Learning Bayesian Networks by Evolution for Classifier Combination

C. De Stefano, F. Fontanella, A. Scotto di Freca
DAEIMI, Universita' di Cassino
Via G. Di Biasio, 43
03043, Cassino (FR), Italy
{destefano, fontanella, a.scotto}@unicas.it

A. Marcelli
DIIIE, Universita' di Salerno
Via Ponte don Melillo, 1
84084, Fisciano (SA), Italy
amarcelli@unisa.it

Abstract

Combining classifier methods have shown their effectiveness in a number of applications. Nonetheless, using simultaneously multiple classifiers may result in some cases in a reduction of the overall performance, since the responses provided by some of the experts may generate consensus on a wrong decision even if other experts provided the correct one. To reduce these undesired effects, in a previous study, we proposed a combining method based on the use of a Bayesian Network. In this paper, we present an improvement of that method which allows to solve some of the drawbacks exhibited by standard learning algorithms for Bayesian Networks. The proposed method is based on an Evolutionary Algorithm which uses a specifically devised data structure to encode direct acyclic graphs. This data structure allows to effectively implement crossover and mutation operators. The experimental results, obtained by using three standard databases, confirmed the effectiveness of the method.

1. Introduction

Methods for combining classification results provided by different experts represent one of the most widely studied topics in the fields of Machine Learning and Pattern Recognition. Even if the effectiveness of such methods has been theoretically demonstrated [7], their use may result in some cases in a reduction of the overall performance. In fact, assuming that all classifiers to be combined are equally reliable, independently from their specific performance, the responses provided by some of them may generate consensus on a wrong decision, even if other classifiers in the combining pool provided the correct class. This drawback can be partially overcome by introducing some confidence or performance measures to weight the results provided by the classifiers, thus giving more importance in the combining rule to the responses of top performing classifiers. In this case, however, some of the advantages of classifiers combination may be lost because less performing classifiers may correctly recognize those samples which have not been adequately learned by the top one.

In a previous work [2], we have tried to overcome the above problems by considering the whole set of responses provided by the experts for an unknown sample, as representative of the collective behaviour of the combining pool when classifying that sample. In other words, we have reformulated the classifier combination problem as a pattern recognition one, in which each pattern is represented by the set of class labels provided by the experts. Thus, the role of the combiner is that of estimating the conditional probability of each class, given the set of labels provided by the experts for each sample of a training set. A similar approach has been followed in [5], in which the conditional probability distribution of each class has been approximated assuming a third-order dependency. This approach, however, require an exhaustive search of all the dependencies until the third order and, for each of them, the estimate of the corresponding probability distribution. Thus, the use of such an approach may be unfeasible for high numbers of classifiers to be combined and classes to be recognized. In our study, we adopted a Bayesian Network (BN) [8] to automatically infer the joint probability distributions between the outputs of the classifiers and the classes. This choice is motivated by the fact that BN's provide a natural and compact way to encode joint probability distributions through graphical models, and allow to gain understanding about complex problem domain.

This paper represents a further development along this direction, in that we learn the structure of the BN by means of an Evolutionary algorithm, which benefits from a direct encoding scheme of the BN structure, as well as from genetic operators explicitly devised for it.

There are in the literature other few approaches for evolutionary learning of the Bayesian Network structure [10] but their main drawback is the use of data structures for representing DAG's in the form of adjacency matrix: this

978-0-7695-3725-2/09 $25.00 © 2009 IEEE
DOI 10.1109/ICDAR.2009.177

data structure makes difficult to implement genetic operators and does not guarantee that the new generated individuals are DAG's. The effect is twofold: on one hand, it is necessary to verify that new generated individuals satisfy the properties of DAG and this is a time consuming task; on the other hand, the individuals not representing DAG's must be deleted making less efficient the exploration of the search space.

The remainder of the paper is organized as follows: Section 2 illustrates the architecture of the combining method. Section 3 discusses the evolutionary algorithm for evolving DAG's. Section 4 reports the experimental results and some concluding remarks.

2. The architecture of the combiner

Consider the responses $e_1, \ldots, e_L$ provided by a set of L classifiers (experts) for an input sample x in a N class problem, and assume that such responses constitute the input to the combiner, as shown in figure 1. The combiner can be defined as a "higher level" classifier that works on a L-dimensional discrete-values feature space.

