. This then says we have found a DAG with maximal goodness of fit. What we find (below) is that the goodness of fit does not improve when we increase the parent limit beyond 3.

The code necessary to analyze the results from the script file and to visualize the resulting DAG, in a linux environment, is the following:

```
load( "RData/Rout1.RData")
ml1 <- fabn$mlik

tographviz( dag.m=mp.dag, data.df=mydat, data.dists=mydists,
```

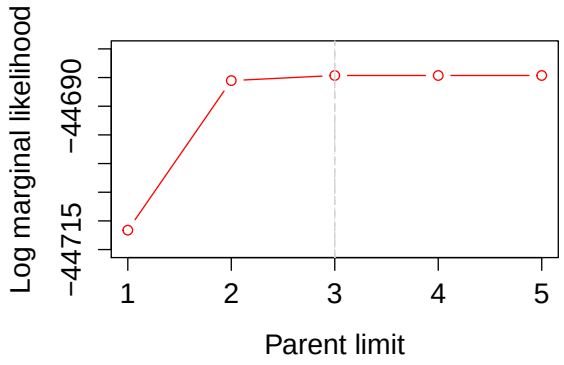

Figure 1: Comparison of goodness of fits for different parent limits

```
            outfile=paste0(datadir,"DAG_cycle.dot"))
# system( "dot -Tpdf -o DAG_cycle.pdf DAG_cycle.dot")
# system( "evince DAG_cycle.pdf&")
```

List of resulting marginal likelihood from the initial search:

```
mp.mlik <- c( -44711.62, -44685.53, -44684.64, -44684.64, -44684.64)
```

The actual DAG corresponding to the maximum marginal likelihood: `mp.mlik`$= -44684.64$ is in Fig. 2.

2.5. Adjustment for overfitting: parametric bootstrapping using MCMC

We have identified a DAG which has the best (maximum) possible goodness of fit according to the log marginal likelihood. This is the standard goodness of fit metric in Bayesian modelling (Mackay 1992), and includes an implicit penalty for model complexity. While it is sometimes not always apparent from the technical literature, the log marginal likelihood can easily (and sometimes vastly) overfit with smaller data sets. Of course the difficulty is identifying what constitutes 'small' here. In other words using the log marginal likelihood alone (or indeed any of the other usual metrics such as AIC or BIC) is likely to identify structural features, which, if the experiment/study was repeated many times, would likely only be recovered in a tiny faction of instances. Therefore, these features could not be considered robust. Overfitting is an ever present issue in model selection procedures, particular is common approaches such as stepwise regression searches, see Babyak (2004).

A well established approach for addressing overfitting is to use parametric bootstrapping (Friedman, Goldszmidt, and Wyner 1999). The basic idea is very simple. We take our chosen model and then simulate data sets from this, the same size as the original observed data, and see how often the different structural features are recovered. For example, is it reasonable for our data set of 434 observations to support a complexity of 29 arcs? Parametric bootstrapping is arguably one of the most defensible solutions for addressing overfitting, although it is likely the most computationally demanding, as for each simulated (bootstrap) data set we need to repeat the same exact model search as used with the original data. And we may need to repeat this analysis hundreds (or more) times to get robust results.

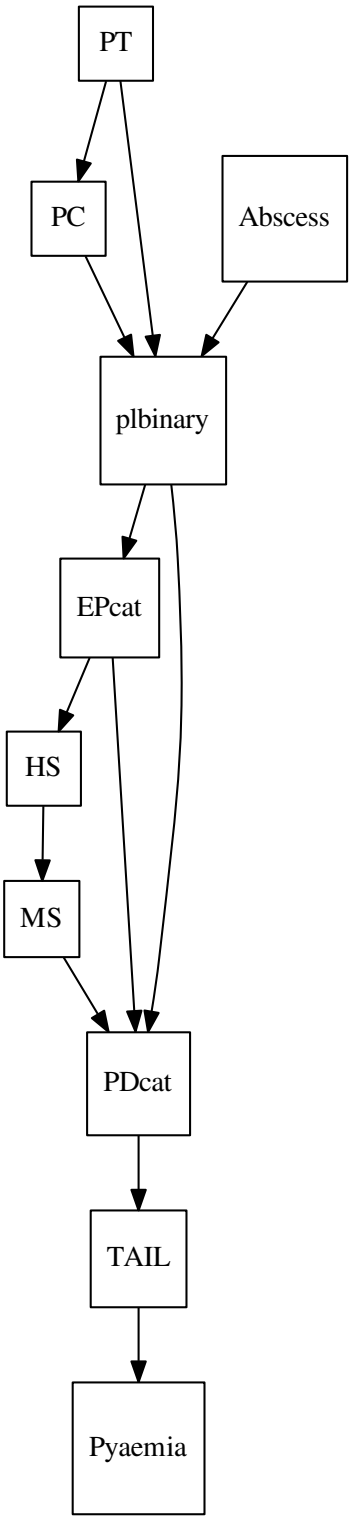

Figure 2: Globally optimal DAG, found with a maximum number of 3 or more parents.

Performing parametric bootstrapping is easy enough to code up if done in small manageable chunks. Here we provide a step-by-step guide along with necessary sample code.

2.6. Software needed

We have a DAG model and MCMC software such as JAGS and WinBUGS are designed for simulating from exactly such models. So all we need to do is implement our DAG model, e.g. in Fig. 2, in the appropriate JAGS (or WinBUGS) syntax (which are very similar). Here I am going to use JAGS, developed by Plummer (2003), in preference to WinBUGS or OpenBUGS for no other reason than that is what I am most familiar with, and JAGS is relatively straightforward to install (compile from source) on a cluster. Binary versions of JAGS are also available for most common platforms (Linux, Windows, OS X). To implement our DAG in JAGS we need write a model definition file (a BUG file) which contains the structure of the dependencies in the model. We also need to provide in here the probability distributions for each and every parameter in the model. Note that in Bayesian modelling the parameter estimates will not generally conform to any standard probability distribution (e.g. Gaussian) unless we are in the very special case of having conjugate priors. The marginal parameter distributions required can be estimated using the `fitabn` function and then fed into the model definition. We next demonstrate one way of doing this which is to use empirical distributions, in effect we provide JAGS with a discrete distribution over a fine grid which approximates whatever shape of density we need to sample from.

2.7. Generating marginal densities

The function `fitabn` has functionality to estimate the marginal posterior density for each parameter in the model. The parameters can be estimated one at a time by manually giving a grid (e.g. the x values where we want to evaluate f(x)) or else all together. In the latter case a very simple algorithm will try and work out where to estimate the density. This can work better sometimes and others, although it seems to work fine here for most variables. In order to use these distributions with JAGS we must evaluate the density over an equally spaced grid as otherwise the approach used in JAGS will not sample correctly. The basic command needed here is:

```
marg.f <- fitabn( mp.dag, data.df=mydat, data.dists=mydists,
                  compute.fixed=TRUE, n.grid=1000)
```

We should not simply assume that the marginals have been estimated accurately, and they should each be checked using some common sense. Generally speaking, estimating the goodness of fit (mlik) for a DAG comprising of GLM nodes is very reliable. This marginalises out all parameters in the model. Estimating marginal posterior densities for individual parameters, however, can run into trouble as this presupposes that the data contains sufficient information to accurately estimate the "shape" (density) for every individual parameter in the model. This is a stronger requirement than simply being able to estimate an overall goodness of fit metric. If a relatively large number of arcs have been chosen for a node with relatively few observations (i.e. “success” in a binary node) then this may not be possible, or at least the results are likely to be suspect. Exactly such issues - overfitting - are why we are performing the parametric bootstrapping in the first place but they can also pose some difficulties before getting to this stage.

It is essential to first visually check the marginal densities estimated from `fitabn`. Something like the following will create one pdf file where each page is a separate plot of a marginal posterior density.

```
pdf( "MargPlots_PigsData.pdf")
par( mfrow=c(5,5),mai=c(.2,.2,.1,0))
for( i in 1:length(marg.f$marginals)){
  cat( "processing marginals for node:",
  nom1 <- names( marg.f$marginals)[i],"\n")
  cur.node <- marg.f$marginals[i]
  # Get the marginal for current node, this is a matrix [x,f(x)]
  cur.node <- cur.node[[1]]
  # This is always [[1]] for models without random effects
  for( j in 1:length(cur.node)){
    cat( "processing parameter:",nom2<- names(cur.node)[j],"\n")
    cur.param <- cur.node[[j]]
    plot( cur.param,type="l",main=paste(nom1,":",nom2))
  }
  }
dev.off()
```

These plots (available in `system.file('bootstrapping_example',package='abn')`) suggests that the all the marginal estimates looks fine. Moreover, we can now perform an additional common sense check on their reliability. A probability density must integrate to unity (the area under the curve is equal to one). The densities here are estimated numerically and so we would not expect to get exactly one (n.b. no internal standarization is done so we can check this), but if the numerical estimation has worked reliably then we would expect this to be close (e.g. $0.99, 1.01$) to on.

