Technical Report
LRDC-ONR-94-1
(1994)

Discrete factor analysis: Learning hidden variables in Bayesian networks.

Joel D. Martin & Kurt VanLehn
3939 O'Hara St.
University of Pittsburgh
Pittsburgh, PA, 15260
(412) 624-0843
martin@cs.pitt.edu

Keywords: Learning; Bayesian nets; Hidden variables; Fast, Heuristic method

Abstract

We present a space of Bayesian network topologies with hidden variables and present a method for rapidly constructing an appropriate such topology given sample data. In the topology, hidden variables can interact and as a result, can explain dependencies and independencies between observable variables. As well, the topology can permit polynomial time probabilistic inference. The learning method can be viewed as both a) factor analysis for discrete variables and b) cluster analysis with overlapping clusters (clumping).

1 Introduction

Even when faced with a flood of interacting variables, an intelligent agent can extract unseen structure that concisely explains its world. This paper presents a space of Bayesian network topologies with hidden variables (or factors) and a method for rapidly learning an appropriate topology from data. The learning method combines techniques from classical statistics and Bayesian motivated approaches to partitioning. It produces simple probabilistic models and can be viewed as both a) factor analysis for discrete variables and b) cluster analysis with overlapping clusters (clumping). Potential applications include creation of predictive domain models, simplification of intractable probabilistic models, and automated scientific discovery.

This approach has many advantages. The class of network topologies is simple and useful.

- It can support polynomial time probabilistic inference.
- Hidden variables can interact to influence observable variables.
- Observable variables are conditionally independent given the hidden variables.
- The resulting network captures conditional independencies among the observable variables.

In addition, the learning method presented here,

- Can approximate a distribution when a simple topology of factors is non-optimal for the data.
- Allows the hidden variables to have an arbitrary number of values.

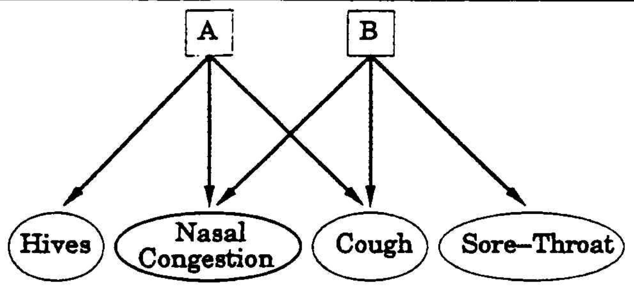

Figure 1: A simple example of factor structure.

- Is very fast compared to other methods for learning hidden variables.

In Section 2, we present a simple topology for hidden variables. Section 3 describes the learning algorithm. Section 4 summarizes preliminary results using the learning method. Section 5 discusses related work, and Section 6 summarizes and discusses some limitations.

2 A topology for concise explanation

Concise explanations must be more compact than the original information, must be easy to reason with, and need not be perfect. For example, two-dimensional pixel arrays can be described as collections of edges, corners, etc. In this case, the explanation is simpler than the original pixel arrangements, is easier to use, and may omit less central information.

Hidden variables can be created in a Bayesian network to explain dependencies and independencies in a domain. Doing so does not necessarily mean that such a variable 'exists' or can be measured. Rather, it is a computational and statistical convenience.

A simple, easily learned, explanation of the dependencies between n discrete variables is a *factor structure* of hidden variables. We define a factor structure to have four characteristics:

1. There are $\lceil \log n \rceil$ hidden variables, called factors.
2. The factors jointly determine the probability of the values of the observable variables.
3. When the values of the observable variables are *not* known, the factors are independent.
4. When the values of the factors are known, the observable variables become independent.

Because there are only $\log n$ factors, inference can be performed in polynomial time. If the values of the factors are known, the observable variables become independent. There are only $r^{\log n}$ assignments of values to the factors, where r is the maximum number of values per factor and is assumed to be constant. Therefore, inference can be performed by combining over all possible assignments of values to factors in $O(n^{1+p})$ time, where $p = \log r$. We also can learn a *complete* factor structure which allows unlimited number of factors. However, in such a structure in the general case, inference in NP-hard.