It is assumed that the combiner uses a supervised learning strategy, where the learning procedure consists in the observation of the set of responses $e = \{e_1, \ldots, e_L\}$ and of the "true" class label u of a sample x, for computing $p(u|e_1, \ldots, e_L)$. Once this conditional probability has been learned, the combiner provides the output $\widehat{u}$ for each unknown input sample, as the most probable class given the expert observations, by the following expression:

$$\widehat{u} = \arg\max_{u \in C} p\,(u|e_1, ..., e_L) \tag{1}$$

where C is the set of classes. Considering the definition of conditional probability and omitting the terms not depending on the variable u to be maximized, Eq. (1) can be rewritten as:

$$\widehat{u} = \arg\max_{u \in C} p\,(u, e_1, ..., e_L). \tag{2}$$

that involves only the joint probabilities $p\,(u, e_1, ..., e_L)$. Hence the combining problem represented in Eq. (1) is

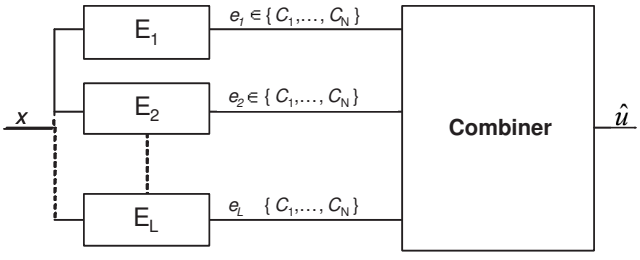

Figure 1. The architecture of our combiner.

equivalent to that of maximizing the joint probability in Eq. (2): this problem may be effectively been solved by using Bayesian Networks.

In the next subsections we will introduce some basic concepts and some mathematical properties of Bayesian Networks, as well as the basic concepts relative to Bayesian Network learning. A more detailed description of the Bayesian Networks theory can be found in [4].

2.1. Bayesian Networks Properties

A BN allows the representation of a joint probability law through the structure of a Direct Acyclic Graph (DAG). The nodes of the graph are the variables, while the arcs are their statistical dependencies. An arrow from the generic node i to node j has the meaning that j is conditionally dependent on i, and we can refer to i as the parent of j. For each node, a conditional probability quantifies the effect that the parents have on that node.

Considering that a DAG describes the statistical dependencies among variables, the conditional probability distribution of a random variable e_i, given all the other, can be simplified as follows:

$$p(\, e_i \,|\, pa_{e_i}, nd_{e_i}) = p(\, e_i \,|\, pa_{e_i}\,) \tag{3}$$

where pa_{e_i} indicates the set of nodes which are parents of node e_i, and nd_{e_i} indicates all the remaining nodes. Eq (3), known as causal Markov property, allows the description of the joint probability of a set of variables $\{u, e_1, \ldots, e_L\}$ as:

$$p\,(u, e_1, \ldots, e_L) = p\,(\, u \,|\, pa_u\,) \prod_{e_i \in L} p\,(\, e_i \,|\, pa_{e_i}\,) \tag{4}$$

In case of a node having no parents, the conditional probability coincides with the a priori probability of that node. It is worth noticing that the node u may be parent of one or more nodes of the DAG. Therefore, it may be useful to divide the L nodes of the DAG in two groups: the first one L_u contains the nodes having the node u among their parents, and the second one $L_{\overline{u}}$ the remaining nodes. With this assumption, the Eq. (4) can be rewritten as:

$$\begin{aligned} p \quad & (u, e_1, \ldots, e_L) = \\ = \quad & p(u|\, pa_u) \prod_{e_i \in L_u} p(e_i|\, pa_{e_i}) \prod_{e_i \in L_{\overline{u}}} p(e_i|\, pa_{e_i}) \end{aligned} \tag{5}$$

Indicating with $\underline{e}_u$ the subset of the variables in $\underline{e}$ which are directly connected to the node u, we can make explicit the following terms in Eq. (5):

$$P(u, \underline{e}_u) = p\,(\, u \,|\, pa_u\,) \prod_{e_i \in L_u} p\,(\, e_i \,|\, pa_{e_i}\,) \tag{6}$$

and

$$Q(\underline{e}) = \prod_{e_i \in L_{\overline{u}}} p\,(\, e_i \,|\, pa_{e_i}\,). \tag{7}$$

Finally, Eq. (6) and Eq. (7) allow to generalize Eq. (2) as follows:

$$\begin{aligned} \widehat{u} &= \arg\max_{u \in C} p(u, e_1, \ldots, e_L) = \\ &= \arg\max_{u \in C} P(u, \underline{e}_u)\, Q(\underline{e}) = \\ &= \arg\max_{u \in C} P(u, \underline{e}_u) \end{aligned} \tag{8}$$

Eq. (9) shows that the term $Q(\underline{e})$ can be discarded since it is constant in the variable u to be maximized. Thus, this approach detects the experts that do not add information to the choice of $\widehat{u}$, or, in other words, selects a reduced set of relevant experts $\underline{e}_u$ whose outputs are actually used by the combiner to provide the final output.

