Exploring data from complex systems using Additive Bayesian Networks in R

Fraser I. Lewis

Abstract

This vignette describes the **abn** package of R which comprises of model fitting and selection functionality for exploring multivariate data using additive Bayesian network models. These are directed acyclic graphs where each node in the graph comprises a generalized linear model, where a greedy search heuristic is used to identify high scoring models that attempt to identify relationships between all variables in the data. Currently implemented are models for data comprising of categorical and/or continuousqq variables where a logit link is used with the former. Laplace approximations are used to estimate marginal likelihoods and compute marginal posterior densities.

Keywords: R, Bayesian Networks, additive models, structure discovery.

1. Introduction

Bayesian network (BN) modeling (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 (Friedman and Koller 2003). 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. Bayesian networks have been developed for analyzing multinomial, multivariate Gaussian or conditionally Gaussian networks (a mix categorical and Gaussian variables). 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 do 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 is planned to extend this to include Poisson distributions for count data and then more complex distributions for overdispersed data such a beta-binomial and negative binomial models.

The general objective in BN modeling/structure discovery is to perform a model search on the data to identify a locally 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 (Heckerman *et al.* 1995); alternatively, a single summary network can be constructed using results across many different searches (Hodges, Dai, Xiang, Woolf, Xi, and He 2010; Poon, Lewis, Pond, and Frost 2007). There are obvious pros and cons to either approach 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, Bradford, Bulpitt, and Westhead 2007. A case study in applying BN models to epidemiological data using the conjugate BN functionality in **abn** can be found in Lewis, Brulisauer, and Gunn 2011.

In this vignette we consider a series of examples illustrating how to fit different types of models and run different searches and summary analyzes 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 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 sufficiently complex to provide a realistic example of data mining using Bayesian Network modeling.

2. Case Study Data

Figure 1 shows the structure of the distribution which generated the data set `var33` included

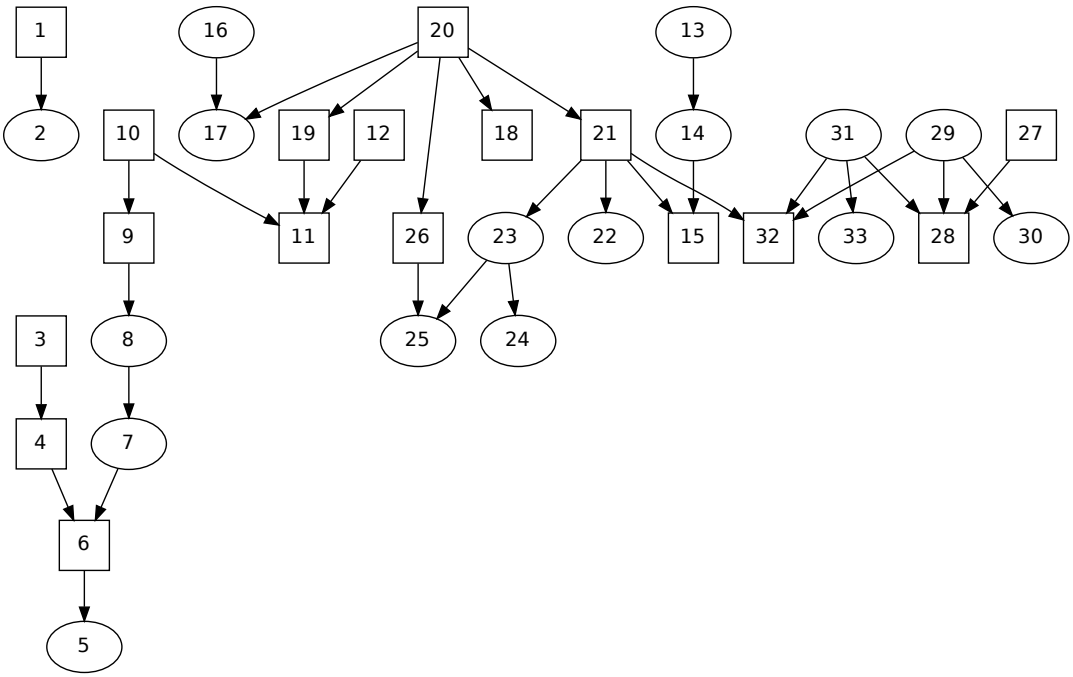

Figure 1: Directed acyclic graph representation of the joint probability distribution which generated data set `var33` which is included with `abn`. The square nodes are categorical (binary) and the oval nodes continuous variables.

with `abn`. This diagram was created using the `tographviz()` function of `abn` (see later examples) which translates a 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. Fitting a single BN model to data

In the next four sections we illustrate how to fit a BN model to different kinds of data. The main purpose of BN analyses 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 the (log) marginal likelihood, typically referred to as the network score.

3.1. Fitting a BN model to categorical data

A conjugate Bayesian network applied to categorical data is the classical application of Bayesian network analysis. Here the data are considered as a contingency table of frequency counts and the model describes conditional dependencies between different cells. Note these are not additive models.