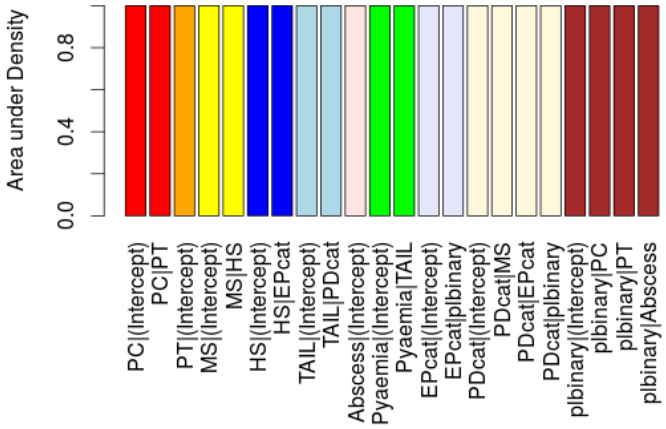

Figure 3: Approximated area under marginal densities

From Fig. 3 it is obvious that everything went fine in the marginal density estimation. To check the area we used the following code, which has as final output a file called `pigs_post_params.R` which contains all the information JAGS needs to sample from the marginal posterior distributions.

```
marnew <- marg.f$marginals[[1]]

for( i in 2: length(marg.f$marginals)){
```

```
  marnew <- c(marnew, marg.f$marginals[[i]])}
```

A straightforward question is: how reliable are the marginals? If the numerical routines work well then the area under the density function should be close to unity.

```
myarea <- rep( NA,length( marnew))
names( myarea) <- names( marnew)
for( i in 1:length(marnew)){
  tmp <- spline(marnew[[i]])
  # Spline just helps make the estimation smoother
  myarea[i] <- sum(diff(tmp$x)*tmp$y[-1])}
  # Just width x height of rectangles
cbind( myarea)
```

Now visualise `myarea` as a plot, the colours need to be adjusted:

```
mycols <- rep("green",length(marnew))
mycols[1:2] <- "red"; # PC
mycols[3] <- "orange"; # PT
mycols[4:5] <- "yellow"; # MS
mycols[6:7] <- "blue"; # HS
mycols[8:9] <- "lightblue"; # TAIL
mycols[10] <- "mistyrose"; # Abscess
mycols[11:12] <- "green"; # Pyaemia, lightcyan
mycols[13:14] <- "lavender"; # EPcat
mycols[15:18] <- "cornsilk"; # PDcat
mycols[19:22] <- "brown" ; # plbinary
png( "Area_PigsData.png", pointsize=10, width=720, height=640)
par( las=2, mar=c(8.1,4.1,4.1,2.1))
barplot( myarea, ylab="Area under Density", ylim=c(0,2), col=mycols)
dev.off()
```

Now we create the data in the right format ready for going to JAGS:

```
print( names(marnew))
# want to bind all the marginals the same nodes into a matrix
m <- marnew;
# PT -> PC, 1 parent, 2 params;
PC.p <- cbind( m[["PC|(Intercept)"]],  m[["PC|PT"]])
# PC --> NO PARENTS, 1 params;
PT.p <- cbind( m[["PT|(Intercept)"]])
# HS -> MS, 1 parent, 2 params;
MS.p <- cbind( m[["MS|(Intercept)"]], m[["MS|HS"]])
# EPcat -> HS, 1 parent, 2 params;
HS.p <- cbind( m[["HS|(Intercept)"]],  m[["HS|EPcat"]])
# PDcat -> TAIL, 1 parent, 2 params;
```

```
TAIL.p <- cbind( m[["TAIL|(Intercept)"]],  m[["TAIL|PDcat"]])
# Abscess --> NO PARENTS, 1 param;
Abscess.p <- cbind( m[["Abscess|(Intercept)"]])
# TAIL -> Pyaemia, 1 parent, 2 params;
Pyaemia.p <- cbind(m[["Pyaemia|(Intercept)"]],m[["Pyaemia|TAIL"]])
# plbinary -> EPcat, 1 parent, 2 params;
EPcat.p <- cbind( m[["EPcat|(Intercept)"]], m[["EPcat|plbinary"]])
# MS, EPcat, plbinary --> PDcat, 3 parents, 4 params;
PDcat.p <- cbind( m[["PDcat|(Intercept)"]], m[["PDcat|MS"]],
                  m[["PDcat|EPcat"]], m[["PDcat|plbinary"]])
# PC, PT, Abscess --> plbinary, 3 parents, 4 params;
plbinary.p <- cbind(m[["plbinary|(Intercept)"]],m[["plbinary|PC"]],
                    m[["plbinary|PT"]], m[["plbinary|Abscess"]])

#dump( c( "PC.p","PT.p","MS.p","HS.p","TAIL.p","Abscess.p",
#         "Pyaemia.p","EPcat.p","PDcat.p","plbinary.p"),
#          file=paste('Jags/',"pigs_post_params.R",sep=""))
```

2.8. Building BUG model

Once we have the posterior distributions the next step is to actually create the DAG in JAGS. This involves creating a BUG file, a file which contains a definition of our DAG (from Fig. 2) in terms which can be understood by JAGS. This can easily be done by hand, if rather tedious, and should be checked carefully for errors (which JAGS will prompt about in any case). The BUG file is here. This file is fairly heavily commented (it might look complicated but most of it is just copy and paste with minor edit) - the syntax is similar to R - and should be fairly self explanatory. Note that unlike a more usual use of WinBUGS or JAGS we have no data here, we are simply providing JAGS with a set of marginal probability distributions and how they are inter-dependent, and we want it to then generate realisations from the appropriate joint probability distribution. The next step is to tell JAGS to perform this simulation, i.e. generate a single bootstrap data set of size $n = 25000$ observations based on the assumption that the DAG is Fig. 2 is our true model. The BUG file can be found in `system.file('bootstrapping_example',package='abn')` in the file `pigs_model.bug`.

2.9. A single bootstrap analysis

To run JAGS we use four separate files: i) the BUG model definition file (model.bug); ii) a file `pigs_post_params.R` which contains the marginal distributions referred to in the BUG file; iii) a script which runs the MCMC simulation `jags_pigs_script.R`; and finally iv) a file which sets of random number seed (`inits.R` - the value in this must be changed to use different streams of random numbers). These four files are contained in this tarball. To run this example extract all the files into one directory and then at the command line type `'jags jags_pigs_script.R'`. In terms of the MCMC component, a burn-in of 100000 is used but as there are no data here this takes no time to run and is likely of little use and could be shorted (it is just included to allow JAGS any internal adaptations or diagnostics that it might need to do). The actual MCMC is then run for 250000 iterations with a thin of 10, which gives 25000 observations for each variable - the same size as the

original data. The number of MCMC steps and thin is a little arbitrary and this could be run for longer with a bigger thin, but for this data looking at autocorrelation plots for the Gaussian variables there appears no evidence of correlation at a thin of 10 and so this seems sufficient here.

The next step is to automate a single run, e.g. generate a single bootstrap sample and then perform an exact search on this, just as was done for the original data. This file performs a single such analysis in R - just drop this into the same directory as the four JAGS file above. For ease we also call the JAGS script from inside R which might require some messing about with the PATH variable on Windows (e.g. if you open up a cmd prompt then you should be able to type "jags" without needed to give a full path). The output from this is given below and "results" is a single matrix (DAG) which is saved in an R workspace called `boot1run.RData`.

Get the pigs data, drop batch variable

```
orig.data <- pigs.vienna[,-11]
```

Now create a single bootstrap sample:

```
system("jags jags_pigs_script.R")
```

Read in boot data and convert to data.frame in same format as the original data, e.g. coerce to factors, using the appropriate R package **coda**, described in Plummer, Best, Cowles, and Vines (2006):

```
library( coda)
boot.data <- read.coda("out1chain1.txt", "out1index.txt")

boot.data <- as.data.frame(boot.data)
for( j in 1:dim(orig.data)[2]){if(is.factor(orig.data[,j]))
        { boot.data[,j]<- as.factor(boot.data[,j])
          levels(boot.data[,j])<- levels(orig.data[,j])}}
```

Now we have the boot.data in identical format as the original:

```
ban <- matrix( 0, nrow=dim(orig.data)[2], ncol=dim(orig.data)[2])
colnames( ban) <- rownames(ban) <- names(orig.data)

retain <- matrix( 0, nrow=dim(orig.data)[2], ncol=dim(orig.data)[2])
colnames( retain) <- rownames(retain) <- names(orig.data)

mydists <- list( PC="binomial",  PT="binomial", MS="binomial",
                 HS="binomial",  TAIL="binomial",
                 Abscess="binomial", Pyaemia="binomial",
                 EPcat="binomial", PDcat="binomial",
                 plbinary="binomial")
```

Set the parent limits to 3, equal to the original data:

```
max.par <- 3
```

Build a cache on bootstrap data:

```
boot1.cache <- buildscorecache( data.df=boot.data,
               data.dists=mydists, max.parents=max.par,
               dag.banned=ban, dag.retained=retain)
```

Run `mostprobable` search on the bootstrap data:

```
boot1.mp <- mostprobable( score.cache=boot1.cache)

datadir <- tempdir()
save( boot1.mp,file=paste( datadir,"boot1run.RData",sep=''))
```

Once this works on your local machine it is then a case of trying to automate this in the most efficient way, for example for use on a cluster. The crucial thing here is that the random number seed used each time (in the file `inits.R`) must be changed for each bootstrap simulation otherwise an identical bootstrap data set will be produced!

2.10. Ways to summarise results from parametric bootstrapping

The globally optimal DAG, Fig. 2, has 12 arcs. It way be that some of these are due to over-modelling which means they will be recovered in relatively few bootstrap analyses. 10240 bootstrap analyses were conducted (on a cluster) and all the R files, MPI wrapper, and also the actual results (in R workspaces) can be found in the folder `system.file('bootstrapping_example', package='abn')`.

The first step is to explore the bootstrap results. Of some interest is how many of the arcs were recovered during each of the bootstrap "simulations". This is given in Fig. 4. We can see right away that not all the arcs in the original DAG (Fig. 2 - with 12 arcs) were recovered - even just once. This provides overwhelming evidence that the original exact search has indeed overfitted to the original data. Not at all surprising given the relatively small sample size. We must therefore trim off some of the complexity - arcs - from the original DAG in order to justify that our chosen DAG is a robust estimate of the underlying system which generated the observed data.

A parametric bootstrapping approach was suggested in Friedman *et al.* (1999) which uses simulation to assess whether a chosen model comprises more complexity than could reasonably be justified given the size of the observed data set. Using Markov chain Monte Carlo simulation via JAGS (open source software), 10240 independent (assumed by inspecting autocorrelations from the MCMC output) data sets of the same size as the original data were generated from our chosen model in i). For each of these bootstrap data sets an identical exact order-based search as in i) was conducted.