2.2. Learning Bayesian Network

BN estimates the joint probability distribution by a supervised procedure that allows to learn, from the training samples, both the network structure and the parameters of such a probability distribution.

Let us denote with S^h the structure of the DAG, with Θ_S the set of parameters describing the conditional probability distributions of the variables e_i and with D the training set of samples. In our study, each sample of D, corresponding to a pattern x to be classified, is made of both the ordered list of labels provided by the classifiers for that pattern, and the "true" label of x.

Learning structure and parameters from data means finding the values for Θ_S and S^h that best fit the training set D, i.e. that maximize the probability $p(S^h, \Theta_S|D)$:

$$\arg\max_{S^h,\, \Theta_S} p(S^h, \Theta_S|D) \tag{9}$$

Using the Bayes' rule, $p(S^h, \Theta_S|D)$ can be written as:

$$p(S^h, \Theta_S|D) = p(\Theta_S|D, S^h)\; p(S^h|D) \tag{10}$$

and therefore Eq. (9) becomes:

$$\arg\max_{S^h,\, \Theta_S} p(\Theta_S|D, S^h)\; p(S^h|D) \tag{11}$$

The term $p(\Theta_S|D, S^h)$ allows to estimate the conditional probability among variables, while the term $p(S^h|D)$ is aimed at capturing the relationship among the variables and hence the structure of the dependencies in the DAG.

Under the assumptions made in [4], once a DAG structure S^h has been fixed, $p(\Theta_S|D, S^h)$ can be directly computed from the training data. As a consequence, the only term that matter in Eq. (11) is $p(S^h|D)$. This term can be written as:

$$p(S^h|D) = \frac{p(D|S^h)p(S^h)}{p(D)} \tag{12}$$

Under the assumption that every structure S^h is equally likely (i.e. $p(S^h)$ is a constant) and considering that we are interested in evaluating how $p(S^h|D)$ varies depending on the values of S^h, the only term to be considered is the likelihood $p(D|S^h)$. Such term can be written as it follows:

$$p(D|S^h) \;=\; Score(S^h) = \prod_{i=0}^{L} localscore(i) \tag{13}$$

where $localscore(i)$ is a function defined in [4], which describes the local behavior of each variable e_i. It is worth noticing that any change in S^h requires that only the local scores of the nodes affected by the change need to be computed for estimating $Score(S^h)$.

Summarizing, the learning of a Bayesian Network can be performed by finding the DAG structure S^h, which maximizes the function $Score(S^h)$. To solve this problem, we have defined an evolutionary algorithm which encodes a DAG structure in each individual and uses the function $Score(S^h)$ as fitness function. In the next Section, a detailed description of the proposed evolutionary algorithm will be provided.

3. Evolutionary Bayesian Network Learning

Evolutionary Algorithms are probabilistic search techniques inspired by the principle of Natural Evolution well suited for problems where the solution space is very large, multidimensional, complex and discontinuous [3]. They work on a population of individuals, each encoding a possible solutions of the problem at hand. The algorithm starts by randomly generating a population of a certain number of individuals. Then the goodness of each individual as solution for the considered problem is measured by means of a fitness function. After this evaluation process, a new population is generated by choosing in the current population the individuals to be modified by suitable operators. This process is iterated until a termination criterion is not fulfilled.