The function `fitbn(data.df, dag.m, prior.obs.per.node=NULL, useK2=FALSE, ...)` fits a multinomial conjugate BN model to the data in `data.df` where the model structure is defined in matrix `dag.m`. There are two choices of priors/goodness of fit metrics; the BDe metric and the K2 metric (see Heckerman *et al.* 1995). In brief, in the BDeu metric it is assumed that a number, `prior.obs.per.node`, of prior observations have been observed at each node and these are uniformly distributed across all the hyperparameters at each node. For example in Figure 1, node 4 is conditionally dependent upon node 3, these are binary nodes and the parameter to be estimated is $P(v4 = T|v3 = T)$ where this has a Beta distributed prior of $Beta(\alpha_1, \alpha_2)$ and supposing `prior.obs.per.node`=16, then with the BDeu metric we have a prior of $Beta(8, 8)$. Similarly, if there were two parents for node 4 then there would be four parameters to estimate (assuming both parents were binary) and in this case the prior for each parameter would be $Beta(2, 2)$ where again the sum of the hyperparameters equals 16. In contrast, in the K2 metric each and every parameter has a flat prior of $Beta(1, 1)$ for binary nodes and Dirichlet $Dir(1, \ldots, 1)$ for multinomial nodes. An advantage of the BDeu metric is that it is likelihood equivalent and so DAGs which are probabilistically equivalent will have identical BDeu network scores. The K2 metric, however, uses identical uninformative priors for each and every parameter which may also be desirable, but in which case the network scores for probabilistic identical networks may differ (although in practice such differences may be small). In `fitbn`, if the `useK2` argument is `TRUE` then `prior.obs.per.node` is ignored.

The following code fits a network to the subset of the variables from `var33` which are categorical. In this data these are all binary but `fitbn` works analogously for multinomial variables. Note that all categorical variables should be set as factors - and will be coerced if necessary.

```
> library(abn);# load library
```

```
> 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
> mydag<-matrix(c(
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v1
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v3
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v4
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v6
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v9
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v10
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v11
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v12
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v15
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v18
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v19
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v20
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v21
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v26
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v27
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v28
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0  #v32
+               ),byrow=TRUE,ncol=17);
> colnames(mydag)<-rownames(mydag)<-names(var33.cat);#set names
> ## now fit the model defined in mydag - full independence model
> fitbn (data.df=var33.cat, dag.m=mydag,useK2=TRUE);

[1] -2807.897

> # this is the network score goodness of fit = log marginal likelihood
```

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 upn `v3`.

```
> # now fit model with some conditional dependencies let v11
> ## depend jointly on v12 and v10
> mydag["v11","v12"]<-1;
> mydag["v11","v10"]<-1;
> ## let v4 depend on v3
> mydag["v4","v3"]<-1;
> fitbn (data.df=var33.cat, dag.m=mydag,useK2=TRUE);

[1] -2794.079

> # network score for a model with conditional independence
```

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=mydag,data.df=var33.cat,outfile="mydag.dot");#create file
> # mydag.dot can then be processed with graphviz
> # unix shell "dot -Tpdf mydag.dot -o mydag.pdf" or use gedit if on Windows
```

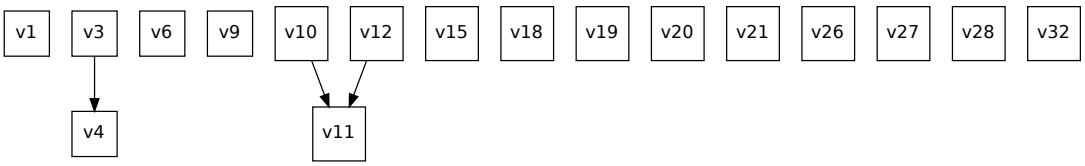

Figure 2: Directed acyclic graph `mydag` created using `tographviz()` and Graphviz

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 Graphviz options simply use the file created by `tographviz()` as a template and manually edit this in a text editor.

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. The parameters in this model are the additive terms in a usual logistic regression and independent Gaussian priors are assumed for each covariate. The covariates here must all be binary, and so 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 mean and standard deviations in the Gaussian priors can be specified but some care is required if informative priors are needed at different nodes (see manual page for `fitabn`).

To fit an additive model use `fitabn(data.df,dag.m, ...)`. 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).

```
> ## move back to independence model
> mydag["v11","v12"]<-0;mydag["v11","v10"]<-0;mydag["v4","v3"]<-0;
> fitabn (data.df=var33.cat,dag.m=mydag,verbose=TRUE);

Binary node=0 score=-178.004211
Binary node=1 score=-178.414495
```

```
Binary node=2 score=-168.248843
Binary node=3 score=-102.675919
Binary node=4 score=-167.644794
Binary node=5 score=-178.679732
Binary node=6 score=-174.534904
Binary node=7 score=-178.293870
Binary node=8 score=-143.134495
Binary node=9 score=-173.338554
Binary node=10 score=-174.152823
Binary node=11 score=-177.448401
Binary node=12 score=-177.448401
Binary node=13 score=-167.644794
Binary node=14 score=-178.735970
Binary node=15 score=-174.900353
Binary node=16 score=-163.647060

   ######################################################################
   ###      log marginal likelihood for Model: -2856.9476195086
   ######################################################################
[1] -2856.948

> # now fit the model with some conditional dependencies
> mydag["v11","v12"]<-1;mydag["v11","v10"]<-1;mydag["v4","v3"]<-1;
> fitabn (data.df=var33.cat, dag.m=mydag,verbose=TRUE);

Binary node=0 score=-178.004211
Binary node=1 score=-178.414495
Binary node=2 score=-166.945426
Binary node=3 score=-102.675919
Binary node=4 score=-167.644794
Binary node=5 score=-178.679732
Binary node=6 score=-168.972182
Binary node=7 score=-178.293870
Binary node=8 score=-143.134495
Binary node=9 score=-173.338554
Binary node=10 score=-174.152823
Binary node=11 score=-177.448401
Binary node=12 score=-177.448401
Binary node=13 score=-167.644794
Binary node=14 score=-178.735970
Binary node=15 score=-174.900353
Binary node=16 score=-163.647060

   ######################################################################
   ###      log marginal likelihood for Model: -2850.0814800302
   ######################################################################
[1] -2850.081
```

```
> # network score for a model with conditional independence
```

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, and as the model is parameterized in terms of precision ($1/\sigma^2$), independent Gamma priors are used for the precision parameter at each node.