For example, Figure 1 shows a simple Bayesian network of four observable variables: sore-throat, nasal congestion, cough, and hives. There are two factors, A and B. In this network, the variable B roughly corresponds to whether a patient has a viral infection and variable A to whether the patient has an allergic reaction.

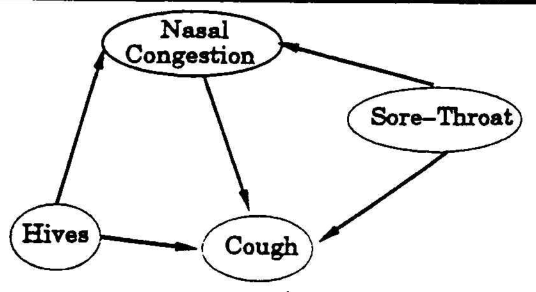

Figure 2: A Bayesian network with no hidden variables.

The two factors concisely summarize important information about the variables. They capture the dependencies between the three symptoms of a viral infection and the three symptoms of an allergic reaction and eliminate the need for direct dependencies between the symptoms. As well, the factors capture that hives are independent of having a sore-throat. This information is also available in a Bayesian network without hidden variables but is less apparent (Figure 2). As well, in a Bayesian network with only observable variables, each dependency must have a direction even if there is no data-driven or theory-driven justification for one direction over another.

Factor structures have important applications whenever simpler representations are beneficial. For instance, discovered factors could correspond to diseases or collection of symptoms that cooccur (syndromes). Because factor structures can produce rich, interpretable models of discrete data, they are useful for the same reasons as are cluster analyses, traditional factor analyses, and re-representation using hidden nodes in neural networks.

3 Learning factor structures

FL (Factor Learner) is a learning method that constructs factor structures from empirical data. To do so it combines techniques from classical statistics and Bayesian motivated approaches to clustering. It takes as input a set of variable-value vectors and produces a factor structure. It searches for an appropriate factor structure in two steps. First, it uses classical tests of association to build the topology, i.e., decide which observable variables each factor should cover and how many factors to use. Second, it uses an approximation of a Bayesian approach to instance partitioning (clustering) to determine a highly probable number of values per factor and highly probable conditional dependencies between factors and observable variables.

3.1 Finding the topology

The topology of factors, i.e., the number of factors and their connectivity, is determined using classical tests of association between pairs of discrete variables[1]. A complete structure of factors (not limited to $\log n$ factors) can be identified using the pattern of dependence and independence between pairs of observable variables[2]. Two rules govern the structure:

[1] FL currently uses Pearson's χ^2 test of association. However, there is no theoretical commitment to a particular test.

[2] Higher-order conditional dependencies and independencies could provide converging evidence but are not strictly necessary.

- If two observable variables are interdependent, then one or more factors directly influence both variables.
- If two observable variables are independent, then no factor directly influences both.

FL constructs the factors by finding a set of cliques in an undirected graph of dependency relations. In this graph, a direct link between two variables means they are dependent, and no link means they are independent.

In learning the factors, FL attempts to find the set of $\lceil \log n \rceil$ cliques that covers the dependencies of which it is most certain. However, it cannot quickly (in polynomial time) check all possible cliques, otherwise it would be able to find the maximal clique which is an NP-hard problem. Instead, FL uses a heuristic approach.

First, FL constructs the graph by introducing a link between two variables when the test of association indicates dependence between those two variables. FL labels each link with the probability that those two variables are dependent. Second, FL extracts cliques by iterating through the variables and adding a variable to an existing clique if the variable is dependent on all other variables in that clique. After doing this, if the variable has dependency links not already in a clique, FL starts another clique with two variables. Third, FL performs a greedy search for the $\log n$ cliques that jointly maximizes the sum of the labels represented in the set of cliques[3].

Each of the resulting cliques corresponds to a factor in the factor structure and the variables in the cliques are the variables that directly dependent on the factor. When two or more cliques share a variable, then the variable has two or more factors on which its value depends.

