Neural Nets WIRN11
B. Apolloni et al. (Eds.)
IOS Press, 2011
© 2011 The authors and IOS Press. All rights reserved.
doi:10.3233/978-1-60750-972-1-97

From Examples to Bayesian Inference

Francesco A. N. PALMIERI [a,1], Domenico CIUONZO [a] Davide MATTERA [b]
Gianmarco ROMANO [a] Pierluigi SALVO ROSSI [a]
[a] *Seconda Università di Napoli (SUN), Italy*
[b] *Universitá di Napoli Federico II, Italy*

Abstract. We show how to build an associative memory from a finite list of examples. By means of a fully-blown example, we demostrate how a probabilistic Bayesian factor graph can integrate naturally the discrete information contained in the list with smooth inference.

Keywords. Bayesian Networks, Artificial Intelligence, Propagation of Belief

Introduction

Belief propagation techniques on graphs [2] provide a very promising paradigm for a merge of probability theory and logic [1]. In many applications of communications, artificial intelligence and digital signal processing [4], Bayesian methods have already demostrated how they can integrate smoothly observations and previous knowledge. The idea of "injecting" in a graph our current observations, and "collecting" the response of the system after belief propagation, can be very useful in providing dynamic inference and support to human decision making.

The Bayesian paradigm seems to be the most "natural" framework on which build adaptive memories, that are generalizations of common logic lists and imperfect knowledge. Most hypotheses about brain functioning [3], seem to indicate that memory is a distributed "hardwired" property of the neural network, and that it is bidirectional, hierachical and inferential.

In our group we have started a combined effort to approach a number of data fusion problems where the challenge is to integrate very heterogeneous data. Systems that can learn directly from the observations the mutual relations among variables, and store them into a distributed graph for subsequent inferences, can have a substantial impact on the applications. Unfortunately the success of a Bayesian network is mainly related to learning the most appropriate graph. Clearly if the graph structure is known, we have to solve a parametric problem in learning the node parameters and this is usually achieved using a distributed version of the Expectation Maximization (EM) algorithm. Conversely, much more challenging is the problem of learning the network structure. Previous contributions have appeared in the literature to approach this task such as the famous Chow and Liu's algorithm [8] which grows a tree using second-order mutual information. Other

[1]Corresponding Author: Dipartimento di Ingegneria dell'Informazione, Seconda Università di Napoli, via Roma 29, 81031 Aversa (CE), Italy; E-mail: francesco.palmieri@unina2.it.

methods, such as Kutato and K2 [12], are based on entropy estimations. Also C4.5 algorithms [10] suggest how to build decision trees from examples. Mixtures of trees have also been proposed in [11], and trees based on greedy searches have been demonstrated in [14]. Also, *module networks*, proposed by Segal et al. [15], are based on information trees with homogeneous groups.

In our work we focus on bottom-up unsupervised tree construction with emphasis on the Factor Graph (FG) formulation [5] [4], that assigns variables to edges, and functions to blocks. Factor Graphs resemble common block-diagrams and seem to provide an easier path to VLSI and FPGA hardware realizations.

We show by means of a fully-blown toy example how to build an associative memory from examples. Even though the available data is just a discrete list of strings, embedding the examples into a probabilistic factor graph allows natural smooth access when knowledge is partial or noisy.

1. Associative Recall

Consider a string of variables $\mathbf{X} = (X_1, X_2, ..., X_N)$ that belong to the finite alphabets $\mathcal{X}_1, \mathcal{X}_2, ..., \mathcal{X}_N$. Suppose that we know nothing about the model that generates these variables, but we have available a finite list of examples $\mathbf{x}[i] = (x_1[i], x_2[i], ..., x_N[i])$, $i = 1, ..., \ell$. Direct memorization of this information would be based on ℓ memory registers of N cells with each cell capable of storing $|\mathcal{X}_1|, |\mathcal{X}_2|, ..., |\mathcal{X}_N|$ symbols respectively, where $|\mathcal{X}_i|$ denotes the cardinality of alphabet $\mathcal{X}_i$.

Standard *Address-Based Recall (ABR)* consists in providing an index that identifies the position of the desired word in a list. A much more general approach for information retrieval is to perform a so-called *Associative Recall (AR)*: *query the memory with a string* $\mathbf{y} = (y_1, y_2, ..., y_N)$*, wich represents one of the memories* $\mathbf{x}$ *with partially missing and/or erroneous elements; according to a pre-specified retrieval algorithm replace* $\mathbf{y}$ *with a best guess* $\hat{\mathbf{x}}$*, possibly providing a level of confidence about it*.

