Improving the SLA algorithm using association rules

Evelina Lamma[1], Fabrizio Riguzzi[1], Andrea Stambazzi[1], Sergio Storari[1,2]

[1]University of Ferrara, Department of Engineering, via Saragat 1, 44100 Ferrara, Italy
[2]University of Bologna, DEIS, viale Risorgimento 2, 40136 Bologna, Italy
{elamma, friguzzi, astambazzi, sstorari}@ing.unife.it

Abstract. A bayesian network is an appropriate tool for working with uncertainty and probability, that are typical of real-life applications. In literature we find different approaches for bayesian network learning. Some of them are based on search and score methodology and the others follow an information theory based approach. One of the most known algorithm for learning bayesian network is the SLA algorithm. This algorithm constructs a bayesian network by analyzing conditional independence relationships among nodes. The SLA algorithm has three phases: drafting, thickening and thinning. In this work, we propose an alternative method for performing the drafting phase. This new methodology uses data mining techniques, and in particular the computation of a number of parameters usually defined in relation to association rules, in order to learn an initial structure of a bayesian network. In this paper, we present the BNL-rules algorithm (Bayesian Network Learner with association rules) that exploits a number of association rules parameters to infer the structure of a bayesian network. We will also present the comparisons between SLA and BNL-rules algorithms on learning four bayesian networks.

1 Introduction

A bayesian network is an appropriate tool for working with uncertainty and probability, that are typical of real-life applications. A bayesian network is a directed, acyclic graph (DAG) whose nodes represent random variables. In bayesian networks each node, V, is conditionally independent of any subset of the nodes that are not its descendants given its parents. Bayesian networks are sometimes called causal networks because the arcs connecting the nodes can be thought of as representing direct causal relationships. Building bayesian networks on the basis of the intuitive (human) notion of causality usually results in networks that respect the conditional independence assumptions. According to [12] "… to construct a bayesian network for a given set of variables, we draw arcs from cause variables to immediate effects. In almost all cases, doing so results in a Bayesian network whose conditional-independence implications are accurate".

By means of bayesian networks, we can use information about the values of some variables to obtain probabilities for the values of others. A probabilistic inference takes place, once the probabilities functions of each node conditioned to just its parents are given. These are usually represented in a tabled form, named conditional probability tables (CPTs).

Learning techniques have been extensively applied (see, for instance [12]) in Bayesian networks. Given a training set of examples, learning such a network is the problem of finding the structure of the direct acyclic graph and the CPTs associated with each node in the DAG that best matches (according to some scoring metric) this dataset. Various scoring metrics have been proposed (e.g., description length or posterior probability [2,17,20,10,12,13,14,25]). Learning algorithms perform a search among possible network structures, however the search space is so vast, that any kind of exhaustive search can not be considered, and often a greedy approach is followed.

One of the most known algorithm for learning bayesian network is the SLA algorithm presented in [8]. This algorithm constructs a bayesian network by analyzing conditional independence relationships among nodes. It computes the mutual information of each pair of nodes as a measure of dependency, and creates the network using this information. The SLA algorithm has three phases: drafting, thickening and thinning. The first phase of this algorithm is essentially Chow and Liu's tree construction algorithm [9]; the second and the third phase are designed to extend tree construction to general bayesian network construction. The draft is a singly connected undirected graph (an undirected graph without loops). In a special case when the bayesian network is a tree or polytree, this phase can construct the network correctly and the second and third phase will not change anything.

In this work, we propose another way for performing the drafting phase. This new methodology uses data mining techniques, and in particular the computation of a number of association rules parameters given a database of examples, in order to learn the structure of a bayesian network. Association rules describe correlation of events, and can be viewed as probabilistic rules. Two events are "correlated" if they are frequently observed together. Each association rule, is characterized by several parameters which can be used in structure learning. In this paper, we present the BNL-rules algorithm (Bayesian Network Learner with association rules) that exploits these parameters to infer the structure of a bayesian network.

Section 2 discusses the conditional independency test used by the SLA algorithm. Section 3 describes the SLA algorithm itself. In Section 4, we present association rules parameters. In Section 5 we introduce the algorithm BNL-rules. In Section 6, we will present the comparisons between SLA and BNL-rules algorithms considering four of the most well-known bayesian networks. Finally, in Section 7, we conclude, and present future work.

