Boosting technique for Combining Cellular GP Classifiers

Gianluigi Folino, Clara Pizzuti and Giandomenico Spezzano

ICAR-CNR,
c/o DEIS, Univ. della Calabria
87036 Rende (CS), Italy
{folino,pizzuti,spezzano}@icar.cnr.it

Abstract. An extension of Cellular Genetic Programming for data classification with the boosting technique is presented and a comparison with the bagging-like majority voting approach is performed. The method is able to deal with large data sets that do not fit in main memory since each classifier is trained on a subset of the overall training data. Experiments showed that, by using a sample of reasonable size, the extension with these voting algorithms enhances classification accuracy at a much lower computational cost.

1 Introduction

The explosive growth of information in different domains has spurred the development of data mining techniques [3] for knowledge extraction from massive data sets. The availability of fast, efficient, and accurate data mining algorithms, able to deal with this huge amount of data, too large to fit into the main memory of computers, is becoming a pressing request. To obtain fast methods several parallel data mining algorithms were realized, such as parallel decision trees [18, 17], and parallel association rules [20]. On the other hand, to improve the accuracy of any learning algorithm, ensemble techniques, that combine the prediction of multiple classifiers, each trained on a different training set obtained by means of resampling, have been introduced.

Bagging [2] and boosting [8] are well known ensemble techniques that repeatedly run a weak learner on different distributions over the training data. Both methods build *bags* of data of the same size of the original data set by applying random sampling with replacement. Unlike bagging, boosting tries to concentrate on harder examples by adaptively changing the distributions of the training set on the base of the performance of the previous classifiers. It has been shown that bagging and boosting improve the accuracy of decision tree classifiers [2, 14, 1].

Genetic programming (GP)[10] has showed to be a particularly suitable technique to deal with the task of data classification [7, 13, 15, 11, 4] by evolving decision trees. Genetic Programming extended by means of ensemble techniques [9, 5] enhances classification accuracy of GP. Genetic programming based classifiers,

however, involve a lot of computation and their performances may drastically degrade when applied to large problems because of the intensive computation of fitness evaluation of each individual in the population. High performance computing is an essential component for increasing the performances and obtaining large-scale efficient classifiers [19, 6]. To this purpose, several approaches have been proposed. The different models used for distributing the computation and to ease parallelize genetic programming, cluster around two main approaches [19]: the well-known *island model* and the *cellular model*. In the island model several isolated subpopulations evolve in parallel, periodically exchanging by migration their best individuals with the neighboring subpopulations. In the cellular model each individual has a spatial location on a low-dimensional grid and the individuals interact locally within a small neighborhood. The model considers the population as a system of active individuals that interact only with their direct neighbors. Different neighborhoods can be defined for the cells and the fitness evaluation is done simultaneously for all the individuals. Selection, reproduction and mating take place locally within the neighborhood. A parallel approach to build predictors speeds up the generation process and, at the same time, allows to deal with large data sets.

Cellular Genetic Programming for data classification ($CGPC$) enhanced with an ensemble bagging-like ($BagCGPC$) technique has been presented in [5] and showed to enhance both the prediction accuracy and the running time of $CGPC$.

In this paper we extend $CGPC$ with a voting classification scheme based on the boosting technique, and present an experimental comparison of $CGPC$ with bagging and boosting voting schemes. The approach can deal with large data sets that do not fit in main memory since each classifier is trained on a subset of the overall training data. Experiments showed that the extension of $CGPC$ with these voting algorithms enhances both accuracy and execution time. In fact, higher accuracy can be obtained by using a sample of reasonable size at a much lower computational cost. The algorithm could also be used to classify distributed data which cannot be merged together. For example, in applications that deal with proprietary, privacy sensitive data, where it is not permitted moving raw data from different sites to a single central location for mining.

The paper is organized as follows. Next section describes Bagging and Boosting algorithms. Section 3 presents the extension of cellular genetic programming with the Boosting technique. In section 4, finally, the results of the method on some standard problems are presented.

2 Ensemble techniques

Let $S = \{(x_i, y_i)|i = 1, \ldots, N\}$ be a training set where x_i, called example, is an attribute vector with m attributes and y_i is the class label associated with x_i. A predictor, given a new example, has the task to predict the class label for it. Ensemble techniques build T predictors, each on a different subset of the training set, then combine them together to classify the test set.

Bagging (bootstrap aggregating) was introduced by Breiman in [2] and it is

based on bootstrap samples (replicates) of the same size of the training set S. Each bootstrap sample is created by uniformly sampling instances from S with replacement, thus some examples may appear more than once while others may not appear in it. T bags $B_1, \ldots, B_T$ are generated and T classifiers $C^1, \ldots, C^T$ are built on each bag B_i. The number T of predictors is an input parameter. A final classifier classifies an example by giving as output the class predicted most often by $C^1, \ldots, C^T$, with ties solved arbitrarily.