Note that AR includes as a special case ABR: it is sufficient to assign to one of the variables the role of the example index, say X_1, and query the memory with a string with all elements missing, except the first one.

Associative recall is much more natural mechanism of accessing stored memories in comparison to simple lists. For example, error-correcting codes, erasure codes, text parsers, all operate in this fashion. Also neural systems seem to be based mainly on an associative mechanism. Furthermore, the AR paradigm allows us to build a probabilistic model of our data that can interact smoothly with stored discrete examples and provide soft inferences from imperfect knowledge.

From a probabilistic point of view, N variables $(X_1, X_2, ..., X_N)$ are fully characterized by their joint probability mass function $p(x_1, x_2, ..., x_N)$ [2]. All the mutual interactions among the variables is contained in the structure of p which can be very complicated and unknown. The function p may represent the structure of a code, a specific generative model, a set of examples, a logic rule that ties the variables together, etc. Access to the information can be obtained via marginalization. For example, if the variable X_i is available, i.e. $p(x_i) = \delta(x_i - \xi_i)$, complete information about all the other variables is obtainable from the density $p(x_1, x_2, ., \xi_i, .., x_N)$. For example, if we are interested only in another variable, say X_j, the marginal density $p(x_j) = \sum_{x_n, n \neq j,i} p(x_1, x_2, ., \xi_i, .., x_N)$

represents the stored information about it. The beauty of the probabilistic framework is that the result of marginalization is a probability density that automatically provides a level of confidence about the answer. A further advantage is that our query can be based on uncertain knowledge (not a delta function) and the system can still be interrogated for a smooth comparison to the stored memory.

Marginalization, of course, can become a formidable task as N grows. However, if the density p is embedded into a loosely connected graph, via message propagation, marginalization can become feasible. Chains, trees, and other graphical structures are commonly used in coding and stochastic modeling as they provide efficient ways to perform inference in many applications [2][4]. When the graphs have loops, approximations, or more complicated algorithms, must be used for inference. However, in most of these cases the graph structure is known a priori, or imposed on the problem.

We focus on the challenging problem of "learning" the structure of the graphical model only from the examples.

In this paper we show, via a toy example, how to build a a hierarchical tree by clustering variables together in a progressive order. The tree has no loops and can be immediately used for belief propagation. We have presented a similar experiment in [7] where the examples were extracted from a Hamming code. The small-scale example that we present here suggests that the idea can be scaled-up with more complex data and become a feasible proposal for a universal associative memory.

2. Example

Consider the list of nine examples shown in Figure 1. Each example appears only once and the six variables have different alphabets. The various sets, their number of occurences and their marginal entropies are also shown.

In the first step of this construction we group the variables in pairs (X_1X_2), (X_3X_4), (X_5X_6), as in Figure 2. After recording and counting their occurences, we define new variables Y_1, Y_2, Y_3. The new variables are a more compact representation of the possible pairs because some configurations are repeated more than once and some never happen. Note that the cardinality of the new alphabets is generally smaller than the dimension of the product space for each pair. In larger scale applications, the cardinality of the new variable it is expected to be of the order of the cardinality of the respective typical sets $\{2^{H(X_1X_2)}, 2^{H(X_3X_4)}, 2^{H(X_5X_6)}\}$. In this small-scale example, all the pairs are coded in the new variables, hence $H(Y_1) = H(X_1X_2)$, $H(Y_2) = H(X_3X_4)$, $H(Y_3) = H(X_5X_6)$. The dependence within each pair is reflected in their mutual information. Except for the first pair where $I(X_1; X_2) = H(X_1) + H(X_2) - H(Y_1) = 0$, because X_1 does not change and has zero entropy, $I(X_3; X_4) = H(X_3)+H(X_4)-H(Y_2) = 1.3921$ and $I(X_5; X_6) = H(X_5) + H(X_6) - H(Y_3) = 0.5033$ (all the values are in bit). With the further grouping of Y_1 and Y_2 we build a new variable Z_1 that has cardinality nine. Again, the coding is with no loss, i.e $H(Z_1) = H(Y_1Y_2)$ and the mutual information is $I(Y_1; Y_2) = H(Y_1) + H(Y_2) - H(Z_1) = 1.0861$. The last grouping of Z_1 with Y_3 defines V that completes the tree. Again $H(Z_1Y_3) = H(V)$ and $I(Z_1; Y_3) = H(Z_1) + H(Y_3) - H(V) = 0.5033$. Note that the entropy of V is the entropy of the whole set of examples. The grouping order is clearly only one of the many possible ones. We will comment about this issue later in the paper. Figure 3 shown the