There are a number of different options in terms of trimming/pruning arcs. One common option which apes the use of majority consensus trees in phylogenetics trees are just special cases of DAGs, is to remove all arcs which were not recovered in at least a majority (50%) of the bootstrap results. Fig. 5 shows the frequency at which each arc was recovered, the maximum value possible being 10240 $(100\%$ support). Collating results across these 10240 searches we find that only 14% of the globally optimal DAGs found comprised 12 or more arcs. Approximately 68% of DAGs had 11 or more arcs -

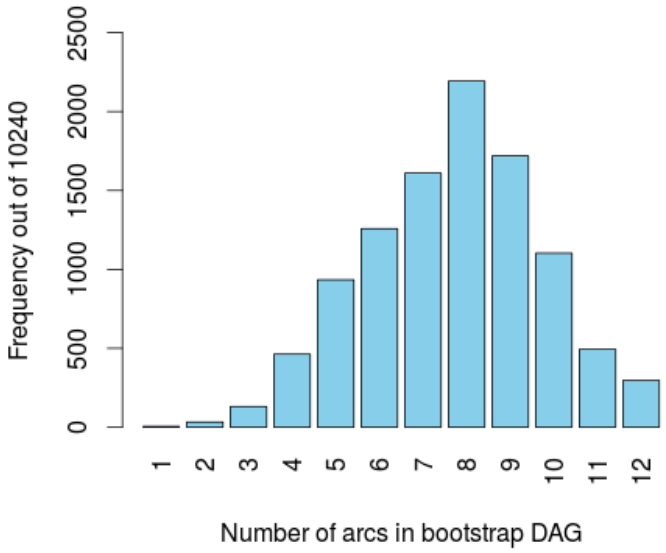

Figure 4: Number of arcs recovered in bootstrapping

therefore a robust model of the original data has no more than 11 arcs. Almost identical results were obtained using a random selection of 5012 searches suggesting that sufficient bootstrap samples had been performed. The usual cut-off for structural support of features (arcs) is 50% in BN modeling (Poon *et al.* 2007a,b, 2008; Lycett, Ward, Lewis, Poon, Pond, and Brown 2009; Lewis *et al.* 2011), and is analogous to the widespread use of majority consensus trees in phylogenetics. We therefore conclude that our chosen model in i) with 11 arcs is robust. This is perhaps not surprising given we have a large data set of $25K$ observations.

	PC	PT	MS	HS	TAIL	Abscess	Pyaemia	EPcat	PDcat	plbinary
PC	0	6607	0	0	0	0	0	0	0	0
PT	0	0	0	0	0	0	0	0	0	0
MS	0	0	0	5243	0	0	0	0	0	0
HS	0	0	0	0	0	0	0	4402	0	0
TAIL	0	0	0	0	0	0	0	0	6190	0
Abscess	0	0	0	0	0	0	0	0	0	0
Pyaemia	0	0	0	0	3257	0	0	0	0	0
EPcat	0	0	0	0	0	0	0	0	0	9686
PDcat	0	0	5096	0	0	0	0	3703	0	9009
plbinary	9249	6606	0	0	0	9799	0	0	0	0

Figure 5: Frequencies at which each arc in the original DAG was recovered during bootstrapping

Note that this 50% is not in any way comparable with the usual 95% points used in parameter estimation as these are entirely different concepts (an explanation for this can be found here).

Another option, other than choosing a different level of support (which is entirely up to the researcher), is to consider an undirected network. That is, include all arcs if there support - considering both directions - exceeds 50%. This is justifiable due to likelihood equivalence which means that - generally speaking - the data cannot discriminate between different arc directions (see here for an explanation) and therefore considering arcs recovered in only one direction may be overly conservative. Again, this decision is likely problem specific. For example, from a purely statistical perspective being con-

servative is generally a good thing, but from the scientists point of view this may then remove most of the more interesting results from the study. Obviously a balance is required.

In this data it turns out that removing all arcs with have less than 50% support gives an identical pruned network as if we were to consider both arc directions jointly. In generally this need not be the case. Fig. 6 shows out optimal DAG after removing these arcs. This is our optimal model of the data.

All the code for analysing the results from the bootstrap analyses can be found here:

```
mydata <- pigs.vienna[,-11]

N <- 10240;
# Write out manually, clearer than using rep()
mydag <- matrix(c(
# PC  PT  MS  HS TAIL Abscess Pyaemia EPcat PDcat plbinary
  0,  1,  0,  0,  0,  0,       0,     0,    0,     0,
  0,  0,  0,  0,  0,  0,       0,     0,    0,     0,
  0,  0,  0,  1,  0,  0,       0,     0,    0,     0,
  0,  0,  0,  0,  0,  0,       0,     1,    0,     0,
  0,  0,  0,  0,  0,  0,       0,     0,    1,     0,
  0,  0,  0,  0,  0,  0,       0,     0,    0,     0,
  0,  0,  0,  0,  1,  0,       0,     0,    0,     0,
  0,  0,  0,  0,  0,  0,       0,     0,    0,     1,
  0,  0,  1,  0,  0,  0,       0,     1,    0,     1,
  1,  1,  0,  0,  0,  1,       0,     0,    0,     0),
  byrow=TRUE,ncol=10)
colnames(mydag) <- rownames(mydag) <- names(mydata)
sum( mydag) # 12 arcs, as in original model, Figure 2

mydists <- list( PC="binomial", PT="binomial", MS="binomial",
                 HS="binomial", TAIL="binomial",
                 Abscess="binomial", Pyaemia="binomial",
                 EPcat="binomial", PDcat="binomial",
                 plbinary="binomial")

# Use fitabn to check mydag is correct (no typos mlik = -44684.64)
print( fitabn(mydag, data.df=mydata, data.dists=mydists)$mlik)
bestdag <- mydag
```

The next command reads all files with `mp[number].RData` and create a list of results:

```
boot.dags <- list()
these <- grep("mp10Kboot\\d+.RData", dir())
num <- 1
for( i in dir()[these]){# Load each file
  load(i) # Provides dags, a list
  tmp <- dags[which(unlist(lapply(dags,sum))>0)]
  # Get valid entries in dags but as a list
```

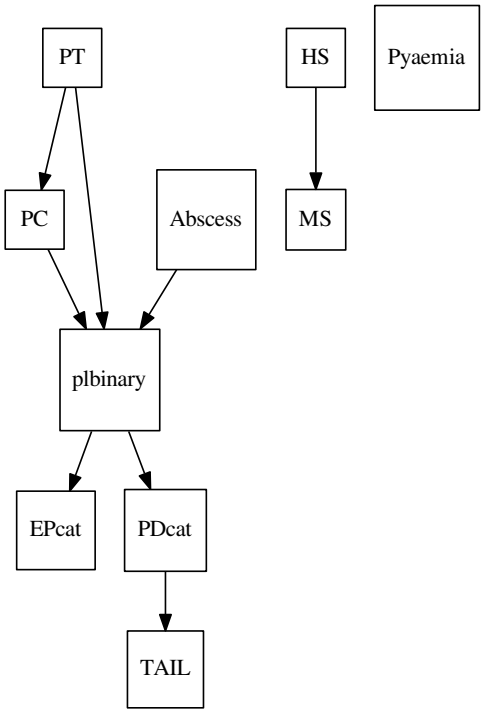

Figure 6: Optimal DAG after removal of arcs supported at less than 50% in bootstrapping. Contains 8 Arcs.

```
  for( j in 1:length(tmp)){
  # For each entry copy into boot.dags, and increment counter
    boot.dags[[num]]<- tmp[[j]]; num <- num+1 }
  rm( dags)
}
```

It is always good practice, have a look at mlik values for the bootstraps viz a viz the original:

```
if( FALSE){
  scores <- rep(0,length(boot.dags))
  for(i in 1:length(boot.dags)){
    scores[i] <- fitabn(dag.m=boot.dags[[i]],data.df=mydata,

  }
  scores.b <- scores[-which(scores< -N)]
  orig.score <- fitabn(dag.m=bestdag,data.df=mydata,

  plot(density(scores.b,from=min(scores.b),to=max(scores.b)))
  abline(v=orig.score,lwd=2,col="blue")
}
```

We now trim all arcs from the boot results which do not occur in the Master DAG - bestdag - since we know these are due to overfitting:

```
boot.dags.trim <- boot.dags
for( i in 1:length(boot.dags)){
   boot.dags.trim[[i]] <- boot.dags.trim[[i]]*bestdag  }

arc.freq <- lapply(boot.dags.trim,sum)
arc.freq <- table(unlist(arc.freq))

png("PigsFreqBootRes.png", pointsize=10, width=720, height=700)
par(las=1, mar=c(6.1,6.1,4.1,2.1))
barplot( arc.freq,ylab="",xlab="",col="skyblue",
         names.arg=names(arc.freq), ylim=c(0,2500))
par( las=1)
mtext( "Number of arcs in bootstrap DAG",1,line=3,cex=1.5)
par( las=3)
mtext( "Frequency out of 10240",2,line=4,cex=1.5)
dev.off()

total.dag <- matrix(rep(0,dim(bestdag)[2]^2),ncol=dim(bestdag)[2])
colnames(total.dag) <- rownames(total.dag)<- colnames(bestdag)
# Get the support for each arc, total.dag:
for( i in 1:length(boot.dags)){
  if(sum(boot.dags[[i]])>0){total.dag <- total.dag+boot.dags[[i]]}}
```

```
total.dag <- total.dag*bestdag # We only want arcs in the best DAG
total.dag
```

The frequencies at which each arc in the original DAG was recovered during bootstrapping.