In our case each individual encodes a DAG structure. Before describing the proposed encoding method, let us briefly recall some basic concepts about DAG's. In the DAG terminology a *source* is a node with no incoming arcs while a *sink* is a node with no outgoing arcs. A DAG must have at least one source and at least one sink. In a DAG structure nodes are partially ordered: a node i comes before a node j if it exists a directed path from i to j. This order relation implies a topological sort, i.e. an ordering of the nodes such that each node comes before all nodes having it as parent. The data structure that we have devised for encoding DAG structures, called *multilist* (ML), consists of two basic lists. The first one, called *main list*, contains all the nodes of the DAG ordered according to the partial ordering previously

defined. This implies that *source* nodes occupy the first positions, while *sink* node, the last positions. Moreover, nodes having both incoming and outgoing arcs are inserted in the *main list* after their parents. To each node of the *main list* is associated a second list called *sublist*, representing the outgoing connections among that node and the other nodes in the DAG. More specifically, if s_i is the *sublist* associated to the $i-th$ element of the *main list*, then it contains information about the outgoing arcs possibly connecting the $i-th$ element and the other elements following it in the *main list*, ordered according to the position of such elements. Since an arc may be present or not, each element of a *sublist* contains a binary information: 1 if the arc exists, 0 otherwise (see figure 2). Note that the length of the *sublists* decreases as the position of the element in the main list increases: assuming that there are N nodes in the DAG, the first sublist contains $(N-1)$ elements, the second one $(N-2)$ elements and so on. In fact, the informations about the arcs connecting a node and the previous ones in the main list are already expressed in the previous sublists. As a consequence, the sublist of the last element in the *main list* is void. Thus a ML has a triangular shape: the base of the triangle is the main list and contains N elements, while the height is represented by the first sublist containing $(N-1)$ elements. It is worth noting that ML have been defined in such a way that they can only represent DAG. This data structure also allows an easy implementation of the genetic operators: we have defined two basic operators, called crossover and mutation. The crossover operator swaps parts of two ML's. In this way, it is possible to generate new solutions by combining parts of previously generated ones. The mutation operator, instead, can modify a single graph in two different ways: (i) swapping two elements of the main list; (ii) adding and/or deleting one or more arcs.

3.1 Crossover Operator

The crossover is applied to two ML's, ML_1 and ML_2, called in the following *parents*, respectively encoding the DAGs S_1 and S_2, and generates two new ML's, $ML_1^{'}$ and $ML_2^{'}$, called *offspring*, respectively encoding the DAG's $S_1^{'}$ and $S_2^{'}$.

The operator is obtained by combining two basic operations that can be applied to a ML. The former, called *t-cut*, splits a generic ML of length N in two ML's, the first one consisting of the first t nodes and the second one of the remaining $N-t$ nodes. The latter operation, called *merge*, given two ML's ML_1 and ML_2, respectively encoding the DAG's S_1 and S_2, yields a new ML of length (N_1+N_2), which encodes a DAG including both the nodes of S_1 and S_2.

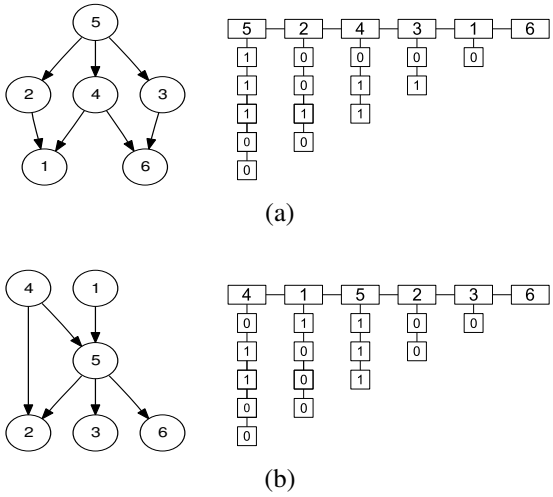

Figure 2. Two DAG's (left) and their encoding multilists (right).

3.2 Mutation Operator

Two mutation operators have been defined. The first one modifies the *main list*, while the second one affects *sublists* elements. In the following these mutations will be called respectively m and s mutation.

The m–mutation performs a permutation on the elements of the main list, as it randomly picks two elements on the main list and swaps their position. The effect is that of changing the dependencies among the variables. We can say that this operator makes a sort of macro-mutation, as it can modify the dependencies among more than two variables. If we consider the DAG in figure 2(b), for instance, the swap of the second and fifth node in the multilist also modifies the dependencies among the nodes 2, 4, 3. Given a multilist, this operator is applied to its *main list* according to a probability value p_m.

The s–mutation actually gives place to a sort of micro-mutation, because it does not modify the structure of the ML, but only the values of the *sublists* elements. Such an operation is applied with probability p_s. For each element of the sublists, p_s represents the probability of flipping its value from 0 to 1, or viceversa. Thus the effect of this operator is that of adding or deleting arcs in the DAG.