X_1	X_2	X_3	X_4	X_5	X_6
B	E	A	T	b	b
B	I	T	b	b	b
B	A	I	T	b	b
B	E	T	b	b	b
B	A	T	b	b	b
B	O	O	T	b	b
B	O	A	T	b	b
B	O	U	G	H	T
B	U	T	b	b	b

$$\begin{array}{lll}
\mathcal{X}_1 = \{\mathrm{B}\}; & \# = \{9\}; & H(X_1) = 0 \\
\mathcal{X}_2 = \{\mathrm{E}, \mathrm{I}, \mathrm{A}, \mathrm{O}, \mathrm{U}\}; & \# = \{2, 1, 2, 3, 1\}; & H(X_2) = 2.1972 \\
\mathcal{X}_3 = \{\mathrm{A}, \mathrm{T}, \mathrm{I}, \mathrm{O}, \mathrm{U}\}; & \# = \{2, 4, 1, 1, 1\}; & H(X_3) = 2.0588 \\
\mathcal{X}_4 = \{\mathrm{T}, \mathrm{b}, \mathrm{G}\}; & \# = \{4, 4, 1\}; & H(X_4) = 1.3921 \\
\mathcal{X}_5 = \{\mathrm{b}, \mathrm{H}\}; & \# = \{8, 1\}; & H(X_5) = 0.5033 \\
\mathcal{X}_6 = \{\mathrm{b}, \mathrm{T}\}; & \# = \{8, 1\}; & H(X_6) = 0.5033
\end{array}$$

Figure 1. The list of examples, the variable alphabets with the number of occurrences (#) and the marginal entropies. Character “b" denotes a “blank".

X_1	X_2	$Y_1(\#)$
B	E	1 (2)
B	I	2 (1)
B	A	3 (2)
B	O	4 (3)
B	U	5 (1)

$H(Y_1) = 2.1972$

X_3	X_4	$Y_2(\#)$
A	T	1 (2)
T	b	2 (4)
I	T	3 (1)
O	T	4 (1)
U	G	5 (1)

$H(Y_2) = 2.0588$

X_5	X_6	$Y_3(\#)$
b	b	1 (8)
H	T	2 (1)

$H(Y_3) = 0.5033$

Y_1	Y_2	$Z_1(\#)$
1	1	1 (1)
2	2	2 (1)
3	3	3 (1)
1	2	4 (1)
3	2	5 (1)
4	4	6 (1)
4	1	7 (1)
4	5	8 (1)
5	2	9 (1)

Z_1	Y_3	$V(\#)$
1	1	1 (1)
2	1	2 (1)
3	1	3 (1)
4	1	4 (1)
5	1	5 (1)
6	1	6 (1)
7	1	7 (1)
8	2	8 (1)
9	1	9 (1)

$H(Z_1) = 3.1699$
$H(V) = 3.1699$

Figure 2. The group occurrences and their associations to the new alphabets with the corresponding entropies.

tree in factor graph form (normal form) [5]. The thin vertical blocks represent the “=" contraints and they are associated to multiple names for the same variable. They act like “buses” because their outgoing messages are simply the product of the incoming ones [6]. The other blocks represent the conditional probability matrices for our generative model. The function that they represent are deterministic because the trasformation from the right to the left has no uncertainty. Vice versa, information from the left to the right is

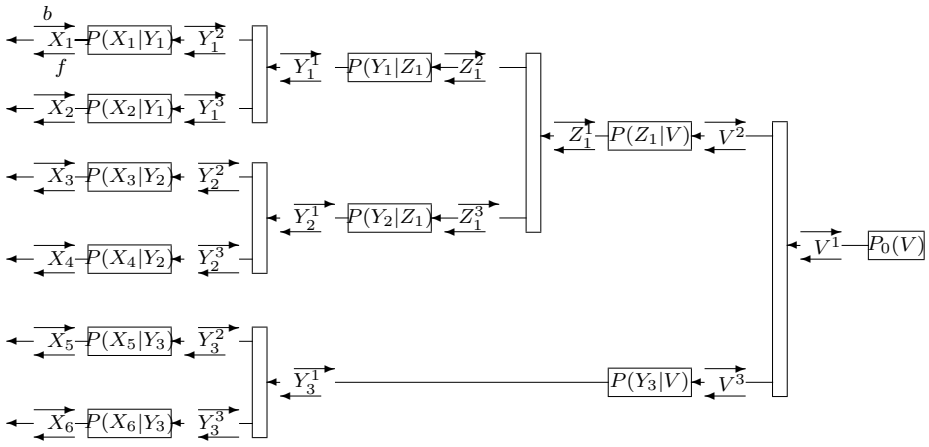

Figure 3. The factor graph in normal form with forward and backward messages shown.