2 An Information Theory Based Approach

In literature we find different approaches for bayesian network learning. Some of them are based on the search and score methodology [2,17,20,10,12,14,25], and the others follow an information theory based approach [8,21]. The SLA algorithm constructs bayesian networks by analyzing conditional independence relationships among nodes, following the second approach. Before introducing the SLA algorithm, we first recall the concepts of *d-separation* [17] and mutual information [8], which play an important role in this algorithm.

For any three disjoint node sets *X*, *Y*, and *Z* in a bayesian network, *X* is said to be *d-separated* from *Y* by *Z* if there is no *active adjacency* path between *X* and *Y* given *Z*. An *adjacency* path is a path between two nodes without considering the directionality of the arcs. An *adjacency* path between *X* and *Y* is *active* given *Z* if:

- every *collider* in the path is in *Z* or has a descendant in *Z*;
- every other node in the path is outside *Z*.

A *collider* [23] of a path is a node where two arcs in the path meet at their endpoints. In a bayesian network, if there is an arc from *a* to *b*, we say that *a* is a parent of *b* and *b* is a child of *a*. We also say that *a* is in the neighborhood of *b* and *b* is in the neighborhood of *a*. If there is a path from node *a* to node *b*, then *a* is an ancestor of *b* and *b* is a descendant of *a*.

To understand *d-Separation*, we can use an analogy, which is similar to the one suggested in [24]. We view a bayesian network as a network system of information channels, where each node is a valve that is either active or inactive and the valves are connected by noisy information channels. The information flow can pass an active valve but not an inactive one. When all the valves (nodes) on one *adjacency* path between two nodes are active, we say this path is **open**. If any one valve in the path is inactive, we say the path is **closed**. When all paths between two nodes are **closed** given the statuses of a set of valves (nodes), we say the two nodes are *d-separated* by the set of nodes. The statuses of valves can be changed through the instantiation of a set of nodes.

The amount of information flow between two nodes can be measured by using mutual information when no nodes are instantiated, or conditional mutual information when some other nodes are instantiated. In information theory, the mutual information of two nodes is defined as:

$$I(X_i, X_j) = \sum_{x_i, x_j} P(x_i, x_j) \log \frac{P(x_i, x_j)}{P(x_i) P(x_j)} \tag{1}$$

where X_i and X_j are two nodes and x_i and x_j are possible values for X_i and X_j. The conditional mutual information is defined as

$$I(X_i, X_j \mid C) = \sum_{x_i, x_j, c} P(x_i, x_j, c) \log \frac{P(x_i, x_j \mid c)}{P(x_i \mid c) P(x_j \mid c)} \tag{2}$$

where *C* is a set of nodes and *c* is a possible assignment of all the nodes in *C*. In the SLA algorithm, the conditional mutual information measures the average information flow between two nodes when the statuses of some valves are changed by the condition-set *C*. When equation (2) is smaller than a certain threshold value ε, we say that X_i, X_j are *d-separated* by the condition-set *C*, and that they are conditionally independent. Thus, the mutual information is a measure of Conditional Independence (CI). The use of mutual information in probabilistic model construction can be traced back to Chow and Liu's tree construction algorithm [9]. In 1987, Rebane and Pearl extended Chow and Liu's algorithm to causal polytree construction [19]. The SLA algorithm extends those algorithms further to Bayesian network construction. This

algorithm also makes the following two assumptions: the database attributes have discrete values and there are no missing values in all the records; the volume of data is large enough for reliable CI tests.

3 The SLA Algorithm

The SLA algorithm has three phases: drafting, thickening and thinning. The drafting phase of this algorithm is essentially Chow and Liu's tree construction algorithm; the thickening and thinning phase are designed to extend tree construction to general bayesian network construction. In the drafting phase, this algorithm computes the mutual information of each pair of nodes as a measure of independency, and creates a draft based on this information. The draft is a singly connected undirected graph (a graph without loops). In a special case, when the bayesian network is a tree or polytree, this phase can construct the network correctly and the second and third phase will not change anything. In the thickening phase, the algorithm adds edges when the pairs of nodes cannot be *d-separated*. The result of this phase has the structure of an *independence map* (*I-map*) [19] of the underlying dependency model in the case that the underlying model is *normal DAG-Faithful* [8]. In the thinning phase, each edge of the *I-map* is examined using CI tests and is removed if the two nodes of the edge can be *d-separated*. The result of this phase has the structure of a *perfect map* [19] when the underlying model is *normal DAG-Faithful*. At the end of the third phase, the SLA algorithm also carries out a procedure to orient the edges of the graph. The basics of the three phases are reported in Figure 1, Figure 2 and Figure 3.