3.2 Constructing the factors

Once a suitable topology is chosen, FL must choose the number of values for each factor and choose the conditional probabilities. FL attempts to do so very rapidly, only approximating the normative Bayesian method for learning hidden variables. The normative way to learn hidden variables is to consider all possible numbers of values and all possible assignments of hidden variable values to the instances in the dataset (Cooper & Herskovits, 1992; Cooper, 1994). With the normative approach, decisions could be based on all possible assignments, their priors, and their probabilities given the instances. Cooper (1994) recently proposed a polynomial-time algorithm for considering all assignments of values that occur in the instances. However, doing so with a large number of variables and a large number of instances can be prohibitively expensive.

To be faster than the normative approach, a learner must have some way to find a highly probable assignment without considering all possible assignments of factor values to instances. FL uses an incremental, greedy search for a single most probable assignment of values. It assigns factor-values to each instance one at a time. The assignment for the *nth* instance is the most probable given the current factor structure which is based on the first $n-1$ instances.

Although FL temporarily assumes a single set of factor values (the most probable) as the cause of each instance, that is merely a convenience for rapid learning of a factor structure. This assignment does not represent a permanent guess by the system about the assignment of values for an instance. Instead, it merely provides the size and direction of the next step for the hill-climbing search.

Also to save computation time, FL does not consider all possible combinations of factor values when seeking the most probable assignment for each instance. Instead, it finds the most probable value for each factor serially, each time including the influence of previously chosen factor-values. For example, if there are three binary factors, there are eight possible combinations of factor

[3]This step is omitted if the user chooses to acquire a complete factor structure.

values. FL does not examine all eight possibilities. Rather it greedily considers the values serially. To reduce the bias that serial assignment introduces, FL performs the assignment to factors in a random order.

The algorithm is as follows:

1. Initialize the factor structure to have one value per factor.
2. Given a factor structure in which the first $n-1$ instances have been partitioned by each factor, calculate the most probable assignment of the *nth* instance, I, to the values of each factor.
    - (a) Choose a random order of factors.
    - (b) Iterate over the factors:
        - i. For each value, V_{ij}, of the *ith* factor, F_i, calculate the probability that the instance has the *jth* value of that factor given the assignments to all previous factors, $P(V_{ij}|I, V_{i-1}, V_{i-2}, ...)$.
        - ii. Calculate the probability of a new value for the *ith* factor, $P(V_{i0}|I, V_{i-1}, V_{i-2}, ...)$.
        - iii. Label the instance with the factor value with the maximum probability, updating the estimated prior probabilities of the *ith* factor's values and the estimated conditional probabilities of the observable values given the factor's value.

In all cases where probabilities must be estimated from frequencies we used the following formula:

$$P(V_i = a_{ij}) = \frac{freq_{ij} + \alpha_j}{n_i + \alpha_{i0}} \qquad (1)$$

The quantities, α_j, are parameters of a Dirichlet distribution (one for each value of the variable) and α_{i0} is the sum over those parameters. We set all the α_i's to be 1.0 and never vary them.

This algorithm never checks if it has done the right assignment and never undoes an assignment. Clearly, it cannot guarantee finding the most probable partitioning for any factor. Indeed, all it does is attempt to find the maximum likelihood partitioning assuming that the current hypothesis is maximally likely.

Although FL cannot guarantee finding the most probable model, it is very fast. Each of m instances is tested against each of r values for each of the $\log n$ factors, making for $O(mr\log n)$ tests. Each test iterates over all observable variables, producing an overall cost of $O(mrn\log n)$. If we assume a constant number of values, this becomes $O(mn\log n)$.

3.3 An example

Figure 3 shows an example factor structure. We took a random sample of 5000 instances based on the distribution shown in the Figure and gave those instances to FL. FLfirst computed the classical test of association for all pairs of variables. It returns the probability of the observed association between the variables given that the variables are independent.

Table 1 shows the results of the tests of association. FL concludes that there are dependencies between V1, V2, and V3, a dependency between V3 and V4, and dependency between V4 and V5. As a result, FL creates three factors: one that influences variables V1, V2, and V3, one that influences variables V3 and V4, and one that influences V4 and V5. After learning, FL recovered the correct factor structure as shown in Figure 3. However, for two of the factors, it proposed additional values that had extremely low prior probabilities. These extra values are artifacts of the incremental hill-climbing search. FL should have a means of eliminating those values that are not adequately supported by the instances.

