Dependency detection with Bayesian Networks

M V Vikhreva

Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University, Leninskie Gory, Moscow, 119991

Supervisor: A G Dyakonov

E-mail: vkhvmr@gmail.com

1. Introduction

Most methods aimed to detect dependency between two variables do not consider interventions with numerous other variables and often make groundless assumptions about relation form. Bayesian networks help to despise of the drawbacks. Bayesian networks allow one to learn about causal relationships. Bayesian networks can consider prior knowledge. Bayesian networks can readily handle incomplete data sets.

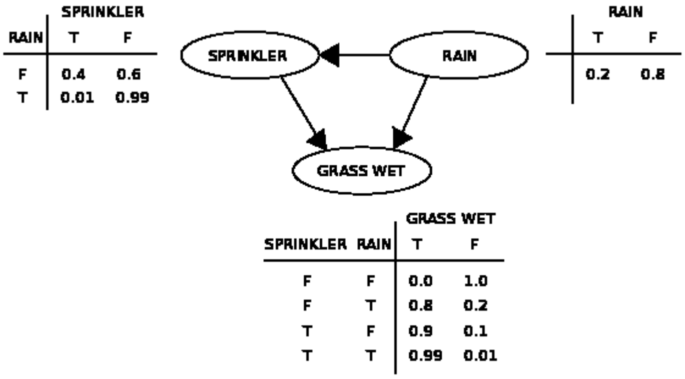
A Bayesian network is a probabilistic graphical model (a type of statistical model) that represents a set of random variables and their conditional dependencies via a directed acyclic graph (DAG). For a long time Bayesian networks were used as a representation for encoding uncertain expert knowledge in expert systems. Recently, methods were developed to learn Bayesian network from data. Nowadays data analysis bring up a problem of learning a Bayesian network structure. This work is dedicated to finding the scope of data dependencies modern applied algorithms can detect with Bayesian networks.

In section 2, there are a few more words about what a Bayesian network is. In section 3, we present a Python package pebl and discuss pebl implemented tools for structure learning and probabilities learning in a fixed Bayesian-network structure. In section 4, testing of pebl tools and experiments on how well pebl detects dependencies are presented.

2. Bayesian Network graphical model

Formally, Bayesian networks are DAGs whose nodes represent random variables. Edges connect causes and effects and variables that are conditionally independent correspond to nodes that are not connected. Each node is associated with a probability function that takes, as input, a particular set of values for the node's parent variables, and gives the probability distribution of the variable represented by the node. For example, if m parent nodes represent m Boolean variables then the probability function could be represented by a table of 2^m entries, one entry for each of the 2^m possible combinations of its parents being true or false (see example on Figure 1).

In order to fully specify the Bayesian network it is necessary to define the joint probability distribution of the statistical model, thus to specify for each node X the probability distribution for X conditional upon X's parents. It may have any form. In this work discrete distribution is used since it simplifies calculations. In a situation when only constraints on a distribution are known, the principle of maximum entropy can help to determine the distribution. It will be a distribution that both maximizes entropy and meet the constraints.



	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99

RAIN	
T	F
0.2	0.8

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

Figure 1. A simple Bayesian network and its conditional probability tables. The joint probability is $\mathrm{P}(G, S, R) = \mathrm{P}(G|S, R)\mathrm{P}(S|R)\mathrm{P}(R)$, where G correspond to Grass Wet variable, S – Sprinkler, R – Rain.

Often these conditional distributions include parameters which are unknown and must be estimated from data, sometimes using the maximum likelihood approach. Direct maximization of the likelihood (or of the posterior probability) is often complex when there are unobserved variables. A classical approach to this problem is the expectation-maximization algorithm which alternates computing expected values of the unobserved variables conditional on observed data, with maximizing the complete likelihood (or posterior) assuming that previously computed expected values are correct.

In the simplest case, a Bayesian network is specified by an expert and is then used to perform inference. But commonly the task of defining the network is too complex for humans. Then the network structure and the parameters of the local distributions must be learned from data.

The common approach to structural learning is to introduce a statistically motivated scoring function that evaluates each network with respect to the training data and to search for the optimal network according to this score. For a scoring function posterior probability of the structure given the training data can be used. The exhaustive maximization of the score is known to be an NP-hard problem. Thus heuristic search is often used.