We get the majority consensus (directed DAG):

```
f <- function(val,limit){ if(val<limit){return(0)}
                          else {return(1)}}
bestdag.trim <- apply( total.dag,c(1,2),FUN=f,limit=N/2)
```

We get the majority consensus (undirected DAG), but with arcs in the most supported direction:

```
bestdag.trim.nodir <- bestdag
bestdag.trim.nodir[,] <- 0 # Set zero
child <- NULL; parent <- NULL
for( i in 1:dim(total.dag)[1]){
  for( j in 1:dim(total.dag)[2]){
     if(i>j){ # Get most supported direction
              if(total.dag[i,j]>total.dag[j,i]){
                m.i <- i; m.j <- j;}
              else {m.i <- j; m.j <- i;}
              # Does arc quality - exceed threshold of support
              if(total.dag[i,j]+total.dag[j,i]>N/2){
              # We want this as more than 5000
              bestdag.trim.nodir[m.i,m.j] <- 1}}}}

tographviz( dag.m=bestdag.trim,data.df=mydata, data.dists=mydists,
           outfile=paste0(datadir,"postbootpigs.dot"))
# system( "dot -Tpdf -o postbootpigs.pdf postbootpigs.dot")
# system( "evince postbootpigs.pdf&")

save( bestdag.trim, file=paste0(datadir, "bestdagpigs_trim.RData"))
```

2.11. Estimating marginals from the final DAG

Once we have identified our optimal DAG then it is usual to want to examine the parameters in this model, e.g. in Fig. 7. These are our results, the effects of the various variables in our study. This process is very similar to when estimating the marginals for the bootstrapping but should now be easier since we should have removed any difficulties due to over-fitting. The posterior density plots for the final DAG can be found in the pdf called `Pigs_PostBootPlots.pdf`. These all look fine. An outlook on the parameters can be done based on the variable plbinary.

The R code for creating the marginals and quantiles can be found below:

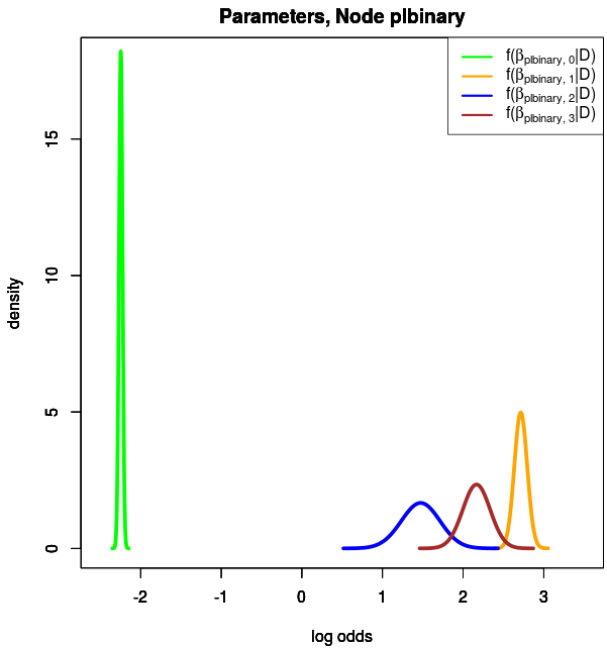

Figure 7: Marginal posterior density based in the node plbinary

```
mydata <- pigs.vienna[,-11]

mydists <- list( PC="binomial", PT="binomial", MS="binomial",
                 HS="binomial", TAIL="binomial",
                 Abscess="binomial", Pyaemia="binomial",
                 EPcat="binomial", PDcat="binomial",
                 plbinary="binomial")

marg.f <- fitabn(dag.m=bestdag.trim,data.df=mydata,
                 data.dists=mydists,compute.fixed=TRUE,
                 n.grid=1000)

pdf("Pigs_PostBootPlots.pdf")
for( i in 1:length(marg.f$marginals)){
  cat( "processing marginals for node:",
  nom1 <- names(marg.f$marginals)[i],"\n")
  cur.node <- marg.f$marginals[i]
  # Get marginal for current node, this is a matrix [x,f(x)]
  cur.node <- cur.node[[1]]
  # This is always [[1]] for models without random effects
   for( j in 1:length(cur.node)){
   cat("processing parameter:",
   nom2 <- names(cur.node)[j],"\n")
   cur.param <- cur.node[[j]]
   plot( cur.param,type="l",main=paste(nom1,":",nom2))}}
```

```
dev.off()

marnew <- marg.f$marginals[[1]]
for(i in 2: length(marg.f$marginals)){
  marnew <- c(marnew, marg.f$marginals[[i]])}

margs <- marnew;
mymat <- matrix(rep(NA,length(margs)*3),ncol=3)
rownames(mymat) <- names(margs)
colnames(mymat) <- c("2.5%","50%","97.5%")
ignore.me <- union(grep("\\(Int",names(margs)),
                   grep("prec",names(margs)))
```

These are not effect parameters, but background constants and precisions:

```
comment <- rep("",length(margs))
for(i in 1:length(margs)){
  tmp <- margs[[i]]
  tmp2 <- cumsum(tmp[,2])/sum(tmp[,2])
  mymat[i,] <- c(tmp[which(tmp2>0.025)[1]-1,1],
                 tmp[which(tmp2>0.5)[1],1],
                 tmp[which(tmp2>0.975)[1],1])
  myvec <- mymat[i,]
  if( !(i%in%ignore.me) &&  (myvec[1]<0 && myvec[3]>0)){
      comment[i] <- "not sig. at 5%"}
  # Truncate for printing
  mymat[i,] <- as.numeric(formatC(mymat[i,],digits=3,format="f"))}
cbind( mymat)
```

The code above produce the table of percentiles that can be found in Table 1.

All of the effect parameters, that is, ignoring the intercept terms (which is just a background constant) and the precision parameters - have 95% confidence (or credible) intervals which do not cross the origin. Note this is not guaranteed to happen this criteria was not part of the model selection process, and all the parameters in the model have been justified using mlik and bootstrapping. But having such marginal intervals can make presenting the results possibly easier to a skeptical audience.

2.12. Pigs Case Study Conclusion

Based on the results presented in the previous section, it possible to draw a conclusion of the study. From Table 1, we can notice that, a part from HS, which has a protective effect on MS, all others parameters have a risk effect on possible disease outcome. Our final optimal ABN model of this disease system suggests that the presence of Pyaemia is independent from other conditions. Moreover, the remaining diseases split into two separate connected components.

Some interesting biological questions, resulting from ABN model, can be related to the similarity of causal pathways between these two groups of diseases and to the common causes shared within each of these groups.

		Odds Ratio	
	2.5%	**50%**	**97.5%**
PC|(Intercept)	0.03	0.03	0.03
PC|PT	17.71	26.44	39.21
PT|(Intercept)	0.00	0.00	0.00
MS|(Intercept)	0.06	0.06	0.07
MS|HS	0.20	0.30	0.44
HS|(Intercept)	0.06	0.06	0.06
TAIL|(Intercept)	0.00	0.00	0.00
TAIL|PDcat	2.61	4.32	6.78
Abscess|(Intercept)	0.00	0.01	0.01
Pyaemia|(Intercept)	0.00	0.00	0.00
EPcat|(Intercept)	0.36	0.37	0.38
EPcat|plbinary	1.97	2.14	2.31
PDCat|(Intercept)	0.04	0.04	0.04
PDcat|plbinary	1.78	2.08	2.41
plbinary|(Intercept)	0.10	0.11	0.11
PC|(Intercept)	12.92	15.12	17.71
PC|(Intercept)	2.72	4.36	6.97
PC|(Intercept)	6.23	8.71	12.17

Table 1: Marginal posterior quantiles for each parameter. The red lines indicate point estimates bigger than 1, corresponding to risk factor. While the blue line refers to estimates smaller than 1, indicating protective factor.

The principal reasons because ABN models should be used are linked to their ability of generalization of the usual GLM to multiple dependent variables: fully multi-dimensional models. Specifically, results from ABN analyses can be used as a basis for developing new biological questions about factors potentially affecting disease's presence, and inform the design of future targeted studies.

3. An additional example

In the next subsections we illustrate how to fit an ABN model to different kinds of data, starting with the introdcution of the data we are going to deal with. The main purpose of BN structure discovery is to estimate the joint dependency structure of the random variables in the available data, and this is achieved by heuristically searching for optimal models and comparing their goodness of fit using Bayes factors. It is assumed that all structures are equally supported in the absence of any data - an uniformative prior on structures - and so comparing Bayes factors collapses to comparing the marginal likelihoods which is done on a log scale. The log marginal likelihood for a BN is typically referred to as the network score.

3.1. Simulated Data Example var33

Figure 1 shows the structure of the distribution which generated the data set `var33` included with **abn**. This diagram was created using the `tographviz()` function of **abn** (see later examples) which translates the matrix which defines a network - a directed acyclic graph - into a text file of suitable format for processing in Graphviz, where this processing was done outside of R. Graphviz is freely available and operates on most platforms and can be downloaded from www.graphviz.org, there is also an R package which interfaces to Graphviz available from the Bioconductor project (requires an installation of Graphviz).

3.2. Fitting an additive BN model to categorical data

An additive BN model for categorical data can be constructed by considering each individual variable as a logistic regression of the other variables in the data, and hence the network model comprises of many combinations of local logistic regressions (Rijmen 2008). The parameters in this model are the additive terms in a usual logistic regression and independent Gaussian priors are assumed for each covariate. Note that the variables here must all be binary, and so all multinomial variables need to be split into separate binary factors (and added to the original data.frame) in order to form the network model. This is analogous to forming the design matrix in a conventional additive model analysis. Similarly, interaction terms can be added by including appropriate additional columns in the data.frame. In these models the log marginal likelihood (network score) is estimated using Laplace approximations at each node. Hyperparameters for the means and variances in the Gaussian priors are fixed at zero and 1000 respectively, and other values can be given explicitly in the call to `fitabn` but this is not recommended without good reason.

To fit an additive model use `fitabn(data.df,dag.m,data.dists, ...)`. In the following code we fit first the independence model with no arcs and then the same dependence model as above. Turning on `verbose=TRUE` simply gives the individual log marginal likelihoods for each node (n.b. the numbering is that used internally and simply denotes the variables in the data.frame from left to right).

