A Bayesian Approach for Combining Ensembles of GP Classifiers

C. De Stefano[1], F. Fontanella[1], G. Folino[2], and A. Scotto di Freca[1]

[1] Università di Cassino
Via G. Di Biasio, 43 02043 Cassino (FR) – Italy
{destefano,fontanella,a.scotto}@unicas.it
[2] ICAR-CNR Istituto di Calcolo e Reti ad Alte Prestazioni
Via P. Bucci 87036 Rende (CS) – Italy
folino@icar.cnr.it

Abstract. Recently, ensemble techniques have also attracted the attention of Genetic Programing (GP) researchers. The goal is to further improve GP classification performances. Among the ensemble techniques, also bagging and boosting have been taken into account. These techniques improve classification accuracy by combining the responses of different classifiers by using a majority vote rule. However, it is really hard to ensure that classifiers in the ensemble be appropriately diverse, so as to avoid correlated errors. Our approach tries to cope with this problem, designing a framework for effectively combine GP-based ensemble by means of a Bayesian Network. The proposed system uses two different approaches. The first one applies a boosting technique to a GP–based classification algorithm in order to generate an effective decision trees ensemble. The second module uses a Bayesian network for combining the responses provided by such ensemble and select the most appropriate decision trees. The Bayesian network is learned by means of a specifically devised Evolutionary algorithm. Preliminary experimental results confirmed the effectiveness of the proposed approach.

1 Introduction

In the last years, in order to further improve classification performance, ensemble techniques [11] have been taken into account in the Genetic Programming (GP) field [8]. The GP approach uses the evolutionary computation paradigm to evolve computer programs, according to a user-defined fitness function. When dealing with classification problems, GP–based techniques exhibited very interesting performance [12]. In this context, the decision tree [14] data structure is typically adopted since it allows to effectively arrange in a tree-structured plans the set of attributes chosen for pattern representation. Successful examples of ensemble techniques applied to GP can be found in [1] [5]. In [7], bagging and boosting techniques have been used for evolving ensembles of decision trees. In [4] a novel GP–based classification system, named *Cellular GP for data Classification* (CGPC), has been presented. Such approach, inspired by cellular automata model, enables a fine-grained parallel implementation of GP. In [5], an extension of CGPC based on the use of two ensemble techniques is presented: the first technique is the Breiman's bagging algorithm [11], while the second one is the AdaBoost.M2 boosting algorithm by Freund and Schapire [11]. The experimental results presented in these papers show that CGPC represents an effective classification algorithm, whose performance have been further improved using ensemble techniques. In this framework, it is generally agreed that a key issue is to ensure that classifiers in the ensemble be appropriately diverse, so as to avoid correlated errors [11]. In fact, as the number of classifiers increases, it may happen that a correct classification provided by some classifiers is overturned by the convergence of other classifiers on the same wrong decision. This event is much more likely in case of highly correlated classifiers and may reduce the performance obtainable with any combination strategy.Classifier diversity for bagging and boosting have been experimentally investigated in [10,9]. The results have shown that these techniques do not ensure to obtain sufficiently diverse classifiers. As regards boosting, in [9] it has been observed that while at first steps highly diverse classifiers are obtained, as the boosting process proceeds classifier diversity strongly decreases.

In a previous work [2] an attempt to solve this problem has been made by reformulating the classifier combination problem as a pattern recognition one, in which the pattern is represented by the set of class labels provided by the classifiers when classifying a sample. Following this approach, the role of the combiner is that of estimating the conditional probability of each class, given the set of labels provided by the classifiers for each sample of a training set. In this way, it is possible to automatically derive the combining rule through the estimation of the conditional probability of each class. It it also possible to identify redundant classifiers, i.e. classifiers whose outputs do not influence the output of the combiner: the behavior of such classifiers is very similar to that of other classifiers in the ensemble and they may be eliminated without affecting the overall performance of the combiner, thus overcoming the main drawback of the combining methods discussed above. In [3] a Bayesian Network (BN) [13] has been used to automatically infer the joint probability distributions between the outputs of the classifiers and the classes. The BN learning has been performed by means of an evolutionary algorithm using a direct encoding scheme of the BN structure. Such encoding scheme is based on a specifically devised data structure, called *Multilist*, which allows an easy and effective implementation of the genetic operators.

In this paper we present a new classification system that merges the two aforementioned approaches. We have combined the BoostCGPC algorithm [5], which produces a high performing ensemble of decision tree classifiers, with the BN based approach to classifier combination. Our system tries to exploit the advantages provided by both techniques and allows to identify the minimum number of independent classifiers able to recognize the data at hand.