3. Pebl library

In this paper, we introduce pebl, a Python library and application for learning Bayesian network structure from data. Although learning the structure of BNs from data is now common, there are still a few high-quality open-source softwares that can meet the needs of various users. End users require software that is easy to use; supports learning with different data types; can accommodate missing values and hidden variables; and can take advantage of various computational clusters and grids. Pebl is the Python Environment for Bayesian Learning to meet these needs.

3.1. Functionality

pebl:

- can work with continuous, discrete and categorical variables (with different type variables simultaneously);
- can perform maximum entropy discretization (that is when single variable values are compressed into fixed number of bins according to maximum entropy criterion);
- uses the BDe metric for scoring networks and handles interventional data using the method described by Cooper and Yoo (2002);
- can handle missing values and hidden variables using exact marginalization and Gibbs sampling (Heckerman, 1998);

	BANJO	BNT	Causal Explorer	Deal	LibB	PEBL
License	Academic	GPL	Academic	GPL	Academic	MIT
Scripting language	Matlab	Matlab	Matlab	R	N/A	Python
Interventional Data	No	Yes	No	No	No	Yes
Dynamic BN	Yes	Yes	No	No	No	No
Structural Priors	Yes	No	No	No	No	Yes
Missing Data	No	Yes	No	No	Yes	Yes
Parallel Execution	No	No	No	No	No	Yes

Table 1. Popular Bayesian network structure learning software.

- supports structural priors over edges specified as hard constraints or soft energy matrices (Imoto et al., 2003) and arbitrary constraints specified as Python functions or lambda expressions;
- includes greedy hill-climbing and simulated annealing learners and makes writing custom learners easy.

3.2. pebl interface use

- In the input data file type of variables should be specified;
- Apply `pebl.data.discretize(num_bins)` to compress unique values of continuous or discrete variables to `num_bins` number of values (so pebl sees continuous variables as discrete);
- Specify network structure learners with `pebl.greedy(max_iter)` and `[pebl.simanneal(max_iter)`;
- For parallel calculations use `pebl.multiprocess`;
- `pebl.result.post[:10]` is made to access top score network configurations and `pebl.result.cons` to access its consensus matrix C (where C_{ij} is a p-value referring to an edge from i-th to j-th node).

3.3. Convenience and Scalability

While many tasks related to Bayesian learning are embarrassingly parallel in theory, few software packages take advantage of it. pebl can execute learning tasks in parallel over multiple processors or CPU cores, an Apple Xgrid1, an IPython cluster2 or the Amazon EC2 platform3.

With appropriate configuration settings and the use of parallel execution, pebl can be used for large learning tasks. Although pebl has been tested successfully with datasets with 10000 variables and samples, BN structure learning is a known NP-Hard problem (Chickering et al., 1994) and analysis using datasets with more than a few hundred variables is likely to result in poor results due to poor coverage of the search space.

3.4. Related Software Comparison

While there are many software tools for working with BNs, most focus on parameter learning and inference rather than structure learning. Of the few tools for structure learning, few are open-source and none provide the set of features included in pebl. As shown in Table 1, the ability to handle interventional data, model with missing values and hidden variables, use soft and arbitrary priors and exploit parallel platforms are unique to pebl. pebl, however, does not currently provide any features for inference or learning Dynamic Bayesian Networks (DBN). Despite its use of optimized matrix libraries and custom C extension modules, pebl can be an order of magnitude or more slower than software written in Java or C/C++; the ability to use a wider range of data and priors, the parallel processing features and the ease-of-use, however, should make it an attractive option for many users.

Feature	Name	Values
main feature	x	$\mathcal{N}(2, 10)$
dependency	f(x)	f(x)
independable feature	y	$\mathcal{N}(2, 10)$

Table 2. Features to detect dependency $x \to f(x)$.

4. Experiments

In the section there are different tests of pebl dependency detection abilities. To check if a program detects dependency or not we use as input features reflected in Table 2. Since an output of pebl learner consists of top scoring networks and a consensus matrix, we use minimum p-value of edges we expect to be present and maximum p-value of all edges connecting independent features to verify algorithm answer. Also Bayesian network do not comprise cycles, so dependencies like linear relation can be represented with two structures (see Figure 2).