4. Experimental Results and Discussion

The proposed method has been tested on three standard databases, namely the NIST, and the Multiple Feature (MF) and the IMAGE database from the UCI Machine Learning Repository, respectively. The first two databases contains handwritten digits, while the third one images with different

textures.

In each experiment we split the samples of each class in three sets: TR1 used for training each single classifier, TR2 for training the Combiner, and TS as test set for performance evaluation. The three sets have been extracted in such a way to be statistically independent.

As with respect to the classifiers, we have used two different schemes: a Back-Propagation neural network (BP) [9] and a Learning Vector Quantization neural network (LVQ) [6]. During a training phase, each classifier was separately trained on TR1. For each database, the pool of experts has been obtained by generating an ensemble of BP nets and an ensemble of LVQ nets.

In the first experiment, we have extracted 3000 samples of each class, and they have been divided into three sets equally numerous to form TR1, TR2 and TS. The samples belonging to each data set have been described by means of two different feature sets, namely the Central Geometrical Moments (CGM) of the binary images up to the 7-th order, and the mean of the pixels belonging to the 8×8 disjoint windows that can be extracted from the binary image (MBI). Thus, each sample is described by means of 33 real variables in the first case, and by means of at most 64 real values in the second one. As with regards to the classifiers, twelve experts have been obtained by using with each feature sets, three randomly initialized nets for each classification scheme, i.e. 3 BP experts using MBI, 3 LVQ using MBI, 3 BP using CGM, and 3 LVQ using CGM.

In the second experiment, the 2000 available samples have been randomly divided into three sets, with TR1 containing 700 samples, and TR2 and TS including 650 samples each. The pool of experts has been obtained by combining each classification scheme with the six feature sets included in the MF Database, totaling twelve experts.

In the last experiment, TR1 was made of 210 samples, 30 per each of the 7 classes, and both TR2 and TS contain 1050 elements, 150 samples per class. The pool of experts has been obtained by combining two ensembles of BP and LVQ nets, each containing 10 different experts obtained by randomly initializing the nets.

Table 1 shows the classification results. The first two columns report, respectively, the results of the worst and the best single expert of the pool, the third column reports the results of the Bayesian Combiner (BN) implemented by using the k2 algorithm [1] to learn the DAG structure. Eventually, the forth column shows the results of our Evolutionary Bayesian Combiner (EvoBN).

The data reported in the Table show that EvoBN improves the performance with respect to BN on all the databases. In particular, it accounts for a reduction of the errors that ranges from 5% to 20%, depending on the database. Even if the results of our combiner constitute a slight improvements with respect to the results of BN, we believe that they are very promising because they confirmed that our evolutionary learning algorithms was able to both explore a very complex search space formed by all possible DAG structures, and to find effective solutions. Finally, it is worth noticing that we have reported in Table 1 only the best results obtained by both EvoBN and BN, even if we have experimentally found that BN exhibits a higher variability in the results obtained in different runs with different random ordering of the variables. EvoBN, on the contrary, provides much more stable results, showing that it less prone to be trapped in local minima of the search space.

Table 1. Comparison of Classification Results

	Worst Expert	Best Expert	BN Combiner	EvoBN Combiner
NIST	88.26% (88.09%)	96.70% (96.68%)	98.65 % (98.70%)	98.76 % (98.86%)
MF	69.67% (64.86)%	96.89% (96.29%)	99.10% (100%)	99.28% (100%)
IMAGE	82.38% (83.14%)	91.00% (91.00%)	91.90% (92.80%)	92.30% (92.84%)

References

[1] G. F. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, October 1992.

[2] C. De Stefano, C. D'Elia, A. Marcelli, and A. Scotto di Freca. Using bayesian network for combining classifiers. In *Proc. of the 14th Int. Conf. on Image Analysis and Processing*, pages 73–80, 2007.

[3] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, January 1989.

[4] D. Heckerman. A tutorial on learning with bayesian networks. Technical report, Learning in Graphical Models, 1995.

[5] H.-J. Kang. Combining multiple classifiers based on third-order dependency for handwritten numeral recognition*1. *Pattern Recognition Letters*, 24(16):3027–3036, 2003.

[6] T. Kohonen. *Self organizing map*. Springer., Berlin, 1995.

[7] L. I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, 2004.

[8] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[9] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[10] M. L. Wong and K.-S. Leung. An efficient data mining method for learning bayesian networks using an evolutionary algorithm-based hybrid approach. *IEEE Trans. Evolutionary Computation*, 8(4):378–404, 2004.