In order to assess the effectiveness of the proposed system, several experiments have been performed. More specifically, four data sets, having different sizes, number of attributes and classes have been considered. The results have been compared with those obtained by the BoostCGPC approach using a weighted majority vote combining rule.

The remainder of the paper is organized as follows. In Section 2 the BN based combining technique is presented. In Section 3 the architecture of the proposed system is described. In Section 4 the experimental results are illustrated, while discussion and some concluding remarks are eventually left to Section 5.

2 Bayesian Networks for Combining Classifiers

The problem of combining the responses provided by a set of classifiers can be also faced considering the joint probability $p\,(c, e_1, ..., e_L)$, where e_i represents the response of the i–th classifier and c is the class to be assigned to the sample taken into account. The problem of computing the joint probability $p\,(c, e_1, ..., e_L)$ may be effectively solved by using a Bayesian Network (BN). In particular, in [2], a BN has been used for for combining the responses of more classifiers in a multi expert system.

A BN is a probabilistic graphical model that allows the representation of a joint probability distribution of a set of random variables through a Direct Acyclic Graph (DAG). The nodes of the graph correspond to variables, while the arcs characterize the statistical dependencies among them. An arrow from a node i to a node j has the meaning that j is conditionally dependent on i, and we can refer to i as a *parent* of j. For each node, a conditional probability quantifies the effect that the parents have on that node. Once the statistical dependencies among variables have been estimated and encoded in the DAG structure, each node e_i is associated with a conditional probability function exhibiting the following property:

$$p(\,e_i\,|\,pa_{e_i}, nd_{e_i}) = p(\,e_i\,|\,pa_{e_i}\,) \tag{1}$$

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. This property allows the description of the joint probability of a set of variables $\{c, e_1, \ldots, e_L\}$ as follows:

$$p\,(c, e_1, \ldots, e_L) = p\,(\,c\,|\,pa_c\,) \prod_{e_i \in E} p\,(\,e_i\,|\,pa_{e_i}\,) \tag{2}$$

It is worth noticing that the node c may be parent of one or more nodes of the DAG. Therefore, it may be useful to divide the e_i nodes of the DAG in two groups: the first one, denoted as E_c, contains the nodes having the node c among their parents, and the second one, denoted as $E_{\overline{c}}$, the remaining ones. With this assumption, Eq. (2) can be rewritten as:

$$p\,(c, e_1, \ldots, e_L) = p\,(\,c\,|\,pa_c\,) \prod_{e_i \in E_c} p\,(\,e_i\,|\,pa_{e_i}\,) \prod_{e_i \in E_{\overline{c}}} p\,(\,e_i\,|\,pa_{e_i}\,) \tag{3}$$

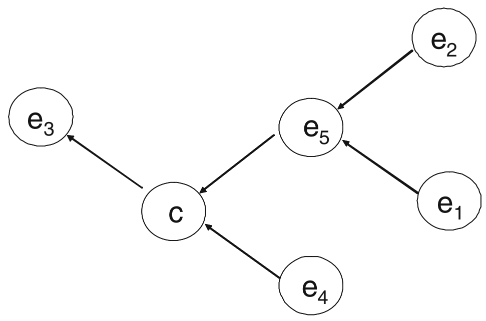

Fig. 1. An example of a BN. The DAG structure induces the factorization of the joint probability $p(c, e_1, e_2, e_3, e_4, e_5) = p(e_3|c)p(c|e_4, e_5)p(e_5|e_1, e_2)p(e_1)p(e_2)p(e_4)$. In this case $E_c = \{e_3\}$, $E_{\overline{c}} = \{e_1, e_2, e_4, e_5\}$ and hence $\widehat{c} = \arg\max_{c \in C} p(e_3|c)p(c|e_4, e_5)$.

Since the third term in Eq. (3) does not depend on c, Eq. (6) assumes the form:

$$\widehat{c} = \arg\max_{c \in C} p(c, e_1, \ldots, e_L) = \arg\max_{c \in C} p\,(\,c\,|\,pa_c\,) \prod_{e_i \in E_c} p\,(\,e_i\,|\,pa_{e_i}\,) \tag{4}$$

For instance, the BN reported in Fig. 1 considers only the responses of the experts e_3, e_4 and e_5, while the experts e_1 and e_2 are not taken into account. Thus, this approach allows to detect a reduced set of relevant experts, namely the ones connected to node c, whose responses are actually used by the combiner to provide the final output, while the set $E_{\overline{c}}$ of experts, which do not add information to the choice of $\widehat{c}$, are discarded.