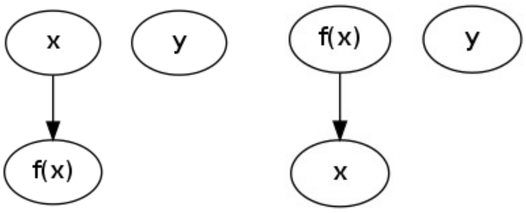


Figure 2. For most functions f(x) these are equivalent structures.

Maximum entropy discretization plays an important role in dependency detection. Given x, y and $f(x + n_1) + n_2$, $n_1 \sim \mathcal{N}(0, 0.2)$, $n_2 \sim \mathcal{N}(0, 0.1)$ we variate `num_bins` and check how p-values of "true" edges (those connecting dependable features) change. Our dataset consists of 100 objects. On Figure 3 p-values of "true" edges exceed p-values of "false" edges (those connecting independable features) in the (35; 38) and (50, 75) `num_bins` intervals, that is where dependency $x \to f(x)$ is detected. "65" value seems the most certain.

How to detect optimal `num_bins` value without knowing true edges?

- On Figure 4 top structures' mean entropy is reflected. The entropy is defined as

$$\frac{1}{|\{G\}|} \sum_{G \text{ - } top\ score\ network} -score(G) \cdot log(score(G)).$$

  Then the entropy is the bigger the more confident edges are. Considering Figures 3 and 4, we conclude that the entropy is zero in (50, 75) interval, where false edges are not included in networks. It also becomes zero in other regions, but the graphic is significantly unstable there.
- We can also consider "false" edges p-values to choose `num_bins` parameter.

Finally we noise data with normal distributed variable to inspect limits of pebl linear dependency detection ability. The dependency is recognised quite well (Figure 5), see example of noised data in Figure 6.

Polynomial dependency detection is reflected on Figures 7, 8.

5. Conclusion

We introduced Bayesian network library pebl, reviewed it's implemented tools and made several suggestions on how to work with pebl. Experiments show that it works much better than a man's eye. With the use of parallel calculations Bayesian networks structure-learners can be a powerful dependency detection tool.

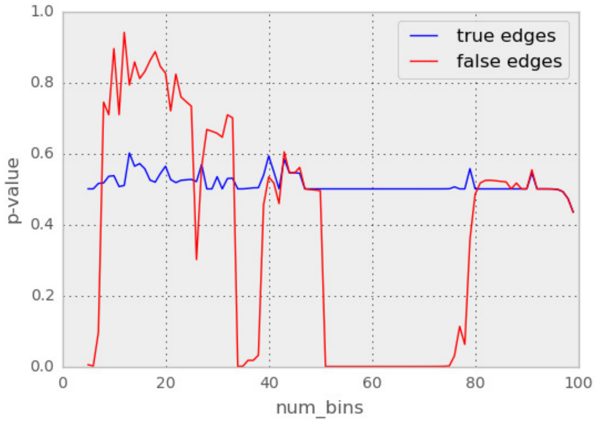


Figure 3. Edges p-values (varying num_bins) for linear dependency dataset.

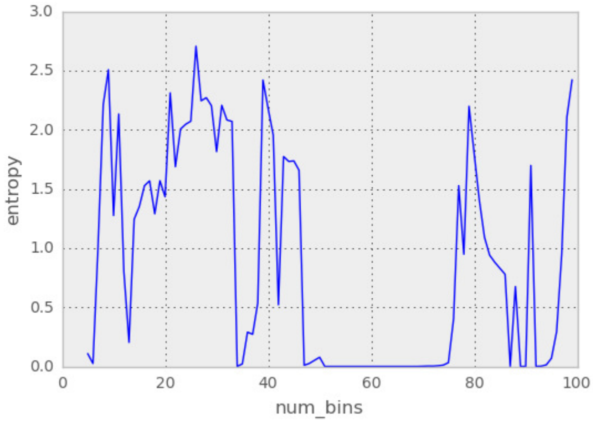


Figure 4. Top network's mean entropy for linear dependency dataset.