The following code fits a network to the subset of the variables from `var33` which are categorical. In

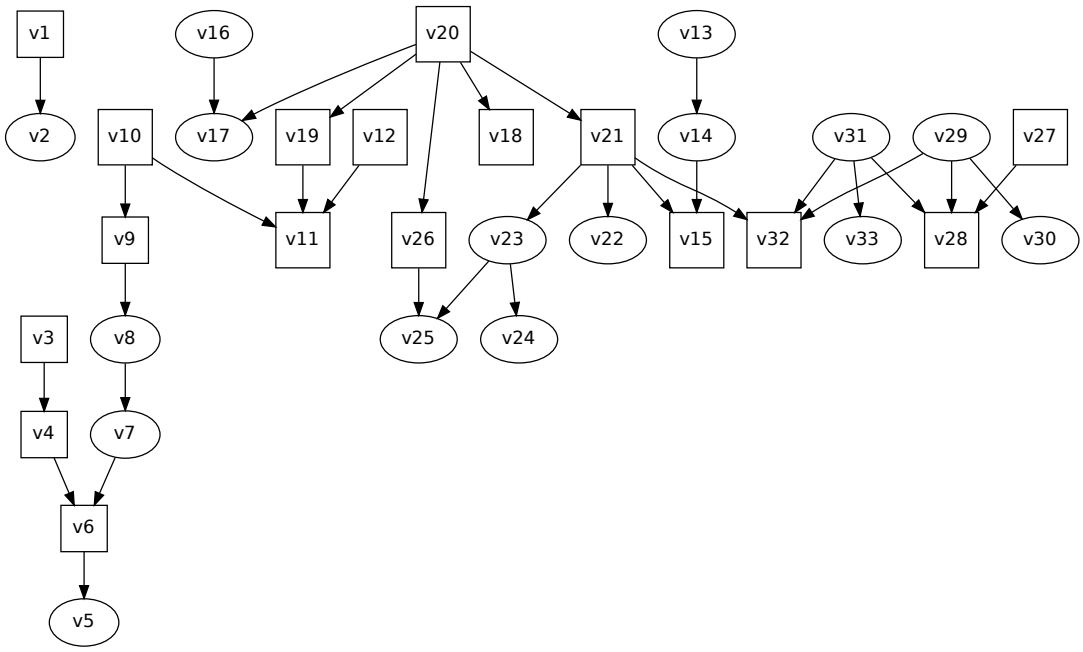

Figure 8: Directed acyclic graph representation of the joint probability distribution which generated data set `var33` which is included with **abn**. The square nodes are categorical (binary) and the oval nodes continuous variables.

this data these are all binary. Note that all categorical variables should be set as factors.

```
library( abn)

## Loading required package:  nnet
## Loading required package:  MASS
## Loading required package:  lme4
## Loading required package:  Matrix

bin.nodes <- c( 1,3,4,6,9,10,11,12,15,18,19,20,21,26,27,28,32)
var33.cat <- var33[,bin.nodes] #Categorical nodes only

dag33 <- matrix( 0, 17, 17)
colnames( dag33) <- rownames( dag33) <- names( var33.cat)#Set names
```

Move back to independence model:

```
dag33["v11","v12"] <- 0
dag33["v11","v10"] <- 0
dag33["v4","v3"] <- 0
```

Setup the distribution list for each categorical node:

```
mydists.cat <- list( v1 ="binomial", v3 = "binomial",
       v4 = "binomial",  v6 = "binomial",  v9 = "binomial",
      v10 = "binomial", v11 = "binomial", v12 = "binomial",
      v15 = "binomial", v18 = "binomial", v19 = "binomial",
      v20 = "binomial", v21 = "binomial", v26 = "binomial",
      v27 = "binomial", v28 = "binomial", v32 = "binomial")
ind.mod.cat <- fitabn( data.df=var33.cat, dag.m=dag33,
                       data.dists=mydists.cat, verbose=FALSE)
```

It is possible to change to `verbose=TRUE` if one want to check how change the score for each individual node.

The network score for a model with conditional independencies is:

```
ind.mod.cat$mlik

## [1] -2856.948
```

The structure of the network definition matrix is where each row is a "child" and each column is its "parents", where a `1` denotes a parent (or arc) is present. Now lets fit a model with some conditional dependencies, for example where `v11` is conditionally dependent upon `v12` and `v10`, and `v4` is conditionally dependent upon `v3`.

Now fit the model with some conditional dependencies:

```
dag33["v11","v12"] <- 1;
dag33["v11","v10"] <- 1;
dag33["v4","v3"] <- 1;
dep.mod.cat <- fitabn( data.df=var33.cat, dag.m=dag33,
                       data.dists=mydists.cat, verbose=FALSE)
```

The network score for a model with conditional dependencies is:

```
dep.mod.cat$mlik

## [1] -2850.081
```

The network score is considerably improved and therefore suggests support for these new structural features. To produce a visual description of the model then we can export to graphviz as follows:

```
tographviz( dag=dag33, data.df=var33.cat, data.dists=mydists.cat,
            outfile=paste0(datadir,"mydagcat.dot"), directed=TRUE)
```

`mydagcat.dot` can then be processed with graphviz unix shell typing: "dot -Tpdf mydagcat.dot -o mydagcat.pdf" or using gedit if on Windows.

In `tographviz()` the `data.df` argument is used to determine whether the variable is a factor or not, where factors are displayed as squares and non-factors as ovals. To use the full range of visual

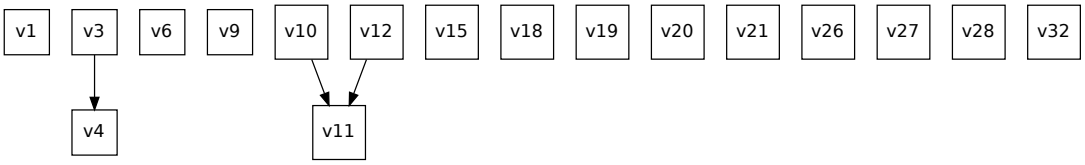

Figure 9: Directed acyclic graph `dag33` created using `tographviz()` and Graphviz

Graphviz options simply use the file created by `tographviz()` as a starting point and manually edit this in a text editor before running through `dot` or one of the other Graphviz layout processors.

3.3. Fitting an additive BN model to continuous data

We now consider analogous models to those in Section 3.2 but where the network comprises of Gaussian linear regressions rather than logistic regressions. The structure of these models again assumes independent Gaussian priors for each of the coefficients in the additive components for the mean response at each node (with hyper means = 0 and hyper variances = 1000). The Gaussian response distribution is parameterized in terms of precision $(1/\sigma^2)$, and independent Gamma priors are used with shape=0.001 and scale=1/0.001 (where these are as defined in the `rgamma` help page). By default, each variable in the data.frame is standardised to a mean of zero and standard deviation of one, this has no effect on the identification of dependencies between variables. We start droping the categorical nodes:

```
var33.cts <- var33[,-bin.nodes]
dag33 <- matrix( 0, 16, 16)
colnames( dag33) <- rownames( dag33) <- names( var33.cts)
```

Setup the distribution list for each continuous node:

```
mydists.cts <- list(       v2 = "gaussian",  v5 = "gaussian",
         v7 = "gaussian",  v8 = "gaussian", v13 = "gaussian",
        v14 = "gaussian", v16 = "gaussian", v17 = "gaussian",
        v22 = "gaussian", v23 = "gaussian", v24 = "gaussian",
        v25 = "gaussian", v29 = "gaussian", v30 = "gaussian",
        v31 = "gaussian", v33 = "gaussian")
```

Now fit the model defined in dag33 - full independence

```
ind.mod.cts <- fitabn( data.df=var33.cts, dag.m=dag33,
                       data.dists=mydists.cts, verbose=FALSE)
```

We use as default priors, a Gaussian density with 0 mean and variance 1000. While for the precision, inverse of variance, we use a Gamma density of hyperparameters 0.001 and $1/0.001$. This is the network score (goodness of fit, log marginal likelihood):

```
ind.mod.cts$mlik

## [1] -5949.52
```

Now fit a model with conditional dependencies, for example let `v33` depend on `v31`, and `v24` depend on `v23`, and `v14` depend on `v13`.

```
dag33["v33","v31"] <- 1;
dag33["v24","v23"] <- 1;
dag33["v14","v13"] <- 1;
dep.mod.cts <- fitabn( data.df=var33.cts, dag.m=dag33,
                data.dists=mydists.cts, verbose=FALSE)
```

The network score for a model with conditional independence is:

```
dep.mod.cts$mlik

tographviz( dag=dag33, data.df=var33.cts, data.dists=mydists.cts,
            outfile=paste0(datadir,"mydagcts.dot"), directed=TRUE)
```

`mydagcat.dot` can then be processed with graphviz unix shell typing: "dot -Tpdf mydagcat.dot -o mydagcat.pdf" or using gedit if on Windows.