The SLA algorithm uses the procedures: **try_to_separate_A**, **try_to_separate_B** and **orient_edges** (see Figure 2 and Figure 3). Given a graph and two nodes *a* and *b*, the procedure **try_to_separate_A** tries to identify if these two nodes are *d-separated.* From the definition of bayesian network [17] we know that if two nodes *a, b* in the network are not connected, they can be *d-separated* by the parent nodes of *b* which are in the paths between those two nodes. (We assume that node *a* appears earlier in the node ordering than *b*.) Those parent nodes form a set *P*. If node ordering is known, we can get *P* immediately and only one CI test is required to check if two nodes are *d-separated*. Since this information is usually not given, we have to use a group of CI tests to find such *P*. By assuming that removing a parent node of *b* will not increase the mutual information between *a* and *b*, the above procedure tries to find set *P* by identifying and removing the child-nodes and irrelevant nodes from the set of neighbors of *a* and the set of neighbors of *b* one at a time using a group of computations and comparisons of conditional mutual information. However, this assumption may not be true when the underlying structure satisfies the following conditions:

1. There exists at least one path from *a* to *b* through a child-node of *b* and this child-node is a *collider* on the path.
2. In such paths, there are one or more *colliders* besides the child-node and all these *colliders* are the parents or ancestors of *b*.

The drafting phase is composed by five steps:

D.1. Initiate a graph $G(V, E)$ where V={all the nodes of a data set}, E={ }. Initiate an empty list L.

D.2. For each pair of nodes (v_i, v_j) where $v_i, v_j \in V$, compute mutual information $I(v_i, v_j)$ using equation (1). For all the pairs of nodes that have mutual information greater than a certain small value ε, sort them by their mutual information and put these pairs of nodes into list L in decreasing order. Create a pointer p that points to the first pair of nodes in L.

D.3. Get the first two pairs of nodes of list L and remove them from it. Add the corresponding edges to E. Move the pointer p to the next pair of nodes.

D.4. Get the pair of nodes from L at the position of the pointer p. If there is no *adjacency* path between the two nodes, add the corresponding edge to E and remove this pair of nodes from L.

D.5. Move the pointer p to the next pair of nodes and go back to step D.4 unless p is pointing to the end of L or G contains n-1 edges. (n is the number of nodes in G.)

Fig. 1. Drafting the network

The thickening phase is composed by three steps:

Tk.1. Move the pointer p to the first pair of node in L. Get the pair of nodes from L at the position of the pointer p.

Tk.2. Call procedure **try_to_separate_A(current graph, node1, node2)** to see if this pair of nodes can be separated in current graph. If so, go to next step; otherwise, connect the pair of nodes by adding a corresponding edge to E. (Procedure **try_to_separate_A** will be presented later in this subsection.)

Tk.3. Move the pointer p to the next pair of nodes and go back to step Tk.2 unless p is pointing to the end of L.

Fig. 2. Thickening the network

The thinning phase is composed by three steps:

Th.1. For each edge in E, if there are other paths besides this edge between the two nodes, remove this edge from E temporarily and call procedure **try_to_separate_A (current graph, node1, node2)**. If the two nodes are dependent, add this edge back to E; otherwise remove the edge permanently.

Th.2. For each edge in E, if there are other paths besides this edge between the two nodes, remove this edge from E temporarily and call procedure **try_to_separate_B (current graph, node1, node2)**. If the two nodes are dependent, add this edge back to E; otherwise remove the edge permanently. (Procedure **try_to_separate_B** will be presented later in this subsection.)

Th.3. Call procedure **orient_edges (current graph)**. (This procedure will be presented later in this subsection.)

Fig. 3. Thinning the network

In such structures, procedure **try_to_separate_A** may identify a parent-node of b as a child-node of b and remove it erroneously. As a result, the procedure fails to separate two *d-separated* nodes. To deal with these structures, a correct procedure **try_to_separate_B** is introduced. Theoretically, is possible to use procedure **try_to_separate_B** to replace procedure **try_to_separate_A,** since they do the same thing and both of them have complexity $O(N^4)$ on CI test. But in practice, procedure **try_to_separate_A** usually uses fewer CI tests and requires smaller condition-sets. Therefore SLA tries to avoid using procedure **try_to_separate_B** whenever it is possible.