```
> var33.cts<-var33[,-bin.nodes];#drop categorical nodes
> mydag<-matrix(c(
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v2
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v5
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v7
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v8
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v13
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v14
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v16
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v17
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v22
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v23
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v24
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v25
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v29
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v30
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v31
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0  #v33
+                ),byrow=TRUE,ncol=16);
> colnames(mydag)<-rownames(mydag)<-names(var33.cts);#set names
> ## now fit the model defined in mydag - full independence
> fitabn (data.df=var33.cts,dag.m=mydag,verbose=TRUE);

Gaussian node=0 score=-396.151434
Gaussian node=1 score=-395.967082
Gaussian node=2 score=-468.252121
Gaussian node=3 score=-384.496529
Gaussian node=4 score=-334.199036
Gaussian node=5 score=-436.487935
Gaussian node=6 score=-358.952748
Gaussian node=7 score=-450.626527
Gaussian node=8 score=-381.688832
Gaussian node=9 score=-379.543123
Gaussian node=10 score=-460.360575
Gaussian node=11 score=-480.835063
```

```
Gaussian node=12 score=-378.705867
Gaussian node=13 score=-458.988366
Gaussian node=14 score=-360.273962
Gaussian node=15 score=-447.594152

   ###################################################################
   ###      log marginal likelihood for Model: -6573.1233504991
   ###################################################################
[1] -6573.123

> ## uses default priors of N(mu=0,var=1000), 1/var=Gamma(0.001,1/0.001)
> # this is the network score goodness of fit = log marginal likelihood
```

Now fit a model with conditional independencies, for example

```
> # now fit model with some conditional dependencies let v33
> ## depend on v31, and v24 depend on 23, and v14 depend on v13
> mydag["v33","v31"]<-1;
> mydag["v24","v23"]<-1;
> mydag["v14","v13"]<-1;
> fitabn (data.df=var33.cts, dag.m=mydag,verbose=TRUE);

Gaussian node=0 score=-396.151434
Gaussian node=1 score=-395.967082
Gaussian node=2 score=-468.252121
Gaussian node=3 score=-384.496529
Gaussian node=4 score=-334.199036
Gaussian node=5 score=-360.076962
Gaussian node=6 score=-358.952748
Gaussian node=7 score=-450.626527
Gaussian node=8 score=-381.688832
Gaussian node=9 score=-379.543123
Gaussian node=10 score=-379.696223
Gaussian node=11 score=-480.835063
Gaussian node=12 score=-378.705867
Gaussian node=13 score=-458.988366
Gaussian node=14 score=-360.273962
Gaussian node=15 score=-360.706854

   ###################################################################
   ###      log marginal likelihood for Model: -6329.1607281852
   ###################################################################
[1] -6329.161

> # network score for a model with conditional independence
> tographviz(dag=mydag,data.df=var33.cts,outfile="mydagcts.dot");#create file
> # mydag.dot can then be processed with graphviz
> # unix shell "dot -Tpdf mydagcts.dot -o mydagcts.pdf" or use gedit if on Windows
```

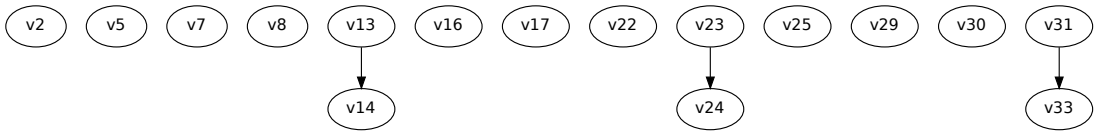

Figure 3: Directed acyclic graph `mydag` for continuous variables only created using `tographviz()` and Graphviz

3.4. Fitting an additive BN model to mixed data

To conclude the fitting of a single pre-specified model to data 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.

```
> mydag<-matrix(c(
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#1
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#2
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#3
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#4
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#5
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#6
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#7
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#8
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#9
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#10
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#11
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#12
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#13
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#14
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#15
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#16
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#17
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#18
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#19
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#20
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#21
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#22
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#23
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#24
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#25
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#26
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#27
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#28
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#29
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#30
```

```
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#31
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#32
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 #33
+                                                    ),byrow=TRUE,ncol=33);
> colnames(mydag)<-rownames(mydag)<-names(var33);#set names
> ## now fit the model defined in mydag - full independence
> fitabn (data.df=var33,dag.m=mydag,verbose=TRUE);


Binary node=0 score=-178.004211
Gaussian node=1 score=-396.151434
Binary node=2 score=-178.414495
Binary node=3 score=-168.248843
Gaussian node=4 score=-395.967082
Binary node=5 score=-102.675919
Gaussian node=6 score=-468.252121
Gaussian node=7 score=-384.496529
Binary node=8 score=-167.644794
Binary node=9 score=-178.679732
Binary node=10 score=-174.534904
Binary node=11 score=-178.293870
Gaussian node=12 score=-334.199036
Gaussian node=13 score=-436.487935
Binary node=14 score=-143.134495
Gaussian node=15 score=-358.952748
Gaussian node=16 score=-450.626527
Binary node=17 score=-173.338554
Binary node=18 score=-174.152823
Binary node=19 score=-177.448401
Binary node=20 score=-177.448401
Gaussian node=21 score=-381.688832
Gaussian node=22 score=-379.543123
Gaussian node=23 score=-460.360575
Gaussian node=24 score=-480.835063
Binary node=25 score=-167.644794
Binary node=26 score=-178.735970
Binary node=27 score=-174.900353
Gaussian node=28 score=-378.705867
Gaussian node=29 score=-458.988366
Gaussian node=30 score=-360.273962
Binary node=31 score=-163.647060
Gaussian node=32 score=-447.594152