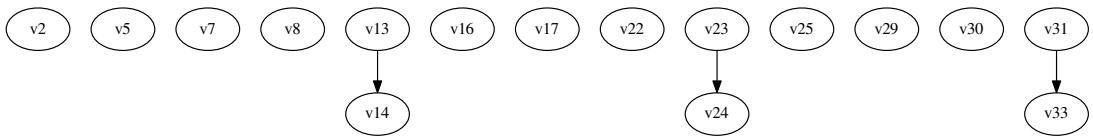

Figure 10: Directed acyclic graph `dag33` for continuous variables only created using `tographviz()` and Graphviz

3.4. Fitting an additive BN model to mixed data

To conclude the fitting of a single pre-specified model to data, e.g. based on expert opinion, we consider an additive BN model which comprises both binary and Gaussian nodes and this comprises of a combination of Binomial (logistic) and Gaussian linear models. Again `fitabn()` is used and the code is almost identical to the previous examples.

```
dag33 <- matrix( 0, 33, 33)
colnames( dag33) <- rownames( dag33)<- names( var33)
```

Setup distribution list for each mixed node:

```
mydists.mix <- list(      v1 = "binomial",  v2 = "gaussian",
        v3 = "binomial",  v4 = "binomial",  v5 = "gaussian",
        v6 = "binomial",  v7 = "gaussian",  v8 = "gaussian",
        v9 = "binomial", v10 = "binomial", v11 = "binomial",
       v12 = "binomial", v13 = "gaussian", v14 = "gaussian",
       v15 = "binomial", v16 = "gaussian", v17 = "gaussian",
       v18 = "binomial", v19 = "binomial", v20 = "binomial",
       v21 = "binomial", v22 = "gaussian", v23 = "gaussian",
       v24 = "gaussian", v25 = "gaussian", v26 = "binomial",
       v27 = "binomial", v28 = "binomial", v29 = "gaussian",
       v30 = "gaussian", v31 = "gaussian", v32 = "binomial",
       v33 = "gaussian")
```

Now fit the model defined in dag33, full independence:

```
ind.mod <- fitabn( data.df=var33, dag.m=dag33,
          data.dists=mydists.mix, verbose=FALSE)
```

The network score with no conditional dependencies is:

```
ind.mod$mlik

## [1] -8806.468
```

We now fit a BN model which has the same structure as the joint distribution used to generate the data and later create a visual graph of this model. We then define a model with many independencies:

```
dag33[2,1] <- 1;
dag33[4,3] <- 1;
dag33[6,4] <- 1; dag33[6,7] <- 1;
dag33[5,6] <- 1;
dag33[7,8] <- 1;
dag33[8,9] <- 1;
dag33[9,10] <- 1;
dag33[11,10] <- 1; dag33[11,12] <- 1; dag33[11,19] <- 1;
dag33[14,13] <- 1;
dag33[17,16] <- 1;dag33[17,20] <- 1;
dag33[15,14] <- 1; dag33[15,21] <- 1;
dag33[18,20] <- 1;
dag33[19,20] <- 1;
dag33[21,20] <- 1;
dag33[22,21] <- 1;
dag33[23,21] <- 1;
dag33[24,23] <- 1;
dag33[25,23] <- 1; dag33[25,26] <- 1;
dag33[26,20] <- 1;
```

```
dag33[33,31] <- 1;
dag33[33,31] <- 1;
dag33[32,21] <- 1; dag33[32,31] <- 1; dag33[32,29] <- 1;
dag33[30,29] <- 1;
dag33[28,27] <- 1; dag33[28,29] <- 1; dag33[28,31] <- 1;
dep.mod <- fitabn( data.df=var33, dag.m=dag33,
           data.dists=mydists.mix, verbose=FALSE)
```

The network score for a model with conditional independence is:

```
dep.mod$mlik
tographviz( dag=dag33, data.df=var33, data.dists=mydists.mix,
            outfile=paste0(datadir,"mydag_all.dot"), directed=TRUE)
```

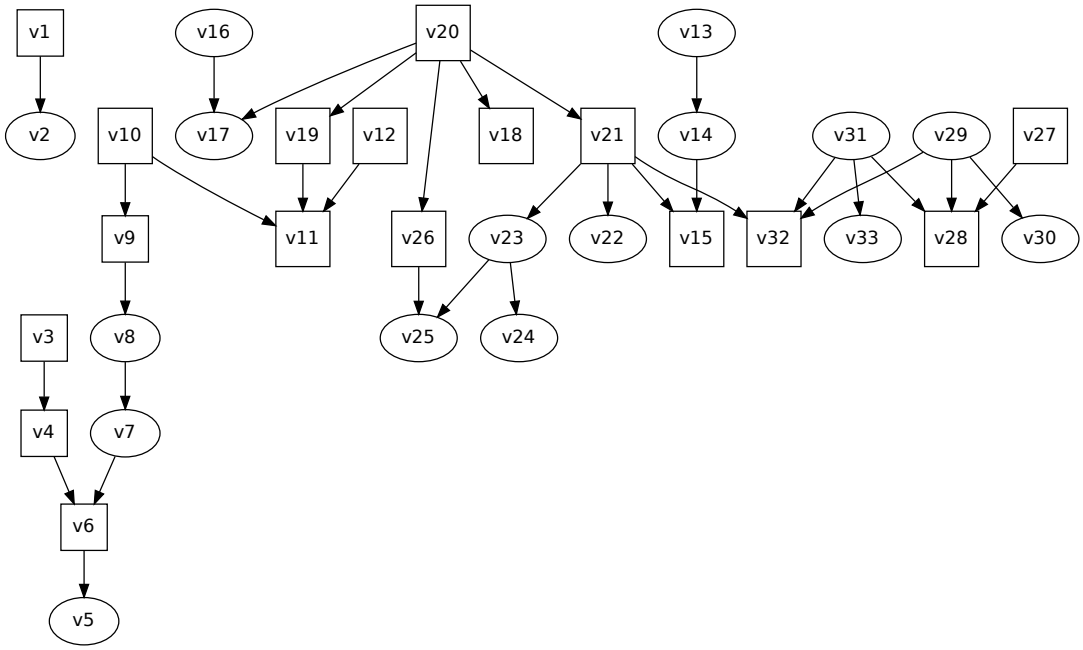

Figure 11: Directed acyclic graph `dag33` for mixed continuous and discrete variables

3.5. Model fitting validation

In order to validate the additive models for mixed binary and Gaussian models, estimates of the posterior distributions for the model parameters using Laplace approximations were compared with those estimated using Markov chain Monte Carlo. These were always in very close agreement for the range of models and data examined. This is an indirect validation of the Laplace estimate of the network score, e.g. if the posterior densities match closely then this implies that the denominator (the marginal likelihood - network score) must also be accurately estimated, as a "gold standard" estimate of the network score is generally unavailable for such non-conjugate models.

4. Further insights into searching strategy

The key objective of the **abn** library is to enable estimation of statistical dependencies in data comprising of multiple variables - that is, find a DAG which is robust and representative of the dependency structure of the (unknown) stochastic system which generated the observed data. The challenge here is that with such a vast model space it is impossible to enumerate over all possible DAGs, and there may be very many different DAGs with similar goodness of fit. In the next sections we first consider searching for additive (non-conjugate) models.

4.1. Single search for optimal additive BN model from categorical data

To run a single search heuristic use `searchHillclimber()`. This commences from a randomly created DAG which is constructed by randomly adding arcs to an empty network until all possible arcs have been tried. The function `searchHillclimber()` then searches stepwise from the initial random network for an improved structure, where three stepwise operations are possible: i) add an arc; ii) remove and arc; or iii) reverse and arc. The stepwise search is subject to a number of conditions, firstly only moves that do not generate a cycle are permitted, secondly, a parent limit is imposed which fixes the maximum number of parents which each child node can have (arcs go from parent to child), and thirdly it is possible to ban or retain arcs. If provided, `banned.m` is a matrix which defines arcs that are not allowed to be considered in the search process (or in the creation of the initial random network). Similarly, `retain.m` includes arcs which must always be included in any model. It is also possible to specific an explicit starting matrix, `start.m` and if using a retain matrix then `start.m` should contain at least all those arcs present in `retain.m`. Note that only very rudimentary checking is done to make sure that the ban, retain and start networks - if user supplied - are not contradictory.

To improve the computational performance of `searchHillclimber()` a cache of all possible goodness of fit must be built in advance, using the function `buildscorecache()`. Rather than re-calculate the score for each individual node in the network (the overall network score is the product of all the scores for the individual nodes) the score for each unique node found during the search is stored in the cache created by the function `buildscorecache()`.

```
bin.nodes <- c( 1,3,4,6,9,10,11,12,15,18,19,20,21,26,27,28,32)
var33.cat <- var33[,bin.nodes] #Categorical nodes only
dag33 <- matrix( 0, 17, 17)
colnames(dag33) <- rownames(dag33) <- names(var33.cat)#Set names
```

Create banned and retain empty DAGs:

```
banned.cat <- matrix( 0, 17, 17)
colnames(banned.cat) <- rownames(banned.cat) <- names(var33.cat)
retain.cat <- matrix( 0, 17, 17)
colnames(retain.cat) <- rownames(retain.cat) <- names(var33.cat)
```

Setup distribution list for each categorical node:

```
mydists.cat <- list(     v1 = "binomial",  v3 = "binomial",
       v4 = "binomial",  v6 = "binomial",  v9 = "binomial",
      v10 = "binomial", v11 = "binomial", v12 = "binomial",
```

```
      v15 = "binomial", v18 = "binomial", v19 = "binomial",
      v20 = "binomial", v21 = "binomial", v26 = "binomial",
      v27 = "binomial", v28 = "binomial", v32 = "binomial")
```

Build cache of all the local computations this information is needed later when running a model search:

```
mycache.cat <- buildscorecache( data.df=var33.cat,
                  data.dists=mydists.cat, dag.banned=banned.cat,
                  dag.retained=retain.cat, max.parents=1)
```

Running a single search heuristic for an additive BN uses `searchHillclimber()`. It uses a parameter prior specifications (as detailed above). Several additional arguments are available which relate to the numerical routines used in the Laplace approximation to calculate the network score. The defaults appear to work reasonably well in practice and if it is not possible to calculate a robust value for this approximation in any model, for example due to a singular design matrix at one or more nodes, then the model is simply assigned a log network score of $-\infty$ which effectively removes it from the model search.