Among the nodes in Bayesian networks, only *colliders* can let information pass through them when they are instantiated. The **orient_edges** procedure uses this feature to identify *colliders*. All other edge orientations are virtually based on these identified *colliders*. The *collider* based edge orientation methods have also been studied in [19,23].

4 Association rules

Association rules describe correlation of events and can be regarded as probabilistic rules. Events are "correlated" if they are frequently observed together. For example, in the case of sale transactions, an event is the sale of a particular product and association rules express which items are usually bought together.

Consider a database D consisting of a single table. An association rule [1] is a rule of the form

$A_1=v_{A1}, A_2=v_{A2}, \ldots, A_j=v_{Aj} \Rightarrow B_1=v_{B1}, B_2= v_{B2}, \ldots, B_k=v_{Bk}$

where $A_1, A_2, \ldots, A_j, B_1, B_2, \ldots, B_k$ are attribute of D and $v_{A1}, v_{A2}, \ldots, v_{Aj}, v_{B1}, v_{B2}, \ldots, v_{Bk}$ are values such that v_{Ai} (v_{Bh}) belongs to the domain of the attribute A_i (B_h).

More formally, an association rule can be defined as follows.

An *item* is a literal of the form $A_i=v_{Ai}$ where A_i is an attribute of D and v_{Ai} belongs to the domain of A_i. Let I be the set of all the possible items. A *transaction* T is a record of D.

An *itemset* X is a set of items, i.e. it is a set X such that $X \subseteq I$. We say that a transaction T *contains* an itemset X if $X \subseteq T$ or, alternatively, if T satisfies all the literals in X.

The *support* of an itemset X (indicated by *support(X)*) is the fraction of transactions in D that contain X. The support of the opposite of an itemset (indicated by *support(!X)*) is the fraction of transactions in D that do not contain X. Thus, *support(!X)*= 1-*support(X)*.

An *association rule* is an implication of the form $X \Rightarrow Y$, where X and Y are itemsets and $X \cap Y \neq \emptyset$. The *support* of $X \Rightarrow Y$ (indicated by *support(X⇒Y)*) is *support(X ∪ Y)*. The *confidence* of $X \Rightarrow Y$ (indicated by *confidence(X⇒Y)*) is the fraction of transactions in D containing X that also contain Y. Thus, *confidence(X⇒Y) = support(X ∪ Y) / support(X).*

Given an association rule $X \Rightarrow Y$, we are interested in the following parameters:

- The *lift* [4] (called *interest* in [6]) of $X \Rightarrow Y$ (indicated by *lift(X⇒Y)*) is given by *lift(X⇒Y) = confidence(X⇒Y) / support(Y)*. Thus, *lift(X⇒Y) = support(X∪Y) / (support(X) × support(Y))*.
- The *leverage* [26] of $X \Rightarrow Y$ (indicated by *leverage(X⇒Y)*) is given by *leverage(X⇒Y) = support(X ∪ Y) - support(X) × support(Y)*.
- The *conviction* [6] of $X \Rightarrow Y$ (indicated by *conviction(X⇒Y)*) is given by *conviction(X⇒Y) = support(X) × support(!Y) / support(X ∪ !Y)*. Observing that *support(X∪!Y) = support(X) - support(X∪Y)*, we have that *conviction(X⇒Y) =(1-support(Y)) / (1-confidence(X⇒Y))*.

Moreover, other parameters can be defined that regard not a single association rule but a set of specific association rules, namely, the set of association rules that relate two variables. Supposing that we have binary variables X and Y, such a set of association rules would be $\{X=0 \Rightarrow Y=0, X=1 \Rightarrow Y=0, X=0 \Rightarrow Y=1, X=1 \Rightarrow Y=1\}$ plus the rules with head and body exchanged.