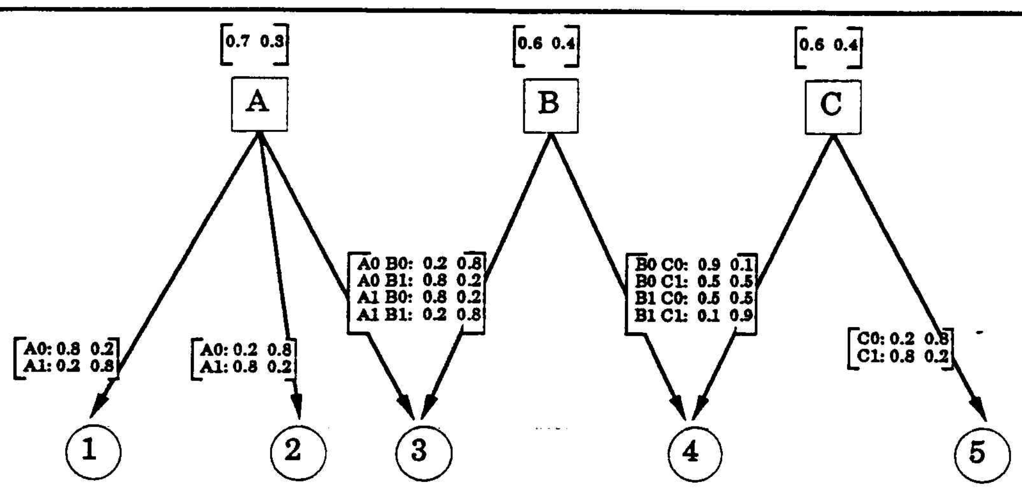

Figure 3: The target factor structure from which the learning sample was generated.

Table 1: Triangular matrix containing $P(V_i \, ind \, V_j)$, the probability that two variables are independent. All probabilities below 0.05 were taken as evidence of association.

Variable	V1	V2	V3	V4	V5
V1	-	0.00	0.00	0.97	0.45
V2	-	-	0.00	0.16	0.82
V3	-	-	-	0.00	0.18
V4	-	-	-	-	0.00

4 Other Results

In this section, we illustrate FL's performance with preliminary empirical tests. First, we examined the extent to which it can recover a factor structure when given instances that were generated by that target structure. Second, we examined the structure of factor structures built to summarize more complex Bayesian nets. In this case, FL learned from instances that were generated using a random Bayesian net of only observable variables.

For the first examination, we generated 10 random factor structures in which there were between 8 and 32 binary observable variables, between 3 and 5 factors, between 2 and 5 values per factor, and the average in-degree of the observable variables was 2. Each graph had a distribution that correctly reflected the graphical dependencies and independencies. From each graph, a single sample of 5000 instances was computed. Each instance corresponds to an assignment of values to all observable and hidden variables. A Monte Carlo simulation technique (Henrion, 1988) was used to generate instances such that the probability of sampling the instance was equal to the probability of the instance given the factor structure.

For all 10 factor structures, FL recovered the correct number of factors. It identified the correct number of values for 10 of the 36 factors across the 10 structures. Most of the errors were the result of extra values with low prior probabilities. FL included one incorrect edge in one of the 10 structures and omitted 40 of the 390 correct edges.