Boosting was introduced by Schapire [16] for boosting the performance of any "weak" learning algorithm, i.e. an algorithm that "generates classifiers which need only be a little bit better than random guessing" [8]. The boosting algorithm, called *AdaBoost*, adaptively changes the distribution of the sample depending on how difficult each example is to classify. Given the number T of trials to execute, T weighted training set $S_1, S_2, \ldots, S_T$ are sequentially generated and T classifiers $C^1, \ldots, C^T$ are built to compute a weak hypothesis h_t. Let w_i^t denote the weight of example x_i at trial t. At the beginning $w_i^1 = 1/n$ for each x_i. At each trial $t = 1, \ldots, T$, a weak learner C^t, whose error ϵ^t is bounded to a value strictly less than $1/2$, is built and the weights of the next trial are obtained by multiplying the weight of the correctly classified examples by $\beta^t = \epsilon^t/(1-\epsilon^t)$ and renormalizing the weights so that $\Sigma_i w_i^{t+1} = 1$. Thus "easy" examples get a lower weight, while "hard" examples, that tend to be misclassified, get higher weights. This induces AdaBoost to focus on examples that are hardest to classify. The boosted classifier gives the class label y that maximizes the sum of the weights of the weak hypotheses predicting that label, where the weight is defined as $ln(1/\beta^t)$. Freund and Schapire [8] showed theoretically that AdaBoost can decrease the error of any weak learning algorithm and introduced two versions of the method. In the next section we present the extension of GP by using AdaBoost.M1.

Regarding the application of ensemble techniques to Genetic Programming, Iba in [9] proposed to extend Genetic Programming to deal with bagging and boosting. A population is divided in a set of subpopulations and each subpopulation is evolved on a training set sampled with replacement from the original data. Best individuals of each subpopulation participate to voting to give a prediction on the testing data. Experiments on some standard problems using ten subpopulations showed the effectiveness of the approach. Another extension of Cellular Genetic Programming for data classification to induce an ensemble of predictors was presented in [5]. Each classifier was trained on a different subset of the overall data, then they were combined to classify new tuples by applying a simple majority voting algorithm, like bagging. Results on a large data set showed that the ensemble of classifiers trained on a sample of the data obtains higher accuracy than a single classifier that uses the entire data set at a much lower computational cost.

3 BoostCGPC

Boost Cellular Genetic Programming Classifier (BoostCGPC), is described in figure 1. Given the training set $S = \{(x_1, y_1), \ldots (x_N, y_N)\}$ and the number P of processors to use to run the algorithm, we partition the population in P subpopulations, one for each processor and draw P samples from S of size $n < N$. Each subpopulation is evolved for k generations and trained on its local sample by running $CGPC$. To take advantage of the cellular model of genetic programming, subpopulations are not independently evolved, but they exchange the outmost individuals in an asynchronous way. On each processor at each generation, every tree undergoes one of the genetic operators (reproduction, crossover, mutation) depending on the probability test. If crossover is applied, the mate of the current individual is selected as the neighbor having the best fitness, and the offspring is generated. The current string is then replaced by the best of the two offspring if the fitness of the latter is better than that of the former. After k generations, the individual with the best fitness is selected for participating to vote. In fact the P best individuals of each subpopulation are exchanged among the P subpopulations and constitute the ensemble of predictors that will determine the weights of the examples for the next round.

Figure 2 illustrates the basic framework for the parallel implementation of the $BoostCGPC$ algorithm on a distributed memory parallel computer. We assume that each training sample $S_i, i = 1, \ldots, P$ resides on a different processor within the parallel computer. We use the diffusion model of GP to parallelize in a natural way the implementation of $BoostCGPC$. The size of each subpopulation $Q_i, i = 1, \ldots, P$ present on a node, must be greater than a threshold determined from the granularity supported by the processor. Each processor, using a training sample S_i and a subpopulation Q_i implements a classifier process $CGPC_i$ as a learning algorithm and generates a classifier. For efficiency reasons, the individuals within a subpopulation are combined into a single process that sequentially updates each individual. This reduces the amount of internal communication on each process, increasing the granularity of the application. Communication between processors is local and asynchronous. The configuration of the structure of the processors is based on a ring topology and a classifier process is assigned to each. During the boosting rounds, each classifier process maintains the local vector of the weights that directly reflect the prediction accuracy on that site. At each boosting round the hypotheses generated by each of these classifiers ($CGPC_i$ in Figure 2) are combined to produce the ensemble of predictors. Then, the ensemble is broadcasted at each classifier process to locally recalculate the new vector of the weights and a copy of the ensemble is stored in a repository. After the execution of the fixed number T of boosting rounds, the classifiers stored in the repository are used to evaluate the accuracy of the classification algorithm. Note that, the algorithm can also be used to classify distributed data which cannot be merged together. For example, in applications that deal with proprietary, privacy sensitive data, where it is not permitted moving raw data from different sites to a single central location for mining.