   ######################################################################
   ###      log marginal likelihood for Model: -9430.0709700077
   ######################################################################
[1] -9430.071
```

We now fit a BN model which has the same structure as the joint distribution used to generate the data and then create a visual graph of this model

```
> # define a model with many independencies
> mydag[2,1]<-1;
> mydag[4,3]<-1;
> mydag[6,4]<-1; mydag[6,7]<-1;
> mydag[5,6]<-1;
> mydag[7,8]<-1;
> mydag[8,9]<-1;
> mydag[9,10]<-1;
> mydag[11,10]<-1; mydag[11,12]<-1; mydag[11,19]<-1;
> mydag[14,13]<-1;
> mydag[17,16]<-1;mydag[17,20]<-1;
> mydag[15,14]<-1; mydag[15,21]<-1;
> mydag[18,20]<-1;
> mydag[19,20]<-1;
> mydag[21,20]<-1;
> mydag[22,21]<-1;
> mydag[23,21]<-1;
> mydag[24,23]<-1;
> mydag[25,23]<-1; mydag[25,26]<-1;
> mydag[26,20]<-1;
> mydag[33,31]<-1;
> mydag[33,31]<-1;
> mydag[32,21]<-1; mydag[32,31]<-1;mydag[32,29]<-1;
> mydag[30,29]<-1;
> mydag[28,27]<-1; mydag[28,29]<-1;mydag[28,31]<-1;
> fitabn (data.df=var33, dag.m=mydag);

[1] -8646.333

> # network score for a model with conditional independence
> tographviz(dag=mydag,data.df=var33,outfile="mydag_all.dot");#create file
> # mydag.dot can then be processed with graphviz
> # unix shell "dot -Tpdf mydag_all.dot -o mydag_all.pdf" or use gedit if on Windows
```

3.5. Model fitting validation

In order to validate the conjugate models, network scores, for both overall networks and individual nodes using the Bayesian Dirichlet equivalence uniform (BDeu) metric were compared with the `deal` library available from `CRAN`. This metric can be used by `useK2=FALSE` and providing an explicit value for `prior.obs.per.node`. A wide range of models for multinomial data were compared and these were always identical to those values produced by `deal`. To validate the additive models mixed categorical and continuous models, estimates of the posterior distributions for the model parameters using Laplace approximations (see later) were

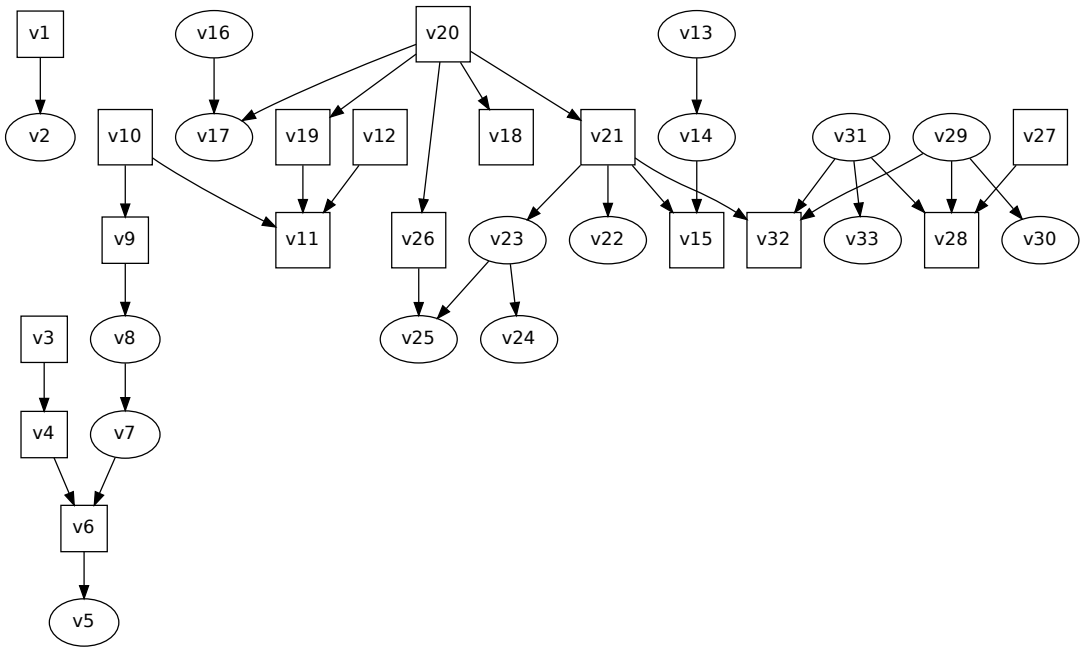

Figure 4: Directed acyclic graph `mydag` for mixed continuous and discrete variables

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. Searching for Optimal Models

A key purpose of BN modeling is to estimate the dependency structure in multivariate data - that is, find a DAG which is robust and representative of the dependency structure of the (unknown) system of processes 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 categorical (conjugate) BN models, then additive models.

4.1. Single search for optimal BN model for categorical data

To run a single search heuristic use `searchbn()` which starts from a randomly chosen DAG (created by randomly adding arcs to an empty network `init.permuts` times) and then searches stepwise 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, sec-

ondly, 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 use ban or retain constraints. 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, and again this includes the initial random network. It is also possible to specific an explicit starting matrix, `start.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 `searchbn()` by default a node cache is used, this is where 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 a lookup table. This can make very significant improvements in speed and the default is for a search to terminate prematurely if the node cache is exceeded, this behaviour can be turned off by `enforce.db.size=FALSE`), but it is generally advisable to use a sufficiently large value for `db.size` to avoid this (a warning will appear to say this limit has been reached if `enforce.db.size=FALSE`).