Using a BN for combining the responses of more classifiers requires that both the network structure, which determines the statistical dependencies among variables, and the parameters of the probability distributions be learned from a training set of examples. The structural learning, is aimed at capturing the relation between the variables, and hence the structure of the DAG. It can be seen as an optimization problem which requires the definition of a search strategy in the space of graph structures, and a scoring function for evaluating the effectiveness of candidate solutions. A typical scoring functions is the posterior probability of the structure given the training data. More formally, if D and S^h denote the training set and the structure of a candidate BN, respectively, the scoring function to be maximized is the likelihood of D given the structure S^h. Once the DAG structure S^h has been determined, the parameters of the conditional probability distributions are computed from training data.

The exhaustive search of the BN structure which maximizes the scoring function is a NP-hard problem in the number of variables. This is the reason why standard algorithms search for suboptimal solutions by maximizing at each step a local scoring function which takes into account only the local topology of the DAG. Moving from these considerations, we have proposed an alternative approach in which the structure of the BN is learned by means of an Evolutionary

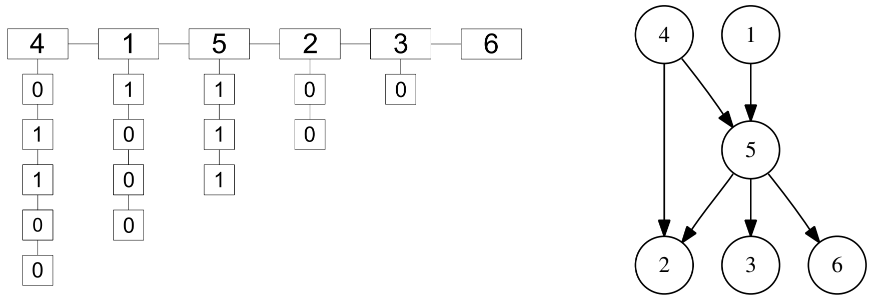

Fig. 2. A multilist (right) and the encoded DAG's(left)

algorithm, using a direct encoding scheme. The algorithm is based on a specifically devised data structure for encoding DAG, called *multilist* (ML), which consists of two basic lists. The first one, called *main list*, contains all the nodes of the DAG arranged in such a way that source nodes occupy the first positions, and sink node, the last ones. 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). This ML data structure allows an easy and effective implementation of genetic operators. Moreover, since the above definition ensures that a ML intrinsically represents a DAG structure, the application of such operators always produces valid offspring.

As regards the genetic operators, we have defined two mutation operators which can modify a ML in two different ways. The m mutation changes a ML by swapping two elements of the main list, whereas the s mutation adds and/or deletes one or more arcs in a sub list.
The m–mutation performs a permutation on the elements of the main list, but leaves unchanged the connection topology of the ML. This mutation consists of two steps:

(i) randomly pick two elements in the main list and swap their positions.
(ii) modify sublist elements in such a way to restore the connection topology as it was before the step (i).

It is worth noticing that the m–mutation generates a new ordering of the variables, which modifies the directions of the existing arcs in the DAG, but preserves dependencies between variables. If we consider the DAG in figure 2, for instance, the swap between the second and the fourth node in the main list changes only the directions of the arcs connecting the couples of nodes $(1, 5)$ and $(5, 2)$. This operator is applied according to a predefined probability value p_m.

The s–mutation, instead, modifies the values of the *sublist* elements. For each element of the sublists, p_s represents the probability of changing its value from 0 to 1, or vice versa. Thus the effect of this operator is that of adding or deleting arcs in the DAG. Such an operation is applied with probability p_s. Further details about ML data structure and the genetic operators can be found in [3].

The evolutionary algorithm starts by randomly generating an initial population of P individuals. Afterward, the fitness of each individual is evaluated by computing the scoring function. At each generation, the best e individuals are selected and copied in the new population in order to implement an elitist strategy. Then, the tournament is used to select $(P - e)$ individuals and the m and s mutation operators are applied to each selected individual according to the probabilities p_m and p_s, respectively. Finally these individuals are added to the new population. This process is repeated for n_g generations.

3 System Architecture

The proposed system consists of two main modules: the first one builds an ensemble of decision tree classifiers (experts); the second one implements the combining rule that produces the final classification result of the whole system.

The first module, called *BoostCGPC*, builds decision tree [14] using a Genetic Programming (GP) technique [8], which is an evolutionary computation-based technique able to evolve computer programs according to a user-defined fitness function. The output ensemble is learned by implementing a modified version of the algorithm *AdaBoost.M2* [6]. Such an implementation allows to run the algorithm on distributed memory parallel computer, making the system able to deal with large data sets. Further details about this algorithm can be found in [5].