These parameters are Person's X^2 and the Cramer index [11]. Person's X^2 has been proposed as an interesting parameter for association rules in [7] and [22]. In order to define Person's X^2 and the Cramer index, let us consider two variables X and Y where X can assume I different values $x_1,...x_I$ and Y can assume J different values $y_1,...y_J$. Moreover, let us define the following parameters: $n=|D|$, $n_{ij}=support(\{X=x_i,Y=y_j\})n$, $n_{i\bullet}=support(\{X=x_i\})n$, $n_{\bullet j}=support(\{Y=y_j\})n$ and $n^*_{ij}=n_{i\bullet}n_{\bullet j}/n$. X^2 is then given by

$$X^2 = \sum_{i=1}^{I}\sum_{j=1}^{J}\frac{(n_{ij}-n^*_{ij})^2}{n^*_{ij}} \tag{3}$$

The Cramer Index V is instead given by

$$V = \sqrt{\frac{X^2}{n\min\{(I-1),(J-1)\}}} \tag{3}$$

X^2 is a statistics proposed by Karl Pearson for verifying the hypothesis of stochastic independence between two variables X and Y. X^2 is 0 if X and Y are independent, while the higher it is the less probable it is that the two variables are independent. The Cramer Index scales the value of X^2 to the maximum that can assume in the table examined, therefore $0 \le V \le 1$. If $V=1$ we have the maximum dependence between X and Y.

5 BNL-rules Algorithm

In this section, we propose a novel method for performing the drafting phase of the SLA algorithm by using association rules parameters. The algorithm we propose is called BNL-rules (Bayesian Network Learner with association rules).

During our experiments we tried to use both the most specific association rules and the most general ones. The most specific rules demonstrated to be unuseful for our task because they were not able describe the single dependency/independency relation between two variables, so we decided to use the most general ones, i.e., those with only one antecedent and only one consequent (named one-to-one rules). Each rule is characterized by the typical association rule parameters described in Section 3 (lift, conviction and leverage). Moreover, we associate to each such rule the values of the X^2 and Cramer Index parameters relative to the couple of variables involved in the rule.

The BNL-rules algorithm considers all the possible one-to-one rules and computes for all of them the values of the parameters. Then it sorts the rules in descending order of one of the parameters and builds a draft of the network: if the network has N

nodes, the tool creates a network connecting the nodes indicated in the rules, without introducing loops, until it reaches N-1 arcs. In this way we obtain an undirected acyclic graph. At the moment we haven't identified a method for learning arc orientations by using association rule parameters.

A formal description of the BNL-rules algorithm is presented below.

BNL-rules algorithm

Given a set of examples described by N variables, the bayesian network drafting performed by BNL-rules is realized in in six steps:

D.1. Create a graph *G(V, E)* where *V*={all the nodes of a data set}, *E*={ }. Create an empty list *L*.

D.2. For each one-to-one rule compute the parameters. Each rule and associated parameters become an element of list *L*.

D.3. **Choose one of the parameters (lift, leverage, convinction, X^2 or Cramer Index), and sort the elements of *L* in decreasing order with respect to this parameter.**

D.4. Get the first element of list *L* and remove it from the list. Add the corresponding edges to *E*. Move the pointer *p* to the next element of *L*.

D.5. Get the element from *L* at the position of the pointer *p*. If there is no *adjacency* path between the two nodes, add the corresponding edge to *E* and remove this element from *L*.

D.6. Move the pointer p to the next element and go back to step D.5 unless p is pointing to the end of L or G contains N-1 edges.

This algorithm is similar to the one proposed by SLA for the drafting phase, but instead of computing the mutual information, the algorithm computes the parameters of all the possible one-to-one rules.

6 Comparison between SLA and BNL-rules algorithms

We experimented the algorithms with four different bayesian networks:

- The "Visit to Asia" network: A belief network for a fictitious medical example about whether a patient has tuberculosis, lung cancer or bronchitis, related to their X-ray, dyspnea, visit-to-Asia and smoking status. It has 8 nodes and 8 arcs. It is described in [16].
- The "Car_diagnosis" network: A belief network for diagnosing why a car won't start, based on spark plugs, headlights, main fuse, etc. It has 18 nodes and 20 arcs. It is described in [18];
- The "ALARM" network: ALARM stands for "A Logical Alarm Reduction Mechanism". This is a medical diagnostic system for patient monitoring. It is a nontrivial belief network with 8 diagnoses, 16 findings and 13 intermediate variables (36 nodes and 46 arcs). It is described in [3];
- The "Boelarge92" network: A subjective belief network for a particular

scenario of neighborhood events, that shows how even distant concepts have some connection. It has 24 nodes and 35 arcs. It is described in [5];

A database of examples was generated from each network using NETICA [18]. Then SLA and BNL-rules are applied to learn back the network. The learned network is compared with the original and the numbers of missing and extra arcs are counted. Tables 1, 2, 3 and 4 show these comparison results for each network, for each algorithm and for three different dataset dimensions (5.000, 20.000 and 100.000 examples).