```
> 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
> mydag<-matrix(c(
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v1
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v3
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v4
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v6
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v9
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v10
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v11
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v12
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v15
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v18
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v19
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v20
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v21
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v26
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v27
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v28
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0  #v32
+                ),byrow=TRUE,ncol=17);
> colnames(mydag)<-rownames(mydag)<-names(var33.cat);#set names
> ## create empty DAGs
> banned.cat<-matrix(rep(0,dim(var33.cat)[2]^2),ncol=dim(var33.cat)[2]);
> colnames(banned.cat)<-rownames(banned.cat)<-names(var33.cat);#set names
> retain.cat<-matrix(rep(0,dim(var33.cat)[2]^2),ncol=dim(var33.cat)[2]);
> colnames(retain.cat)<-rownames(retain.cat)<-names(var33.cat);#set names
> start.cat<-matrix(rep(0,dim(var33.cat)[2]^2),ncol=dim(var33.cat)[2]);
> colnames(start.cat)<-rownames(start.cat)<-names(var33.cat);#set names
> myres<-searchbn(data.df=var33.cat,
```

```
+                  banned.m=banned.cat,
+                  retain.m=retain.cat,
+                  start.m=start.cat,
+                  useK2=TRUE,max.parents=2,init.permuts=0,db.size=1000);
```

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

To run a single search heuristic for an additive BN use `searchabn()` which is very similar to `searchbn()`, the main difference is in the parameter prior specifications. It only makes sense to use uninformative priors for the parameters for each variable as these are fixed for all DAG structures e.g. parameter priors are not structure specific. By default diffuse priors of Gaussians of mean zero and precision of 1000 are used, where these parameters are for the usual additive terms in a logistic regression. These can be overridden in the command line arguments if desired (not advisable). 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 this model is simply assigned a log network score of $-\infty$ which effectively removes it from the model search.

```
> ## just use default priors
> myres.add<-searchabn(data.df=var33.cat,
+                  banned.m=banned.cat,
+                  retain.m=retain.cat,
+                  start.m=start.cat,
+                  max.parents=2,
+                  init.permuts=0,db.size=1000,error.verbose=TRUE);
```

4.3. Single search for optimal BN model for continuous data

As above but for a network of Gaussian nodes.

```
> var33.cts<-var33[,-bin.nodes];#drop categorical nodes
> mydag<-matrix(c(
+                  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v2
+                  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v5
+                  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v7
+                  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v8
+                  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v13
+                  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v14
+                  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v16
+                  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v17
+                  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v22
+                  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v23
+                  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v24
+                  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v25
```

```
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v29
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v30
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v31
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0  #v33
+                ),byrow=TRUE,ncol=16);
> colnames(mydag)<-rownames(mydag)<-names(var33.cts);#set names
> banned.cts<-matrix(rep(0,dim(var33.cts)[2]^2),ncol=dim(var33.cts)[2]);
> colnames(banned.cts)<-rownames(banned.cts)<-names(var33.cts);#set names
> retain.cts<-matrix(rep(0,dim(var33.cts)[2]^2),ncol=dim(var33.cts)[2]);
> colnames(retain.cts)<-rownames(retain.cts)<-names(var33.cts);#set names
> start.cts<-matrix(rep(0,dim(var33.cts)[2]^2),ncol=dim(var33.cts)[2]);
> colnames(start.cts)<-rownames(start.cts)<-names(var33.cts);#set names
> #
> myres.add<-searchabn(data.df=var33.cts,
+                   banned.m=banned.cts,
+                   retain.m=retain.cts,
+                   start.m=start.cts,
+                   max.parents=2,
+                   init.permuts=0,db.size=1000,error.verbose=TRUE);

initial network: (log) network score = -6573.123350
search iteration...1 new score=-6486.236053
search iteration...2 new score=-6401.043802
search iteration...3 new score=-6320.379450
search iteration...4 new score=-6242.083760
search iteration...5 new score=-6165.672787
search iteration...6 new score=-6097.414657
search iteration...7 new score=-6045.701536
search iteration...8 new score=-6044.623774
```

4.4. 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.

```
> mydag<-matrix(c(
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#1
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#2
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#3
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#4
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#5
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#6
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#7
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#8
```

```
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#9
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#10
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#11
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#12
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#13
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#14
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#15
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#16
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#17
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#18
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#19
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#20
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#21
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#22
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#23
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#24
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#25
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#26
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#27
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#28
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#29
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#30
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#31
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#32
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 #33
+                                                  ),byrow=TRUE,ncol=33);
> colnames(mydag)<-rownames(mydag)<-names(var33);#set names
> ## create empty DAGs
> banned<-matrix(rep(0,dim(var33)[2]^2),ncol=dim(var33)[2]);
> colnames(banned)<-rownames(banned)<-names(var33);#set names
> retain<-matrix(rep(0,dim(var33)[2]^2),ncol=dim(var33)[2]);
> colnames(retain)<-rownames(retain)<-names(var33);#set names
> start<-matrix(rep(0,dim(var33)[2]^2),ncol=dim(var33)[2]);
> colnames(start)<-rownames(start)<-names(var33);#set names
> ## giving diffuse priors - same as default but explicitly stated
> myres.add<-searchabn(data.df=var33,
+                  banned.m=banned,
+                  retain.m=retain,
+                  start.m=start,
+                  hyper.params=list(
+                                   mean=rep(0,dim(var33)[2]+1),
+                                   sd=rep(sqrt(1000),dim(var33)[2]+1),
+                                   shape=rep(0.001,16),## 16 Gaussian nodes
+                                   scale=rep(1/0.001,16)## 16 Gaussian nodes
+                  ),
+                  max.parents=2,
+                  init.permuts=0,db.size=10000,
```