Given $S = \{(x_1, y_1), \ldots (x_N, y_N)\}$, $x_i \in X$
with labels $y_i \in Y = \{1, 2, \ldots, k\}$, and a population Q of size q
For j = 1, 2, ..., P (for each processor in parallel)
Draw a sample S_j with size n for processor j
Initialize the weights $w_i^1 = \frac{1}{N}$ for $i = 1, \ldots, n$,
where n is the number of training examples on each processor j.
Initialize the subpopulation Q_i, for $i = 1, \ldots, P$
with random individuals
end parallel for
For t = 1,2,3, ..., T
For j = 1, 2, ..., P (for each processor in parallel)
Train $CGPC$ on the sample S_j using a weighted
fitness according to the distribution w^t
Compute a weak hypothesis $h_{j,t} : X \rightarrow Y$
Exchange the hypotheses $h_{j,t}$ among the P processors
let $D_i = 1$ if arg max $h_{j,t}(x_i) \neq y_i$
0 otherwise
Compute the error $\epsilon_j^t = \sum_i^n w_i^t D_i$
if $\epsilon_j^t \geq 1/2$ break loop
Set $\beta_j^t = \epsilon_j^t / (1 - \epsilon_j^t)$,
Update the weights $w^t : w_i^{t+1} = 2(1 - \epsilon_j^t) \times w_i^t$ if $h_{j,t}(x_i) = y_i$
$2\epsilon_j^t \times w_i^t$ otherwise
end parallel for
end for t
output the hypothesis :
$h_f = arg\ max\ (\sum_j^p \sum_t^T log(\frac{1}{\beta_j^t}) D_j^t)$
where $D_j^t = 1$ if $h_{j,t}(x_i) = y_i$, 0 otherwise

Fig. 1. The algorithm parallel BoostCGPC

4 Experimental Results

In this section we compare *BagCGPC*, *BoostCGPC* and classical *CGPC* using some well known data sets taken from the UCI Machine Learning Repository [12]. The data sets are described in table 1 and present different characteristics in the number and type (continuous and nominal) of attributes, two classes versus multiple classes and number of tuples. In particular, the last two are real large data sets. *Cens* contains weighted census data extracted from the 1994 and 1995 current population surveys conducted by the U.S. Census Bureau and *CovType* comprises data representing the prediction of forest cover type from cartographic variables determined from US Forest Service and US Geological Survey. The experiments were performed on a Linux cluster with 16 dual-processor 1,133 Ghz Pentium III nodes having 2 Gbytes of memory connected by Myrinet and running Red Hat v7.2.

The parameters used for the experiments are shown in table 2. All results were obtained by averaging 10-fold cross-validation runs. In order to do a fair

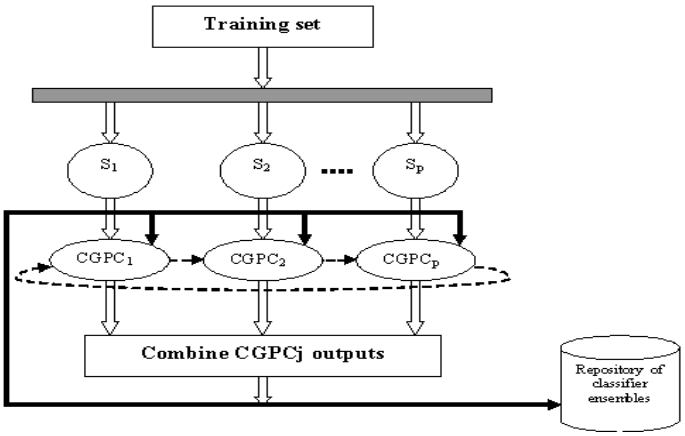

Fig. 2. Implementation of *BoostCGPC* on a distributed memory parallel computer.