Run a single search heuristic for an additive BN:

```
heur.res.cat <- searchHillclimber( score.cache=mycache.cat,
                          num.searches=1, timing.on=FALSE)
```

4.2. Single search for optimal additive BN model for continuous data

As above but for a network of Gaussian nodes.

```
var33.cts <- var33[,-bin.nodes] #Drop categorical nodes
dag33 <- matrix( 0, 16, 16)
colnames(dag33) <- rownames(dag33) <- names(var33.cts) #Set names
banned.cts <- matrix( 0, 16, 16)
colnames(banned.cts) <- rownames(banned.cts) <- names(var33.cts)
retain.cts <- matrix( 0, 16, 16)
colnames(retain.cts) <- rownames(retain.cts) <- names(var33.cts)
```

Setup distribution list for each continuous node:

```
mydists.cts <- list(       v2 = "gaussian",  v5 = "gaussian",
         v7 = "gaussian",  v8 = "gaussian", v13 = "gaussian",
        v14 = "gaussian", v16 = "gaussian", v17 = "gaussian",
        v22 = "gaussian", v23 = "gaussian", v24 = "gaussian",
        v25 = "gaussian", v29 = "gaussian", v30 = "gaussian",
        v31 = "gaussian", v33 = "gaussian")
```

Build cache of all local computations, information needed later when running a model search:

```
mycache.cts <- buildscorecache( data.df=var33.cts,
             data.dists=mydists.cts, dag.banned=banned.cts,
             dag.retained=retain.cts, max.parents=1)
```

Run a single search heuristic for an additive BN:

```
heur.res.cts <- searchHillclimber( score.cache=mycache.cts,
                              num.searches=1, timing.on=FALSE)
```

4.3. Single search for optimal additive BN model for mixed data

Model searching for mixed data is again very similar to the previous examples. Note that in this example the parameter priors are specified explicitly (although those given are the same as the defaults). The `+1` in the hyperparameter specification is because a constant term is included in the additive formulation for each node.

```
dag33 <- matrix( 0, 33, 33)
colnames(dag33) <- rownames(dag33) <- names(var33)#Set names
```

Create empty DAGs:

```
banned.mix <- matrix( 0, 33, 33)
colnames(banned.mix) <- rownames(banned.mix) <- names(var33)
retain.mix<- matrix( 0, 33, 33)
colnames(retain.mix) <- rownames(retain.mix) <- names(var33)
```

Setup distribution list for mixed node:

```
mydists.mix <- list(       v1 = "binomial",  v2 = "gaussian",
         v3 = "binomial",  v4 = "binomial",  v5 = "gaussian",
         v6 = "binomial",  v7 = "gaussian",  v8 = "gaussian",
         v9 = "binomial", v10 = "binomial", v11 = "binomial",
        v12 = "binomial", v13 = "gaussian", v14 = "gaussian",
        v15 = "binomial", v16 = "gaussian", v17 = "gaussian",
        v18 = "binomial", v19 = "binomial", v20 = "binomial",
        v21 = "binomial", v22 = "gaussian", v23 = "gaussian",
        v24 = "gaussian", v25 = "gaussian", v26 = "binomial",
        v27 = "binomial", v28 = "binomial", v29 = "gaussian",
        v30 = "gaussian", v31 = "gaussian", v32 = "binomial",
        v33 = "gaussian")
```

Build cache of all local computations, information needed later when running a model search:

```
mycache.mix <- buildscorecache( data.df=var33,
             data.dists=mydists.mix, dag.banned=banned.mix,
             dag.retained=retain.mix, max.parents=1)
```

Run a single search heuristic for an additive BN:

```
heur.res.mix <- searchHillclimber( score.cache=mycache.mix,
                              num.searches=1, timing.on=FALSE)
```

4.4. Multiple Search Strategies

To estimate a robust additive BN for a given dataset is it necessary to run many searches and then summarize the results of these searches. The function `searchHillclimber()` with `num.searches>1` run multiple searches. It is necessary to use a single joint node cache over all searches, using the function `buildscorecache`.

Conceptually it may seem more efficient to use one global node cache to allow node information to be shared between different searches, however, in practice as the search space is so vast for some problems this can result in extremely *slow* searches. As the cache becomes larger it can take much more time to search it (and it may need to be searched a very large number of times) than to simply perform the appropriate numerical computation. Profiling using the google performance tool google-pprof suggests that more than 80% of the computation time may be taken up by lookups. When starting searches from different random places in the model space the number of individual node structures in common between any two searches, relative to the total number of different node structures searched over can be very small meaning a common node cache is inefficient. This may not be the case when starting networks are relatively similar.

To help with performance monitoring it is possible to turn on timings using `timing.on=TRUE` which then outputs the number of seconds of CPU time each individual search takes (using standard libc functions declared in time.h).

```
dag33 <- matrix( 0, 33, 33)
colnames(dag33) <- rownames(dag33) <- names(var33) #Set names
```

Create empty DAGs:

```
banned.mix <- matrix( 0, 33, 33)
colnames(banned.mix)<- rownames(banned.mix)<- names(var33)
retain.mix <- matrix( 0, 33, 33)
colnames(retain.mix) <- rownames(retain.mix)<- names(var33)
```

Setup distribution list for mixed node:

```
mydists.mix <- list(       v1 = "binomial",   v2 = "gaussian",
        v3 = "binomial",   v4 = "binomial",   v5 = "gaussian",
        v6 = "binomial" , v7 = "gaussian",   v8 = "gaussian",
        v9 = "binomial", v10 = "binomial", v11 = "binomial",
```

```
        v12 = "binomial", v13 = "gaussian", v14 = "gaussian",
        v15 = "binomial", v16 = "gaussian", v17 = "gaussian",
        v18 = "binomial", v19 = "binomial", v20 = "binomial",
        v21 = "binomial", v22 = "gaussian", v23 = "gaussian",
        v24 = "gaussian", v25 = "gaussian", v26 = "binomial",
        v27 = "binomial", v28 = "binomial", v29 = "gaussian",
        v30 = "gaussian", v31 = "gaussian", v32 = "binomial",
        v33 = "gaussian")
n.searches <-  10
```

The number 10 is an example only, it must be much larger in practice. Set the parent limits:

```
max.par <- 1
```

We set only one parent, because the search with `buildscorecache()` take some minutes. Now we build the cache:

```
mycache.mix <- buildscorecache( data.df=var33, data.dists=mydists.mix,
   dag.banned=banned.mix, dag.retained=retain.mix, max.parents=max.par)
```

Repeat but this time have the majority consensus network plotted as the searches progress:

```
myres.mlp <- searchHillclimber(score.cache=mycache.mix,
                  num.searches=n.searches, timing.on=FALSE)
```

4.5. Creating a Summary Network: Majority Consensus

Having run many heuristic searches, then the next challenge is to summarise these results to allow for ready identification of the joint dependencies most supported by the data. One common, and very simple approach is to produce a single robust BN model of the data mimicing the approach used in phylogenetics to create majority consensus trees. A majority consensus DAG is constructed from all the arcs present in at least 50% of the locally optimal DAGs found in the search heuristics. This creates a single summary network. Combining results from different runs of `searchHillclimber()` is straightforward, although note that it is necessary to check for duplicate random starting networks, as while highly unlikely this is theoretically possible. The following code provides a simple way to produce a majority consensus network and Figure 12 shows the resulting network - note that this is an example only and many thousands of searches may need to be conducted to achieve robust results. One simple ad-hoc method for assessing how many searches are needed is to run a number of searches and split the results into two (random) groups, and calculate the majority consensus network within each group. If these are the same then it suggests that sufficient searches have been run. To plot the majority consensus network use the result of the function `searchHillclimber`, see below for some example.

```
tographviz( dag=myres.mlp$consensus, data.df=var33,
   data.dists=mydists.mix, outfile=paste0(datadir,"dagcon.dot"))
```

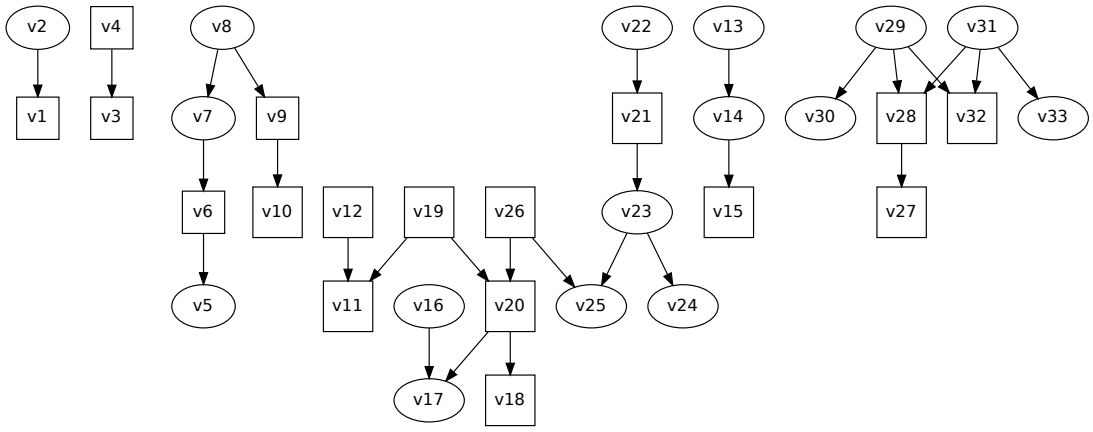

Figure 12: Example majority consensus network (from the results of only 10 searches)

`dagcon.dot` can then be processed with graphviz un a unix shell typing: "dot -Tpdf dagcon.dot -o dagcon.pdf" or using gedit if on Windows.

4.6. Creating a Summary Network: Pruning

Rather than use the majority consensus network as the most appropriate model of the data, an alternative approach is to choose the single best model found during a large number of searches. To determine sufficient heuristic searches have been run to provide reasonable coverage of all the features of the model landscape, then again checking for a stable majority consensus network as in Section 4.5, seems a sensible approach. Once the best overall DAG has been identified then the next task is to check this model for over-fitting. Unlike with the majority consensus network, which effective "averages" over many different competing models and therefore should generally comprise only robust structural features, choosing the DAG from a single model search is far more likely to contain some spurious features. When dealing with smaller data sets, say, of several hundred observations then this is extremely likely, as can easily be demonstrated using simulated data. A simple assessment of overfitting can be made by comparing the number of arcs in the majority consensus network with the number of arcs in the best fitting model. We have found that in larger data sets the majority consensus and best fitting model can be almost identical, while in smaller data sets the best fitting models may have many more arcs - suggesting a degree of overfitting.