```
+                   error.verbose=TRUE,enforce.db.size=TRUE);
```

5. Multiple Search Strategies

To estimate a robust BN or additive BN for a given dataset is it necessary to run many searches and then summarize the results of these searches. The functions `hillsearchbn()` and `hillsearchabn()` are similar `searchbn()` and `searchabn()` but run multiple searches. There are some differences. Firstly, it is necessary to provide a list of starting networks - these can all be the null network - but must be explicitly given, and must have the same number of entries as the number of searches requested. Secondly, these functions also use a node cache for speed and there is the option now to either use a single joint node cache over all searches, or else use a local node cache which is reset to empty at the start of each new search. The parameter which governs this is `localdb` and defaults to true which resets the cache at the start of each new search.

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.

It is suggested to use `localdb=FALSE` with some caution as it may lead to computations, the duration of which, is hard to estimate as each successive search becomes slower as the node cache increases. To help with performance monitoring it is possible to turn on timings using `timing=TRUE` which then outputs the number of seconds of CPU time each individual search takes (using standard libc functions declared in time.h).

```
> mydag<-matrix(c(
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#1
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#2
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#3
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#4
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#5
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#6
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#7
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#8
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#9
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#10
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#11
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#12
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#13
```

```
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#14
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#15
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#16
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#17
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#18
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#19
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#20
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#21
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#22
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#23
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#24
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#25
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#26
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#27
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#28
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#29
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#30
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#31
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#32
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 #33
+                                                    ),byrow=TRUE,ncol=33);
> colnames(mydag)<-rownames(mydag)<-names(var33);#set names
> ## create empty DAGs
> banned<-matrix(rep(0,dim(var33)[2]^2),ncol=dim(var33)[2]);
> colnames(banned)<-rownames(banned)<-names(var33);#set names
> retain<-matrix(rep(0,dim(var33)[2]^2),ncol=dim(var33)[2]);
> colnames(retain)<-rownames(retain)<-names(var33);#set names
> start<-matrix(rep(0,dim(var33)[2]^2),ncol=dim(var33)[2]);
> colnames(start)<-rownames(start)<-names(var33);#set names
> set.seed(10000);## only affects init.permuts
> start.list<-list();
> n.searches<-10;#example only - must be much larger in practice
> for(i in 1:n.searches){start.list[[i]]<-retain;} ## empty networks
> myres<-hillsearchabn(data.df=var33,banned.m=banned,retain.m=retain,
+                      start.m=start.list,
+                      hyper.params=list(
+                      mean=rep(0,dim(var33)[2]+1),
+                      sd=rep(sqrt(1000),dim(var33)[2]+1),
+                      shape=rep(0.001,16),## 16 Gaussian nodes
+                      scale=rep(1/0.001,16)## 16 Gaussian nodes
+                      ),
+                      max.parents=2,
+                      num.searches=n.searches,
+                      init.permuts=20,db.size=20000,
+                      localdb=TRUE,timing=TRUE);
>
```

5.1. Creating a Summary Network - Majority Consensus

One approach to producing a single robust BN model of the data is to mimic the approach used in phylogenetics to create majority consensus trees. A DAG is constructed comprising of all the arcs present in more than 50% of the DAGs found from the search heuristics, that is all the locally optimal models found are combined into a single summary network. Combining results from different runs of `searchbn()` or `searchabn()` is straightforward, although note that it is necessary to check for duplicate random starting networks (unlikely generally but not impossible). The following code provides a simple way to produce a majority consensus network and Figure 5 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.

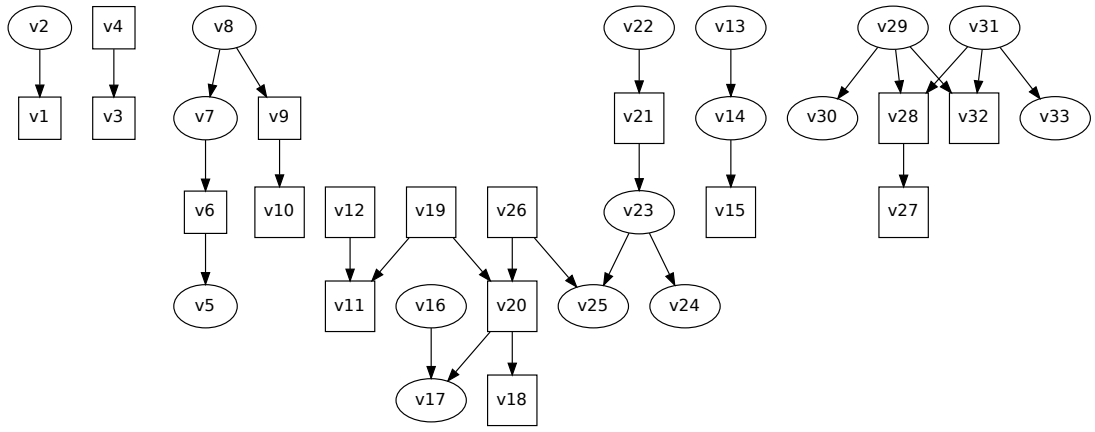

Figure 5: Example majority consensus network (from the results of only 10 searches)