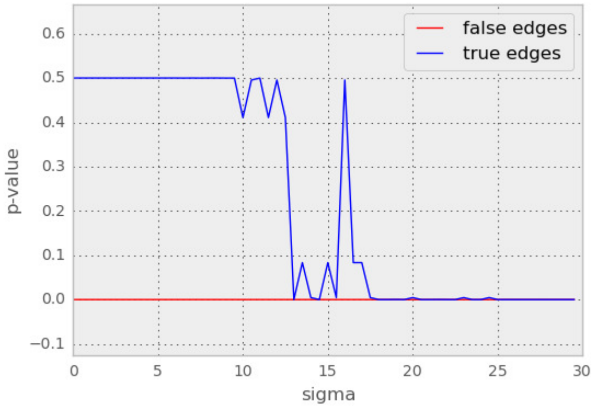


Figure 5. Edges p-values (varying noise∼ $\mathcal{N}(0, sigma)$) for linear dependency dataset.

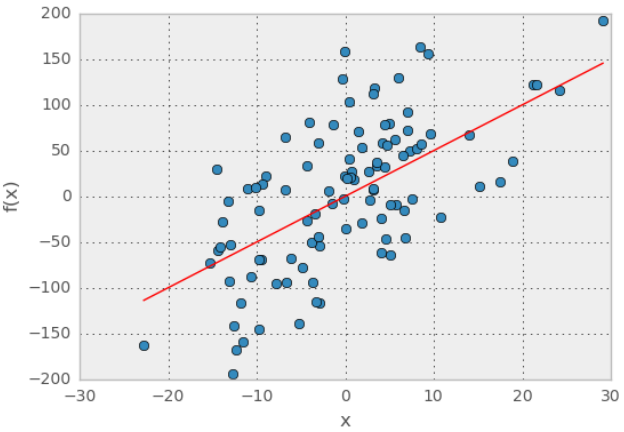


Figure 6. Linear dataset with noise sigma=10.

References

[1] Heckerman D 1998 *A tutorial on learning with bayesian networks.* The MIT Press

[2] Abhik Shah, Peter Woolf 2009 *Python Environment for Bayesian Learning: Inferring the Structure of Bayesian Networks from Knowledge and Data* JMLR

[3] Peer D, Regev A, Elidan G, Friedman N 2001 *Inferring subnetworks from perturbed expression profiles* Bioinformatics

[4] Yoo C, Thorsson V, Cooper GF 2002 *Discovery of causal relationships in a gene-regulation pathway from a mixture of experimental and observational DNA microarray data* Pac Symp Biocomput

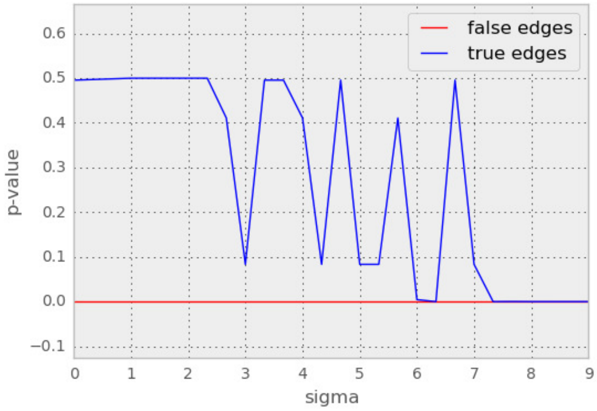


Figure 7. Edges p-values (varying noise∼ $\mathcal{N}(0, sigma)$) for polynomial dependency dataset.

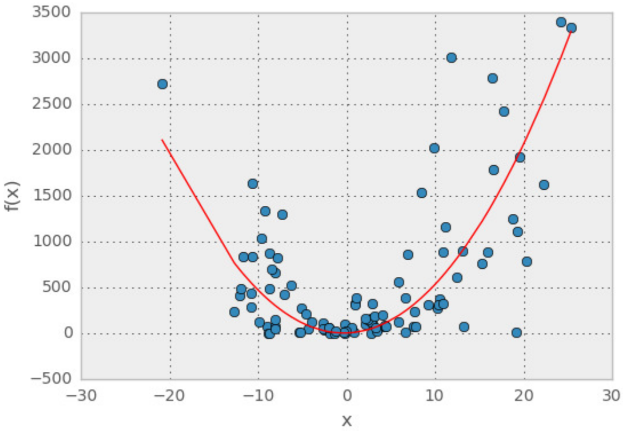


Figure 8. Polynomial dataset with noise sigma=5.