An advantage of choosing a DAG from an individual search is that unlike averaging over lots of different structures, as in the construction of a majority consensus network, the model chosen here has a structure which was actually found during a search across the model landscape. In contrast, the majority consensus network is a derived model which may never have been found chosen during even an exhaustive search, indeed it may even comprise of contradictory features as is a usual risk in averaging over different explanations (models) of data. In addition, a majority consensus network need also not be acyclic, although in practice this can be easily corrected by reversing one or more arcs to produce an appropriate DAG.

A simple compromise between the risk of over-fitting in choosing the single highest scoring DAG, and the risk of inappropriately averaging across different distinct data generating processes, is to prune the highest scoring DAG using the majority consensus model. In short, an element by element multiply

of the highest scoring DAG and the majority consensus DAG, which gives a new DAG which only contains the structural features in *both* models.

5. Summary

The **abn** library provides a range of Bayesian network models to assist with identifying statistical dependencies in complex data, in particular models which are multidimensional analogues of generalised linear models. This process is typically referred to as structure learning, or structure discovery, and is computational extremely challenging. Heuristics are the only options for data comprising of larger numbers of variables. As with all model selection, over-modelling is an everpresent danger and using either: i) summary models comprising of structural features present in many locally optimal models or else; ii) using parametric bootstrapping to determine the robustness of the features in a single locally optimal model are likely essential to provide robust results. An alternative presented was exact order based searches, in particular finding the globally most probable structure. This approach is appealing as it is exact, but despite collapsing DAGs into orderings for larger scale problems it may not be feasible. For further in-depth analysis about **abn** refer to the website: www.r-bayesian-networks.org.

References

Babyak MA (2004). *What you see may not be what you get: a brief, nontechnical introduction to overfitting in regression-type models.* Psychosomatic Medicine.

Boettcher SG (2004). *Learning Bayesian networks with mixed variables*. Ph.D. thesis, Aalborg University - Department of Mathematical Sciences.

Buntine W (1991). "Theory refinement on Bayesian networks." In *Uncertainty in artificial intelligence: proceedings of the seventh conference*, pp. 52–60. Morgan Kaufmann Publishers Inc.

Djebbari A, Quackenbush J (2008). "Seeded Bayesian Networks: Constructing genetic networks from microarray data." *BMC Systems Biology*, **2**, 57.

Dojer N, Gambin A, Mizera A, Wilczynski B, Tiuryn J (2006). "Applying dynamic Bayesian networks to perturbed gene expression data." *BMC Bioinformatics*, **7**, 249.

Firestone SM, Lewis FI, Schemann K, Ward MP, Toribio JA, Dhand NK (2013). "Understanding the associations between on-farm biosecurity practice and equine influenza infection during the 2007 outbreak in Australia." *Preventive Veterinary Medicine*, **110**(1), 28–36. ISSN 0167-5877.

Firestone SM, Lewis FI, Schemann K, Ward MP, Toribio JA, Taylor MR, Dhand NK (2014). "Applying Bayesian network modelling to understand the links between on-farm biosecurity practice during the 2007 equine influenza outbreak and horse managers' perceptions of a subsequent outbreak." *Preventive Veterinary Medicine*, **116**(3), 243–251. ISSN 0167-5877.

Friedman N, Goldszmidt M, Wyner A (1999). "Data analysis with Bayesian networks: A Bootstrap approach." In *Proc. Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI'99) (pp.206-215). San Francisco: Morgan Kaufmann.*

Friedman N, Koller D (2003). "Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks." *Machine Learning*, **50**(1-2), 95–125.

Geiger D, Heckerman D (1994). "Learning Gaussian networks." In *Proceedings of Tenth Conference on Uncertainty in Artificial Intelligence*, UAI'94, pp. 235–243. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1-55860-332-8.

Heckerman D, Geiger D, Chickering DM (1995). "Learning Bayesian networks: the combination of knowledge and statistical data." *Machine Learning*, **20**(3), 197–243.

Hodges AP, Dai DJ, Xiang ZS, Woolf P, Xi CW, He YQ (2010). "Bayesian network expansion identifies new ros and biofilm regulators." *PLOS One*, **5**(3), e9513.

Jansen R, Yu HY, Greenbaum D, Kluger Y, Krogan NJ, Chung SB, Emili A, Snyder M, Greenblatt JF, Gerstein M (2003). "A Bayesian networks approach for predicting protein-protein interactions from genomic data." *Science*, **302**(5644), 449–453.

Jensen FV (2001). *Bayesian network and decision graphs*. Springer-Verlag, New York.

Koivisto M, Sood K (2004). "Exact Bayesian structure discovery in Bayesian networks." *Journal of Machine Learning Research*, **5**, 549–573.

Lauritzen SL (1996). *Graphical Models*. Oxford Univ Press, New York.

Lewis F, Ward M (2013). "Improving epidemiologic data analyses through multivariate regression modelling." *Emerging Themes in Epidemiology*, **10**(1), 4. ISSN 1742-7622.

Lewis FI (2012). "Bayesian networks as a tool for epidemiological systems analysis." In S Sivasundaram (ed.), *9th International Conference On Mathematical Problems In Engineering, Aerospace and Sciences (icnpaa 2012), Amer Inst Physics*, volume 1493 of *AIP Conference Proceedings*, pp. 610–617. ISBN 978-0-7354-1105-0. ISSN 0094-243X.

Lewis FI, Brulisauer F, Gunn GJ (2011). "Structure discovery in Bayesian networks: An analytical tool for analysing complex animal health data." *Preventive Veterinary Medicine*, **100**(2), 109–115.

Lewis FI, McCormick BJJ (2012). "Revealing the complexity of health determinants in resource-poor settings." *American Journal of Epidemiology*, **176**(11), 1051–1059.

Ludwig A, Berthiaume P, Boerlin P, Gow S, Léger D, Lewis FI (2013). "Identifying associations in Escherichia coli antimicrobial resistance patterns using additive Bayesian networks." *Preventive Veterinary Medicine*, **110**(1), 64–75. ISSN 0167-5877.

Lycett SJ, Ward MJ, Lewis FI, Poon AFY, Pond SLK, Brown AJL (2009). "Detection of mammalian virulence determinants in highly pathogenic avian influenza H5N1 viruses: multivariate analysis of published data." *Journal of Virology*, **83**(19), 9901–9910.

Mackay DJC (1992). "Bayesian Interpolation." *Neural Computation*, **4**(3), 415–447.

McCormick B, Sanchez-Vazquez M, Lewis F (2013). "Using Bayesian networks to explore the role of weather as a potential determinant of disease in pigs." *Preventive Veterinary Medicine*, **110**(1), 54–63. ISSN 0167-5877.

Needham CJ, Bradford JR, Bulpitt AJ, Westhead DR (2007). "A primer on learning in Bayesian networks for computational biology." *PLOS Computational Biology*, **3**(8), e129.

Plummer M (2003). "JAGS: a program for analysis of Bayesian graphical models using Gibbs sampling." *Proc 3rd Int Work Dist Stat Comp*.

Plummer M, Best N, Cowles K, Vines K (2006). "CODA: Convergence Diagnosis and Output Analysis for MCMC." *R News*, **6**(1), 7–11.

Poon AFY, Lewis FI, Frost SDW, Pond SLK (2008). "Spidermonkey: rapid detection of co-evolving sites using Bayesian graphical models." *Bioinformatics*, **24**(17), 1949–1950.

Poon AFY, Lewis FI, Pond SLK, Frost SDW (2007a). "Evolutionary interactions between N-linked glycosylation sites in the HIV-1 envelope." *PLOS Computational Biology*, **3**(1), e11.

Poon AFY, Lewis FI, Pond SLK, Frost SDW (2007b). "An evolutionary-network model reveals stratified interactions in the V3 loop of the HIV-1 envelope." *PLOS Computational Biology*, **3**(11), e231.

Rijmen F (2008). "Bayesian networks with a logistic regression model for the conditional probabilities." *International Journal of Approximate Reasoning*, **48**(2), 659–666.

Sanchez-Vazquez M, Nielen M, Edwards S, Gunn G, Lewis F (2012). "Identifying associations between pig pathologies using a multi-dimensional machine learning methodology." *BMC Veterinary Research*, **8**(1), 151. ISSN 1746-6148.

Schemann K, Lewis FI, Firestone SM, Ward MP, Toribio JALML, Taylor MR, Dhand NK (2013). "Untangling the complex inter-relationships between horse managers' perceptions of effectiveness of biosecurity practices using Bayesian graphical modelling." *Preventive Veterinary Medicine*, **110**(1, SI), 37–44. ISSN 0167-5877.

Ward MP, Lewis FI (2013). "Bayesian graphical modelling: applications in veterinary epidemiology." *Preventive Veterinary Medicine*, **110**(1), 1–3. ISSN 0167-5877.

Wilson AJ, Ribeiro R, Boinas F (2013). "Use of a Bayesian network model to identify factors associated with the presence of the tick Ornithodoros erraticus on pig farms in southern Portugal." *Preventive Veterinary Medicine*, **110**(1, SI), 45–53. ISSN 0167-5877.

Affiliation:

Gilles Kratzer
Institute of Mathematics, University of Zurich
Winterthurerstrasse 190, Zurich 8057, Switzerland
E-mail: gilles.kratzer@math.uzh.ch

Marta Pittavino
Institute of Mathematics, University of Zurich
Winterthurerstrasse 190, Zurich 8057, Switzerland
E-mail: marta.pittavino@math.uzh.ch

Fraser Ian Lewis
Office of Health Economics
United Kingdom, London
E-mail: flewis@ohe.org

Reinhard Furrer
Institute of Mathematics, University of Zurich
Institute of Computational Science, University of Zurich
Winterthurerstrasse 190, Zurich 8057, Switzerland
E-mail: reinhard.furrer@math.uzh.ch