Table 1. Results for the "Visit to Asia" network

Data Set	SLA Drafting		BNL-rules Leverage		BNL-rules Convinction		BNL-rules Lift		BNL-rules Pearson's X^2		BNL-rules Cramèr Index	
	Miss	Extra	Miss	Extra	Miss	Extra	Miss	Extra	Miss	Extra	Miss	Extra
5000	1	0	3	2	3	2	3	2	1	0	1	0
20000	1	0	3	2	3	2	3	2	1	0	1	0
100000	1	0	3	2	4	3	3	2	1	0	1	0

Table 2. Results for the "Car diagnosis" network

Data Set	SLA Drafting		BNL-rules Leverage		BNL-rules convinction		BNL-rules Lift		BNL-rules Pearson's X^2		BNL-rules Cramèr Index	
	Miss	Extra	Miss	Extra	Miss	Extra	Miss	Extra	Miss	Extra	Miss	Extra
5000	4	1	5	2	9	6	10	7	5	2	5	2
20000	4	1	3	0	7	4	8	5	3	0	4	1
100000	4	1	3	0	7	4	8	5	3	0	4	1

Table 3. Results for the "Alarm" network

Data Set	SLA Drafting		BNL-rules Leverage		BNL-rules Convinction		BNL-rules Lift		BNL-rules Pearson's X^2		BNL-rules Cramèr Index	
	Miss	Extra	Miss	Extra	Miss	Extra	Miss	Extra	Miss	Extra	Miss	Extra
5000	12	2	13	3	13	3	13	3	11	1	13	3
20000	12	2	11	1	15	5	12	2	11	1	13	3
100000	12	2	11	1	13	3	10	0	11	1	13	3

Table 4. Results for the "Boerlage" network

Data Set	SLA Drafting		BNL-rules Leverage		BNL-rules Convinction		BNL-rules Lift		BNL-rules Pearson's X^2		BNL-rules Cramèr Index	
	Miss	Extra	Miss	Extra	Miss	Extra	Miss	Extra	Miss	Extra	Miss	Extra
5000	14	0	14	0	15	1	16	2	14	0	15	1
20000	14	0	14	0	14	0	16	2	14	0	14	0
100000	14	0	14	0	14	0	16	2	14	0	14	0

In order to test whether the tool used for generating the dataset has an influence over the results, we repeated the experiments with datasets generating the database with HUGIN [15]. The results obtained are identical except for the "Alarm" network

drafted by BNL-rules employing leverage and a dataset of 100,000 examples that, in the case of HUGIN, has 12 missing arcs and 2 extra arc. Therefore the results are sufficiently independent from the database generation tool.

From the tables we can observe that BNL-rules with Pearson's X^2 always obtains the best results when used with 20,000 and 100,000 examples, outperforming SLA in two cases and performing equally well in the other two cases. When used with 5,000 examples, BNL-rules with Pearson's X^2 does worse than SLA only for the "Car diagnosis" network.

BNL-rules with leverage obtains results that are as good as those obtained with X^2 for all networks and for 20.000 and 100.000 examples apart from "Visit to Asia". However, note that "Visit to Asia" is very small and thus the results on it are less significative than those on the others.

Note also that while the performances of SLA do not improve with the number of examples, those of BNL-rules with X^2 improve in one case and those of BNL-rules with leverage improve in two cases when going from 5.000 to 20.000 examples.

7 Conclusions

In this work we describe a method for improving SLA [8], one of the most known algorithm for learning Bayesian network, by exploiting a number of association rules parameters. We propose a number of novel algorithms for performing the drafting phase. Each algorithm is based on the computation of a parameter usually defined in relation to association rules: leverage, conviction, lift, Pearson's X^2 or Cramer Index.

We experimented the algorithms on four different Bayesian networks: "Visit to Asia", "Car_diagnosis", "Alarm" and "Boelarge": the algorithm BNL-rules with X^2 obtained the best results and performed better that SLA in two cases and equally well in two other two cases better. By starting the second and the third learning phase of SLA from a better structure it is possible to achieve a better final result.

In the future we plan to compare SLA and BNL-rules on larger networks in order to obtain a more definitive confirmation of the validity of the approach.