The second module uses the approach described in the previous section for combining the responses provided by the classifiers making up the ensemble built in the first module. More specifically, let us denote with N the number of classes to be discriminated, with L the number of decision tree classifiers included the ensemble and with $E = \{e_1, \ldots, e_L\}$ the set of responses provided by such classifiers for a given input sample. Let us assume that such responses constitute the input to the combiner module. In this module, the combining technique operates as a "higher level" classifier, working on a L-dimensional discrete-valued feature space, which is trained by using a supervised learning procedure. This procedure requires to observe both the "true" class label c, and the set of responses provided by the classifiers for each sample of a training set, in order to estimate the conditional probability $p(c|e_1, \ldots, e_L)$. Once this conditional probability has been learned, the combiner evaluates the most probable class $\widehat{c}$ of an unknown input sample, given the expert observations, as follows:

$$\widehat{c} = \arg\max_{c \in C} p\left(c|e_1, ..., e_L\right) \tag{5}$$

where C is the set of classes. Considering the definition of conditional probability and omitting the terms not depending on the variable c to be maximized, Eq. (5) can be rewritten as:

$$\widehat{c} = \arg\max_{c \in C} p\left(c, e_1, ..., e_L\right). \tag{6}$$

that involves only the joint probabilities $p\left(c, e_1, ..., e_L\right)$. This problem may be effectively solved by using a Bayesian Network, according to the approach outlined in the previous section.

Note that the devised system recognizes unknown samples using a two–step procedure: (i) the feature values describing the unknown sample are provided to each of the ensemble classifiers built by the BoostCGPC module; (ii) the set of responses produced is given in input to the BN module. Such module labels the sample with the most likely class, among those of the problem at hand, given the responses collected by the first module [1]. Note that, for some samples, the BN is not able to assign them a label. This case occurs when two or even more classes are equally likely. In this case, the unknown sample is labeled using the majority vote rule, applied to the first module responses.

4 Experimental Results

The proposed approach has been tested on four data sets: *Census*, *Segment*, *Adult* and *Phoneme*. The size and class distribution of these data sets are described in Table 1. They present different characteristics in the number and type (continuous and nominal) of attributes, two classes versus multiple classes and number of samples. In particular, Census and Adult, are real large data set containing census data collected by the U.S. Census Bureau. The Segment contains image data. Finally, the Phoneme data set contains data distinguishing between nasal and oral vowels. For each data set, two statistically independent sets of equal size, have been built randomly splitting the samples of each class. The first set has been used for training, while the second set for the test.

All the experiments were performed on a Linux cluster with 16 Itanium2 1.4GHz nodes each having 2 GBytes of main memory and connected by a Myrinet

Table 1. The data sets used in the experiments

datasets	**attr.**	**samples**	**classes**
Adult	14	48842	2
Census	4	299285	2
Phoneme	5	5404	2
Segment	36	2310	6

[1] Note that the second step does not require any further computation with respect to the Majority Voting rule. In fact, it only needs to read tables storing class probabilities.

high performance network. As regards the boostGCPC algorithm, it has been run on five nodes, using standard GP parameters and a population of 100 individuals for node. The original training set has been partitioned among the nodes and respectively 5 and 10 rounds of boosting, with 100 generations for round, have been used to produce respectively 25 and 50 classifiers on 5 nodes. It is worth to remember the algorithm produce a different classifier for each round on each node.

All results were obtained by averaging over 30 runs. For each run of the BoostCGPC module, a run of the BN module has been carried out. Each BN run has been performed by using the responses, on the whole training set, provided by the classifiers learned in the corresponding BoostCGPC run. The results on the test set has been obtained by first submitting each sample to the learned decision trees ensemble. The ensemble responses have been then provided to the learned BN. Finally, the BN output label has been compared with the true one of that sample.

The results achieved by the our approach (hereafter BN-BoostCGPC) have been compared with those obtained by BoostCGPC approach, which uses the Weighted Majority rule for combining the ensemble responses. The comparison results are shown in Tab. 2. The second column shows the ensembles (25 or 50 classifiers), while the columns 3 and 6 shows the training set errors of the BoostCGPC and BN-BoostCGPC, respectively. Similarly, the columns 4 and 7 show the test set errors of the BoostCGPC and BN-BoostCGPC, respectively. The columns 5 and 8 contain the number of classifiers actually used by both approaches. It is worth noticing that for the BoostCGPC approach such number equals the number of classifier making up the ensemble (25 or 50). The BN-BoostCGPC, instead, uses only the classifiers that are directly connected to the output node in the DAG.