```
> # use results from above searches which are stored in ``myres''
> #step 1. discard any duplicate searches (these are unlikely)
> indexes<-uniquenets(myres$init.dag);
> all.res<-list();
> all.res$init.score<-myres$init.score[indexes];
> all.res$final.score<-myres$final.score[indexes];
> all.res$init.dag.<-myres$init.dag[indexes];
> all.res$final.dag<-myres$final.dag[indexes];
> # for every possible arc calculate how many times it appears in the searches
> mypruneddags<-prunenets(all.res$final.dag,round(0.51*length(all.res$final.dag)));
> # now get a matrix/DAG for the majority network comprising of 1/0s
> myfunc<-function(arg1,threshold,netdata){#trivial helper for apply()
+          if(arg1>=round(threshold*length(netdata$final.dag)))
+          {return(1);} else {return(0);}}
> dag.con<-apply(mypruneddags$arcs.sum,c(1,2),FUN=myfunc,threshold=0.51,
```

```
+                netdata=all.res);
> tographviz(dag=dag.con,data.df=var33,outfile="dagcon.dot");#create file
> # dagcon.dot can then be processed with graphviz
> # unix shell "dot -Tpdf dagcon.dot -o dagcon.pdf" or use gedit if on Windows
>
```

6. Larger Scale Problems

Determining an optimal DAG (e.g. a majority consensus model) for data sets with larger numbers of variables can require additional ad-hoc measures beyond limiting the number number of parents per node. One approach is to iteratively limit the search space by adaptively adjusting the ban and retain lists. Initially set the parent limit to one and conduct a large number of searches, and check that the majority consensus network is robust. Next, for those variables in the majority consensus network which had (strictly) less than the current number of parent limit arcs (e.g. one) append these to the "ban list" so that future searches will not again consider adding further arcs to these variables. Then create a "retain list" containing all those arcs in the current majority consensus network so that in all subsequent searches those arcs that are currently supported will be retained automatically through all searches (and set the start network to the retain network - plus some random permutations). Now increase the parent limit (e.g. to two) and repeat the whole procedure. This approach successively limits the search space finally narrowing to a summary majority consensus network, although a number of iterations might be needed. Each iteration of this approach can still take many hours on a multi-processor compute server depending on the number of variables in the data.

7. Estimating Posterior Densities

After determining an appropriate BN model it is typical to estimate the parameter effects, e.g. 95% posterior intervals, which is done using the function `getmarginal()` which again uses Laplace approximations. An appropriate domain (range) for each parameter needs to be supplied which can either be done by some trial and error, searching for where the mode of the posterior density is, or else by using the classic `glm()` function in `R` to provide an approximate range. Given below are three examples, and note that this function works with one node and one parameter in that node at a time.

```
> #specific a DAG model
> mydag<-matrix(c(
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#1
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#2
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#3
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#4
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#5
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#6
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#7
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#8
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#9
```

```
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#10
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#11
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#12
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#13
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#14
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#15
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#16
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#17
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#18
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#19
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#20
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#21
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#22
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#23
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#24
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#25
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#26
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#27
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#28
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#29
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#30
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#31
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,#32
+ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 #33
+                                                    ),byrow=TRUE,ncol=33);
> colnames(mydag)<-rownames(mydag)<-names(var33);#set names
> ## now fit the model defined in mydag - full independence
> # define a model with many independencies
> mydag[2,1]<-1;
> mydag[4,3]<-1;
> mydag[6,4]<-1; mydag[6,7]<-1;
> mydag[5,6]<-1;
> mydag[7,8]<-1;
> mydag[8,9]<-1;
> mydag[9,10]<-1;
> mydag[11,10]<-1; mydag[11,12]<-1; mydag[11,19]<-1;
> mydag[14,13]<-1;
> mydag[17,16]<-1;mydag[17,20]<-1;
> mydag[15,14]<-1; mydag[15,21]<-1;
> mydag[18,20]<-1;
> mydag[19,20]<-1;
> mydag[21,20]<-1;
> mydag[22,21]<-1;
> mydag[23,21]<-1;
> mydag[24,23]<-1;
> mydag[25,23]<-1; mydag[25,26]<-1;
> mydag[26,20]<-1;
```

```
> mydag[33,31]<-1;
> mydag[33,31]<-1;
> mydag[32,21]<-1; mydag[32,31]<-1;mydag[32,29]<-1;
> mydag[30,29]<-1;
> mydag[28,27]<-1; mydag[28,29]<-1;mydag[28,31]<-1;

> #now get the marginal distribution for the parameters of v33
> marg1<-getmarginal(data.df=var33,
+                     dag.m=mydag,
+                     whichnode="v33",
+                     whichvar="constant",#this is the intercept
+                      hyper.params=list(
+                       mean=rep(0,dim(var33)[2]+1),
+                       sd=rep(sqrt(1000),dim(var33)[2]+1),
+                       shape=rep(0.001,16),## 16 Gaussian nodes
+                       scale=rep(1/0.001,16)## 16 Gaussian nodes
+                       ),
+                       post.x=seq(from=-1.5,to=-0.5,len=1000),
+                       verbose=TRUE);

Gaussian node=32 score=-360.706854

> cum.marg<-cumsum(marg1[,"f"])/sum(marg1[,"f"])
> marg<-cbind(marg1,cum.marg);
> marg[which(marg[,3]>0.5)[1]];#approx.median