FL learned very rapidly. It was implemented in Lucid common lisp (not optimized) on a

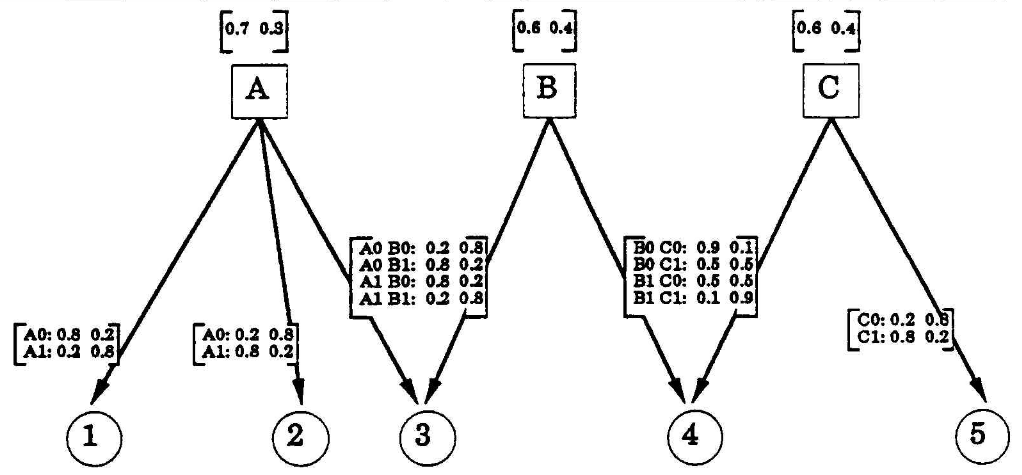

Figure 4: The factor structure learned from the instances.

DecStation 5000. Its slowest time for learning was for a situation with 32 observable variables, five factors, and 5000 instances. It required 1936 seconds (32 minutes, 16 seconds) to finish.

For the second examination, we generated 10 random Bayesian networks in which there were between 8 and 32 binary observable variables, average degree of the observable variables was 3. These graphs had no factors, and all edges were between observable variables. Each graph had a distribution that correctly reflected the graphical dependencies and independencies. From each graph, a single sample of 5000 instances was computed.

For all 10 Bayesian networks, FL always constructed the maximum number of factors, $\lceil \log n \rceil$. It produced an average in-degree for the observable variables of 2.6 and an average number of values per factor of 4.5.

5 Related Work

FL is one method for learning a probabilistic model from data. FL contrasts with most other Bayesian net learning methods in that it learns a factor structure. Many of the other methods (e.g., Chow & Liu, 1968; Cooper & Herskovits, 1992) attempt to learn the most appropriate (most probable) direct dependencies between observable variables. Some of these methods (Cooper, 1994; Spirtes, Glymour, & Scheines, 1993) do permit hidden nodes in exceptional cases when direct dependencies are inappropriate.

The learning methods for structuring the observables have the advantage of directly representing the conditional independencies between the variables and of representing arbitrary joint probability distributions. However, these methods must decide how to obtain complete Bayesian nets when they are not able to determine the directionality of all dependencies. FL only represents the dependencies between the observables and is not concerned with the directionality of the association.

Pearl (1988) and Connally (1993) have explored the possibility of using trees of hidden variables that render all observable variables independent. In effect, the trees explain all the dependencies between the observables. The trees presuppose a completely connected dependency graph relating the observables and construct a tree of new variables that concisely represent the joint probability distribution. The advantage of doing so is that there are fewer parameters to estimate and the resulting tree is a simpler, more efficient probabilistic model.

However, trees of hidden variables are non-optimal when there are independencies between observables. Indeed, because the observable variables are at the leaves of the tree, the trees do not graphically represent any independencies between the observable variables (but not conditional independencies). In contrast, factor structures can graphically represent the independencies between any pair of variables. Interestingly, each factor in a factor structure corresponds to a completely connected dependency graph. It is possible that factors can be decomposed into simpler sets of interacting factors. Later versions of FL will incorporate methods for decomposing a single factor into a tree of interacting factors.

Because FL does learn a factor structure, it can be viewed as performing discrete factor analysis or overlapping cluster analysis. FL learns a topology from the same class as does traditional factor analysis (e.g., Cattell, 1978). However, in FL both the factors and the observable variables have discrete, unordered values; whereas classical factor analysis assumes linear relationships between the factors and the measured variables.