ambiguous and must be resolved with message propagation, as we will see in the following. All the examples are “hard-wired" in the tree in a distributed fashion and message propagation will sort through their structure.

The matrices representing the various blocks are immediately deducted from the tables of Figure 2.

$$P(X_1|Y_1) = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}; \quad P(X_2|Y_1) = P(X_3|Y_2) = I_{5\times 5}; P(X_4|Y_2) = \begin{pmatrix} 1\,0\,0 \\ 0\,1\,0 \\ 1\,0\,0 \\ 1\,0\,0 \\ 0\,0\,1 \end{pmatrix};$$

$$P(X_5|Y_3) = P(X_6|Y_3) = I_{2\times 2}; P(Y_1|Z_1) = \begin{pmatrix} 1\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0 \\ 0\,0\,1\,0\,0 \\ 1\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0 \\ 0\,0\,0\,1\,0 \\ 0\,0\,0\,1\,0 \\ 0\,0\,0\,1\,0 \\ 0\,0\,0\,0\,1 \end{pmatrix}; \quad P(Y_2|Z_1) = \begin{pmatrix} 1\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0 \\ 0\,0\,1\,0\,0 \\ 0\,1\,0\,0\,0 \\ 0\,1\,0\,0\,0 \\ 0\,0\,0\,1\,0 \\ 1\,0\,0\,0\,0 \\ 0\,0\,0\,0\,1 \\ 0\,1\,0\,0\,0 \end{pmatrix};$$

$$P(Z_1|V) = I_{9\times 9}; \quad P(Y_3|V) = \begin{pmatrix} 1\,0 \\ 1\,0 \\ 1\,0 \\ 1\,0 \\ 1\,0 \\ 1\,0 \\ 1\,0 \\ 0\,1 \\ 1\,0 \end{pmatrix}.$$

$P_0(V)$ is uniform on the nine elements since all the examples occur only once.

3. Associative recall

Memory access can be obtained via message propagation using the usual sum-product rules of Bayesian factor graphs [6]. More specifically, we access the memory by injecting backward messages at the terminal nodes. A variable can be: 1. known (instantiated) -> the backward message is delta distribution; 2. completely unknown (erased) -> the backward message is a uniform distribution; 3. known softly -> the backward message is a density. In all cases after message propagation the associative memory responds with a forward message that is a smooth comparison with the stored memories. The final recall is the normalized product of backward and forward message. To see how this idea is applied to our small example, we have implemented a complete message propagation system in Matlab.

Initialization: Before running any inference, the system has to be initialized. Actually the initial configurations could be arbitrary, but proper initialization can produce faster and cleaner convergence. Since the priors about all the variables are hardwired in the system, the forward messages $fX_1, ..., fX_N$ converge to the marginal priors if we inject uniform distributions on $bX_1, ..., bX_N$ and let the system run for a few steps (with arbitrary primordial initial conditions): $fX1 = (1.0000)$, $fX2 = (.2222, .1111, .2222, .3333, .1111)$, $fX3 = (.2222, .4444, .1111, .1111, .1111)$, $fX4 = (.4444, .4444, .1111)$, $fX5 = (.8889, .1111)$, $fX6 = (.8889, .1111)$. Also the other variables fY_{11}, fY_{12}, fY_{13} converge to the prior of Y_1, etc. (not shown for simplicity). The priors for all the variables reflect the number of occurrences shown in the tables of Figure 2 (everything is embedded into the tree). Note that the maximum number of steps is equal to the graph diameter. In our realization, we run the system synchronously and at every clock cycle there is an evolution across each blocks. The maximum number of steps to go from a leaf to another leaf is nine.