In order to statistically validate the comparison results, we performed the two–tailed t–test($\alpha = 0.05$) over the 30 carried out runs. The values in bold in the test set error columns highlight, for each ensemble, the results which are significantly better according to the two–tailed t–test. The proposed approach achieves better performance on the majority of the considered ensembles while,

Table 2. Comparison results

Datasets	**ens.**	**BoostCGPC**			**BN-BoostCGPC**		
		Train	**Test**	**# sel.**	**Train**	**Test**	**# sel.**
Adult	**25**	15.90	16.94	25	15.85	**16.28**	3.4
	50	16.88	18.23	50	16.55	**16.99**	3.8
Segment	**25**	11.82	12.69	25	10.82	**11.68**	2.9
	50	10.39	12.06	50	10.34	11.99	2.9
Phoneme	**25**	16.41	18.87	25	17.70	19.23	3.2
	50	16.90	20.04	50	17.23	19.51	7.8
Census	**25**	8.81	8.89	25	5.14	**5.27**	4.3
	50	8.88	9.07	50	5.27	**5.37**	3.9

in the remaining cases, the performance are comparable. It is also worth noticing that the most significant improvements have been obtained on Adult and Census data sets, which are the largest ones among those considered. This result is due to the fact that larger data sets allow the BN learning process to better estimate the conditional probabilities to be modeled. Finally, is worth to remark that the results of our system are always achieved by using only a small number of the available classifiers.

5 Conclusions and Future Work

We presented a novel approach for improving the performance of derivation tree ensembles, learned by means of a boosted GP algorithm. The approach consists of two modules, the first one uses a boosted GP algorithm to generate ensembles of decision trees. The second module, instead, employs Bayesian networks to effectively combine the responses provided by the ensemble decision trees.

The experimental results have shown that the proposed system further improves the performance achieved by using the boosted GP algorithm. Moreover, such performances are obtained by using a reduced number of classifiers. Finally, the presented approach seems to be particularly suited to deal with very large data sets. Future work will include testing on several ensembles, having a different number of classifiers. Furthermore, larger data sets will be taken into account, to further investigate the capability of the presented system to deal with very large data sets.

References

1. Cantú-Paz, E., Kamath, C.: Inducing oblique decision trees with evolutionary algorithms. IEEE Transaction on Evolutionary Computation 7(1), 54–68 (2003)
2. De Stefano, C., D'Elia, C., Scotto di Freca, A., Marcelli, A.: Classifier combination by bayesian networks for handwriting recognition. Int. Journal of Pattern Rec. and Artif. Intell. 23(5), 887–905 (2009)
3. De Stefano, C., Fontanella, F., Marrocco, C., Scotto di Freca, A.: A hybrid evolutionary algorithm for bayesian networks learning: An application to classifier combination. In: EvoApplications (1). pp. 221–230 (2010)
4. Folino, G., Pizzuti, C., Spezzano, G.: A cellular genetic programming approach to classification. In: Proc. Of the Genetic and Evolutionary Computation Conference (GECCO 1999), pp. 1015–1020. Morgan Kaufmann, Orlando (1999)
5. Folino, G., Pizzuti, C., Spezzano, G.: Gp ensembles for large-scale data classification. IEEE Transaction on Evolutionary Computation 10(5), 604–616 (2006)
6. Freund, Y., Shapire, R.: Experiments with a new boosting algorithm. In: Proceedings of the 13th Int. Conference on Machine Learning, pp. 148–156 (1996)
7. Iba, H.: Bagging, boosting, and bloating in genetic programming. In: Proc. Of the Genetic and Evolutionary Computation Conference (GECCO 1999), pp. 1053–1060. Morgan Kaufmann, Orlando (1999)
8. Koza, J.R.: Genetic Programming: On the Programming of Computers by means of Natural Selection. MIT Press, Cambridge, MA (1992)

9. Kuncheva, L., Shipp, C.: An investigation into how adaboost affects classifier diversity. In: Proc. of IPMU (2002)
10. Kuncheva, L., Skurichina, M., Duin, R.P.W.: An experimental study on diversity for bagging and boosting with linear classifiers. Information Fusion 3(4), 245–258 (2002)
11. Kuncheva, L.I.: Combining Pattern Classifiers: Methods and Algorithms. Wiley Interscience, Hoboken (2004)
12. Nikolaev, N., Slavov, V.: Inductive genetic programming with decision trees. In: Proceedings of the 9th International Conference on Machine Learning, Prague, Czech Republic (1997)
13. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Francisco (1988)
14. Quinlan, J.R.: C4.5 Programs for Machine Learning. Morgan Kaufmann, San Mateo (1993)