Similarly, like traditional cluster analysis FL partitions the instances into disjoint subsets (e.g., Hartigan, 1975). However, the probabilistic model rather than the partitioning is the goal of learning. Because it permits multiple factors, it expands the capabilities of most existing clustering/concept formation methods (e.g., Anderson & Matessa, 1992; Fisher, 1987). Moreover, in the few existing methods for extracting overlapping clusters, the clusters are either not permitted to interact or they interact in ad hoc ways (e.g., Segen, 1988; Martin & Billman, 1994).

Finally, the classical statistics techniques used in FL are similar to most other methods for learning Bayesian networks (e.g., Chow & Liu, 1968; Pearl, 1988). Cooper and Herskovits (1992) use Bayesian techniques to find the most probable structure and can use this technique to add hidden variables. In principle, exact Bayesian methods for hidden variables could identify the most probable structures of factors given the data and suitable priors (Cooper & Herskovits, 1992). However, with a large number of variables, exact methods are prohibitively expensive. FL is one method for approximating the exact methods.

6 Discussion

The class of topologies called factor structures can be learned that represent the joint probability distribution across discrete observable variables. Most importantly, factor structures are extracted to provide a simple explanation of the dependencies among the variables. It provides a description with fewer variables, it supports polynomial time probabilistic inference, and it may lose some precision.

We present a learning algorithm called FL that combines techniques from classical statistics and Bayesian motivated approaches to clustering to construct these factor structures. Preliminary evaluations suggest that it does its task very well and is very rapid. Future work will seek an analysis of the conditions under which FL can recover a factor structure that was used to generate a set of instances. As well, we will determine empirically its ability to simplify large, complex Bayesian nets ($\tilde{5}00$ nodes).

There are several ways that FL can be improved. First, it can be extended to permit both discrete and real-valued observable variables to interact in the same factor structure. In addition, we will build a completely incremental version of FL by incorporating techniques used by Martin and Billman (1994) to learn overlapping concepts.

Finally, although FL is more efficient than normative Bayesian learning techniques for hidden variables, it cannot easily incorporate prior belief in factor structure and cannot provide an overall confidence or probability of the resulting factor structure. We will explore ways in which FL can mimic these features of a Bayesian approach and ways in which FL can be used to speed up search using a Bayesian approach.

Bibliography

Anderson, J. R. & Matessa, M. (1992). Explorations of an incremental, Bayesian algorithm for categorization. *Machine Learning, 9*, 275–308.

Cattell, R. B. (1978). *The scientific use of factor analysis in behavioral and life sciences.* New York: Plenum Press.

Cheeseman, P., Freeman, D., Kelly, J., Self, M., Stutz, J., & Taylor, W. (1988). Autoclass: a Bayesian classification system. In *Proceedings of the Fifth International Conference on Machine Learning*. Ann Arbor, MI: Morgan Kaufmann.

Chow, C.K. & Liu, C.N. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory, 14*, 462–467.

Connolly, D. (1993). Constructing hidden variables in Bayesian networks via conceptual clustering. In *Proceedings of the Tenth International Conference on Machine Learning*, (pp. 65–72). Amherst, MA: Morgan Kaufmann.

Cooper, G.F. (1994). A Bayesian method for learning belief networks that contain hidden variables. To appear in *Journal of Intelligent Information Systems.*

Cooper, G.F. & Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning, 9*, 309–347.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning, 2*, 139–172.

Hartigan, J. A. (1975). *Clustering Algorithms*. New York: Wiley.

Henrion, M. (1988). Propagating uncertainty in Bayesian networks by logic sampling. In J.F. Lemmer & L.N. Kanal (Eds.), *Uncertainty in Artificial Intelligence 2.* North-Holland, Amsterdam, pp. 149–163.

Martin, J. D. and Billman, D. O. (in press). The acquisition and use of overlapping concepts, *Machine Learning.*

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems*, San Mateo, CA: Morgan Kaufmann.

Segen, J. (1988). Conceptual clumping of binary vectors with Occam's razor. In *Proceedings of the Fifth International Conference on Machine Learning*, (pp. 47–53). Ann Arbor, MI: Morgan Kaufmann.

Spirtes, P., Glymour, C., & Scheines, R. (1993). *Causation, Prediction, and Search.* New York: Springer-Verlag.