Pattern completion: Suppose that we have available only a subset of variables and we need to acces the memory to complete it. For example, assume evidence "?OA???", i.e. only X_2 and X_3 are known. This is equivalent to inject the following backward messages at the leaves: $bX_1 = U_1$; $bX_2 = \delta_5(4)$; $bX_3 = \delta_5(1)$; $bX_4 = U_3$; $bX_5 = U_2$; $bX_6 = U_2$, where U_n denotes the uniform distribution over n elements and $\delta_n(i)$ denotes a distribution over n elements with all zeros, except the ith that is equal to one. After three steps the inference about X_1 and X_4 are already completed with $fX_1 = (1)$ ("B"), $fX_4 = (1, 0, 0)$ ("T"), while $fX5 = (.8889, .1111)$ and $fX6 = (.8889, .1111)$, still show their priors. After 9 steps $fX_5 = (1, 0)$ ("b"), $fX_6 = (1, 0)$ ("b") and the string is perfectly reconstructed. Characters X_1 and X_4 di not need to "call" the tree root because they could be resolved at the lowest hierachical level in only three steps. Vice versa X_5 and X_6 need the information to flow through the upper part of the tree to be estimated. This example shows that local dependence can be exploited for reconstruction and messages do not have necessarily reach the higher hierarchical levels to provide the answer.

Ask for an opinion: More generally, we may have available *a distribution* about a subset of the variables (perhaps coming from another inference system). We can ask our associative memory to verify the coherence of our information with the stored memories. In our example suppose that we inject the following backward messages $bX1 = (1.)$, $bX2 = (0, 0, 0, .5, .5)$, $bX3 = (.5, 0, 0, .5, 0)$, $bX4 = (.2, .6, .2)$, $bX5 = (.7, .3)$, $bX6 = (.1, .9)$. The messages reflect the fact that we are not sure about the values of the

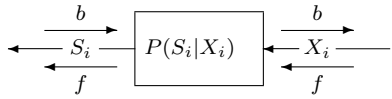

Figure 4. The error model for each leaf variable $i = 1, ..., N$.

six variables. More specifically, we think that X_2 can be either "O" or "U" with equal confidence. Also we infer that X_3 may be either "A" or "U". Similarly more uncertainties are available on the other variables. We let the message propagate in the tree and after convergence we get the following forward messages at the leafs: $fX1 = (1)$, $fX2 = (.3333, .0000, .0000, .6667, .0000)$, $fX3 = (.1129, .3387, .0000, .1129, .4355)$, $fX4 = (1.0000, .0000, .0000)$, $fX5 = (1.0000, .0000)$, $fX6 = (1.0000, .0000)$. The result means that with that information available the string $(X_4X_5X_6)$ must be corrected to (Tbb); X_2 can be either "E" or "U" with different probabilities and X_3 can be "A","T","O" or "U" with different probabilities; X_1 is always "B". The answer that accounts for both opinions can be obtained by computing the normalized products $pX_i = bX_i \cdot fX_i/|bX_i \cdot fX_i|$, $i = 1, ..., N$. In this example $pX1 = (1.0000)$, $pX2 = (.0000, .0000, .0000, 1.0000, .0000)$, $pX3 = (.5, .0000, .0000, .5000, .0000)$, $pX4 = (1.0000, .0000, .0000)$, $pX5 = (1.0000, .0000)$, $pX6 = (1.0000, .0000)$. Only X_3 remains unsolved and can be either "A" or "U" with equal probabilities.

Error correction: The hierarchical memory can be easily adapted to perform also error correction. To this purpose we add to the tree the error model of Figure 4 at the leaves. If we allow each character of $\mathcal{X}_i$ to be confused with any other of the same set with equal probability: $P(S_i|X_i) = (1 - p_e, \frac{p_e}{|\mathcal{X}_i|-1}, ..., \frac{p_e}{|\mathcal{X}_i|-1}; \frac{p_e}{|\mathcal{X}_i|-1}, 1 - p_e, ..., \frac{p_e}{|\mathcal{X}_i|-1}; ... ; \frac{p_e}{|\mathcal{X}_i|-1}, ..., \frac{p_e}{|\mathcal{X}_i|-1}, 1 - p_e)$, $i = 1, ..., N$, where p_e is the error probability. We assume an error probability $p_e = 0.1$ and present to the leafs the string "BOObbT". The backward messages injected are $bS1 = (1)$, $bS2 = (0, 0, 0, 1, 0)$, $bS3 = (0, 0, 0, 1, 0)$, $bS4 = (0, 1, 0)$, $bS5 = (1, 0)$, $bS6 = (0, 1)$. After 12 iterations we get forward and backward messages for $X_1....X_6$ and compute $pX1 = (1.0000)$, $pX2 = (.0132, .0125, .0132, .9487, .0125)$, $pX3 = (.0257, .0499, .0007, .8988, .0250)$, $pX4 = (.9251, .0499, .0250)$, $pX5 = (.9750, .0250)$, $pX6 = (.9750, .0250)$. The memory suggests to shift belief towards "BOOTbb".