[1] -1.071572

> ## now for the precision at node v33
> marg2<-getmarginal(data.df=var33,
+                     dag.m=mydag,
+                     whichnode="v33",
+                     whichvar="precision",#this is the intercept
+                      hyper.params=list(
+                       mean=rep(0,dim(var33)[2]+1),
+                       sd=rep(sqrt(1000),dim(var33)[2]+1),
+                       shape=rep(0.001,16),## 16 Gaussian nodes
+                       scale=rep(1/0.001,16)## 16 Gaussian nodes
+                       ),
+                       post.x=seq(from=0.5,to=1.5,len=1000),
+                       verbose=TRUE);

Gaussian node=32 score=-360.706854

> cum.marg<-cumsum(marg2[,"f"])/sum(marg2[,"f"])
> marg<-cbind(marg2,cum.marg);
> marg[which(marg[,3]>0.5)[1]];#approx.median
```

```
[1] 1.114615

> ## now for a covariate effect at node v6
> marg3<-getmarginal(data.df=var33,
+                    dag.m=mydag,
+                    whichnode="v6",
+                    whichvar="v4",#this is the covariate effect
+                     hyper.params=list(
+                      mean=rep(0,dim(var33)[2]+1),
+                      sd=rep(sqrt(1000),dim(var33)[2]+1),
+                      shape=rep(0.001,16),## 16 Gaussian nodes
+                      scale=rep(1/0.001,16)## 16 Gaussian nodes
+                      ),
+                      post.x=seq(from=-1.5,to=2.5,len=1000),
+                      verbose=TRUE);

Binary node=5 score=-87.882271

> cum.marg<-cumsum(marg3[,"f"])/sum(marg3[,"f"])
> marg<-cbind(marg3,cum.marg);
> marg[which(marg[,3]>0.5)[1]];#approx.median

[1] 0.5900901
```

Figure 6 shows an example of posterior densities estimated using `getmarginal()`, all posterior densities for all parameters in the additive BN can be estimated in the same way.

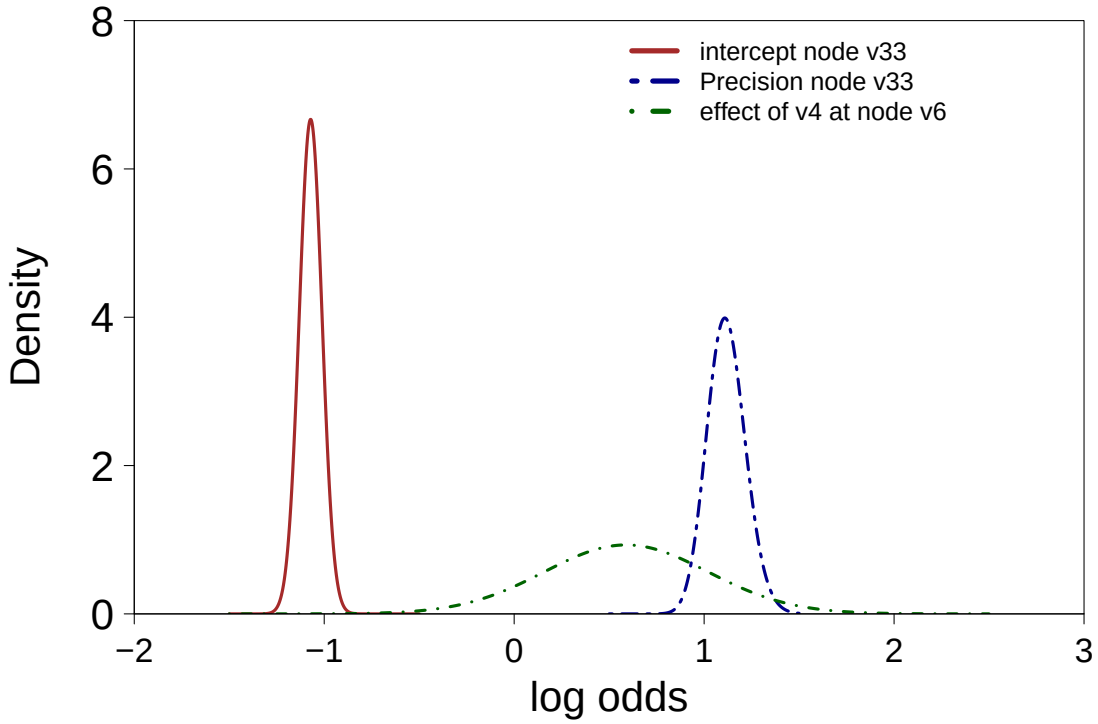

Figure 6: Some Posterior densities

References

Buntine W (1991). "Theory refinement on Bayesian networks." In *Proceedings of Seventh Conference on Uncertainty in Artificial Intelligence*, pp. 52–60. Morgan Kaufmann, Los Angeles, CA, USA.

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.

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 D, Xiang Z, Woolf P, Xi C, He Y (2010). "Bayesian Network Expansion Identifies New ROS and Biofilm Regulators." *PLoS ONE*, **5**(3), e9513–.

Jensen FV (2001). *Bayesian Network and Decision Graphs*. Springer-Verlag, New York.

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

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. doi:10.1016/j.prevetmed.2011.02.003.

Needham CJ, Bradford JR, Bulpitt AJ, Westhead DR (2007). "A Primer on Learning in Bayesian Networks for Computational Biology." *PLoS Comput Biol*, **3**(8), e129. doi:10.1371/journal.pcbi.0030129.

Poon AFY, Lewis FI, Pond SLK, Frost SDW (2007). "Evolutionary interactions between N-linked glycosylation sites in the HIV-1 envelope." *Plos Computational Biology*, **3**(1), 110–119.

Affiliation:

F. I. Lewis
Applied Statistician
Vetsuisse Faculty, University of Zurich
Winterthurerstrasse 270, Zurich 8057
Switzerland
E-mail: fraseriain.lewis@uzh.ch