References

1. Agrawal, R., Imielinski, T., Swami, A.: Mining association rules between sets of items in large databases. In: Buneman, P., Jajodia, S. (eds.): Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993. ACM Press 22(2) (1993) 207-216
2. Akaike, H.: A new Look at Statistical Model Identification. IEEE Trans. Automatic Control 19 (1974) 716-723
3. Beinlich, I.A., Suermondt, H.J., Chavez, R.M., Cooper, G.F.: The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In: Hunter, J. (ed.): Proceedings of the Second European Conference on Artificial Intelligence in Medicine (AIME 89). Springer, Berlin (1989) 247-256
4. Berry, J.A., Linoff, G.S.: Data Mining Techniques for Marketing, Sales and Customer Support. John Wiley & Sons Inc., New York (1997)

5. Boerlage, Brent: Link Strength in Bayesian Networks. MSc Thesis, Dept. Computer Science, Univ. of British Columbia, BC (1992).
6. Brin, S., Motwani, R., Ullman, J., Tsur, S.: Dynamic Itemset Counting and Implication Rules for Market Basket Data. In: Peckham, J. (ed.): Proceedings ACM SIGMOD International Conference on Management of Data. SIGMOD 26(2) (1997) 255-264
7. Brin, S., Motwani, R., Silverstein, C.: Beyond market baskets: Generalizing association rule to correlations. In: Proceedings ACM SIGMOD International Conference on Management of Data. SIGMOD (1997) 265-276
8. Cheng, J., Greiner, R., Kelly, J., Bell, D., Liu, W.: Learning Bayesian networks from data: An information-theory based approach, Artificial Intelligence 137 (2002) 43–90
9. Chow, C.K., Liu, C.N.: Approximating discrete probability distributions with dependence trees. IEEE Transactions on Information Theory 14 (1968) 462-467
10. Cooper, G., Herskovits, E.: A Bayesian method for the induction of probabilistic networks from data. Machine Learning 9 (1992) 309-347
11. Giudici, P.: Data mining - Metodi statistici per le applicazioni aziendali. McGraw-Hill, Milano (2001)
12. Heckerman, D., Geiger, D., Chickering, D.: Learning Bayesian Networks: the combination of knowledge and statistical data. Machine Learning 20 (1995) 197-243
13. Heckerman, D.: Tutorial on learning in Bayesian networks. In: Jordan, M. (ed.): Learning in Graphical Models. MIT Press, Cambridge, MA (1999)
14. Herskovits, E.H.: Computer-based probabilistic-network construction. Doctoral Dissertation, Medical Informatics, Stanford University (1991)
15. Hugin, http://www.hugin.com
16. Lauritzen, S.L., Spiegelhalter D.J.: Local computations with probabilities on graphical structures and their application to expert systems. J. Royal Statistics Society B 50(2) (1988) 157-194
17. Madigan, D., Raftery A.:. Model Selection and Accounting for Model Uncertainty in Graphical Models Using Occam's Window. J. Am. Statist. Association 89 (1994) 1535-1546
18. Netica, http://www.norsys.com
19. Pearl, J.: Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann, San Francisco (1988)
20. Rissanen, J.: Stochastic Complexity (with discussion). J. Roy. Statist. Soc. B 49 (1987) 223-239
21. Sigh, M., Valtorta, M.: Construction of Bayesian Network Structures from Data: a Brief Survey and an Efficient Algorithm. In: Heckerman, D., Mamdani, A. (eds.): Proceedings of the Conference on Uncertainty in Artificial Intelligence. Morgan Kaufmann, San Francisco (1993) 259–265
22. Silverstein, C., Brin, S., Motwani, R., Ullman, J.D.: Scalable Techniques for Mining Causal Structures. Data Mining and Knowledge Discovery 4(2/3) (2000) 163-192
23. Spirtes, P., Glymour, C., Scheines, R.: Causation, Prediction, and Search. Lecture Notes in Statistics. Springer, Berlin (1993)
24. Spirtes, P., Glymour, C., Scheines, R.: An algorithm for fast recovery of sparse causal graphs. Social Science Computer Review 9 (1991) 62-72
25. Suzuki, J.: Learning Bayesian Belief Networks Based on the MDL principle: An Efficient Algorithm Using the Branch and Bound Technique. IEICE Transactions on Communications Electronics Information and Systems (1999)
26. WEKA http://www.cs.waikato.ac.nz/~ml/weka/doc_gui/weka.associations.ItemSet.html#leverageForRule