4. Discussion

The example presented above shows how on a small Bayesian hierarchy the inference flow may automatically remain confined to low layers, or may "call" higher nodes, according to the necessary dependence to be exploited. This is because in some cases local dependences suffices, while in others it is necessary to look at the information at a more global scale. The potential of the Bayesian trees is then related to its capability, through compound variables, to account for high-order dependences. We believe that the limited success of growing algorithms such as of Chow and Liu's [8] comes from being confined second-order mutual information. Also mixtures of Chow and Liu's net-

works [11] do not seem to go much further. Clearly, the natural sketicism about building large-scale trees is the combinatorial explosion that this seems to produce. However, we maintain that if the tree construction is constrained to small groups and to appropriate partitions, efficient and feasible associative memories can be built. Recall that the information globally exchanged among m variables is described by the mutual information $I(X_1; X_2; ...; X_m) = \sum_{i=1}^{m} H(X_i) - H(X_1 X_2 ... X_m)$. If the variables are independent, the mutual information is null and there is no structure because p is simply the product of m marginal densities. However, in most practical cases the dependence is distributed among the variables in structured form and the cardinality of the typical set for $(X_1 X_2 ... X_m)$ is $2^{H(X_1 X_2 ... X_m)}$ that can be much smaller than $2^{\sum_{i=1}^{m} H(X_i)}$. Efficient algorithm for variable partitions are currently under investigation and will be reported elsewhere.

References

[1] E. T. Jaynes, *Probability Theory : The Logic of Science*, Cambridge University Press, 2003.

[2] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*, 2nd ed. San Francisco: Morgan Kaufmann, 1988.

[3] J. Hawkins, *On Intelligence*, Times Books, 2004.

[4] H. A. Loeliger, "An Introduction to Factor Graphs," *IEEE Signal Processing Magazine*, pp. 28-41, Jan 2004.

[5] G. D. Forney, Jr., "Codes on graphs: normal realizations, " *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 520-548, 2001.

[6] F. Palmieri, "Notes on Factor Graphs," *New Directions in Neural Networks, IOS Press in the KBIES book series, Proceedings of WIRN 2008, Vietri sul mare*, June 2008.

[7] F. Palmieri, G. Romano, P. Salvo Rossi, D. Mattera, "Building a Bayesian Factor Tree From Examples," *Proceedings of the the 2nd International Workshop on Cognitive Information Processing*, 14-16 June, 2010 Elba Island (Tuscany) - Italy.

[8] C. K. Chow and C. N. Liu, "Approximating Discrete Probability Distributions with Dependence Trees," *IEEE Trans. on Information Theory*, Vol. 14, N. 3, May 1968.

[9] J. Cheng, D. A. Bell and W. Liu, "Learning Belief Networks from Data: An Information Theory Based Approach," *Proceedings of the Sixth ACM International Conference on Information and Knowledge Management*, Las Vegas, Nevada, 1997.

[10] Quinlan, J. R., *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, 1993.

[11] M. Meila dn M. I. Jordan, "Learning with Mixtures of Trees," *Journal of Machine Learning Research*, vol. 1, pp. 1-48, October 2000.

[12] E. H. Herskovits and G. F. Cooper, "Kutato: An entropy-driven system for the construction of probabilistic expert systems from databases," *Proceedings of the Conference on Uncertainty in Artificial Intelligence* (pp. 54Ű62), Cambridge, MA, 1990.

[13] D. Heckerman, D. Geiger and D. M. Chickering, "Learning Bayesian Networks: The Combination of Knowledge and Statistical Data," *Machine Learning*, Volume 20, Number 3, September 1995.

[14] A. K. Haynes, "Learning Hidden Structure from Data: A Method for Marginalizing Joint Distributions Using Minimum Cross-Correlation Error," Master's Thesis, University of Illinois at Urbana-Champaign, 1997.

[15] E. Segal, D. Pe'er, A. Regev, D. Koller and N. Friedman, "Learning Module Networks," *Journal of Machine Learning Research*, Vol. 6, pp. 557-588, 2005.