Table 1. Data sets used in the experiments

Dataset	Attr.	Tuples	Classes
Adult	14	48842	2
SatImage	36	6435	6
Segment	20	2310	7
Census	42	299285	2
Covtype	54	581012	7

comparison among *CGPC*, *BagCGPC*, and *BoostCGPC* we used 5 processors for all the three algorithms, population size 500 and number of generations 500 for *CGPC*. To obtain the same parameters, *BagCGPC* was executed 5 times on five processors in parallel, with population size 100 on each processor (for a total size of 100 $\times$ 5=500) and number of generations 100 (for a total number of generations 100 $\times$ 5=500), thus generating 25 classifiers. On the other hand, the number T of rounds of *BoostCGPC* was 5, again on 5 processors, population size 100 on each processor, number of generations 100 for each round, thus generating the same total population size, number of generations, and number of classifiers, i.e. 500, 500, and 25, respectively. In table 3 we report the mean error rate over the 10-fold-cross-validations, execution time in seconds, and size of the classifiers. For these experiments *CGPC* used the complete data sets for training, while both *BagCGPC* and *BoostCGPC* employed only the 20% of the overall data sets. Note that in the case of the ensembles, the size is the sum of the 25 classifiers composing the ensemble; thus the average size of a single tree can be obtained dividing this size by 25. From the table we can observe that *BagCGPC* and *BoostCGPC* always obtain better results than *CGPC* in terms of mean error, time and average tree size, the only exception being an

Table 2. Main parameters used in the experiments

Name	Value
max_depth_for_new_trees	6
max_depth_after_crossover	17
max_mutant_depth	2
grow_method	RAMPED
selection_method	GROW
crossover_func_pt_fraction	0.7
crossover_any_pt_fraction	0.1
fitness_prop_repro_fraction	0.1
parsimony_factor	0

error of 8.33 of *BoostCGPC* for the *Cens* data set and an error of 38.03 of *BagCGPC* for the *CovType* data set. Considering that the ensemble methods use only the 20% of the data set, these results are impressive and substantiate the already showed characteristics of this kind of approach in improving accuracy. As regard the comparison between *BagCGPC* and *BoostCGPC*, it is worth to note that *BoostCGPC* obtains always mean error and tree size lower than *BagCGPC*, except for mean error of Census data set. Execution times of *BoostCGPC*, however, are worse than those of *BagCGPC*, though we can observe that while for the smallest data set (Segment, 2310 tuples) the execution time of *BoostCGPC* is 2.5 times more than that of *BagCGPC*, for the biggest data set (CovType, 581012 tuples) the execution time is 1.5 times more than that of *BagCGPC*. Furthermore, in the latter case, *BagCGPC* gives an error of 38.03, while *BoostCGPC* error is 33.65. The running time increase is mainly due to the weight computations and to the necessity of exchanging the hypotheses found after each round.

We wanted also to investigate the influence of the sample sizes on the accuracy of the method. To this end we used the *CovType* data set and run *CGPC* with the overall data set, while *BoostCGPC* was executed with 5%, 10%, and 20% of the tuples for 5 rounds, each round using an increasing number of classifiers. Figure 3 shows the effect of these different sample sizes on accuracy as the number of classifiers generated increases at each round. Parameters are the same of the previous experiments. *CGPC* used a population size equal to 100 $\times$ *number of classifiers* of the boosting algorithm. From the figure we can note that when *BoostCGPC* is trained on a sample of size 5% the overall data set, it is not able to outperform *CGPC* working on the entire data set. But, as the size increases, *BoostCGPC* is able to obtain an error lower than *CGPC*. An ensemble of two classifiers, 5 round, for a total of 10 classifiers, using the 10% of the data set obtains higher accuracy. Augmenting the sample size and the number of classifiers a further increase can be obtained. Using from 5 to 10 classifier at each round seems to be a good compromise between results obtained

Table 3. Comparing error, execution time, and size of the best tree for *CGPC* (using the complete data set), *BagCGPC* (20% tuples, 25 classifiers), and *BoostCGPC* (20% tuples, 5 rounds x 5 classifiers)

	Dataset				
CGPC	**Adult**	**Sat**	**Segment**	**Census**	**CovType**
Error	17.18	23.02	12.73	5.19	36.27
Time (sec)	4230	1212	1498	32820	66145
Size	233	122	205	389	112
BagCGPC					
Error	17.01	22.33	12.45	5.08	38.03
Time (sec)	528	83	46	3743	11277
Size	6055	3549	2806	14614	1636
BoostCGPC					
Error	16.53	21.22	10.90	8.33	33.65
Time (sec)	796	207	119	4448	16502
Size	4290	2930	755	8328	1467

and resources employed. In fact figure 3 shows a slow decrease of the mean error after these values.

5 Conclusions

An extension of Cellular Genetic Programming for data classification to induce an ensemble of predictors that uses a voting classification scheme based on boosting technique was presented, and a comparison with bagging like majority voting approach was performed. Experiments showed that the extension of *CGPC* with these voting algorithm enhances both accuracy and execution time. The approach is able to deal with large data sets that do not fit in main memory since each classifiers is trained on a subset of the overall training data. Experiments on a large real data set showed that, analogously to *BagCGPC*, higher accuracy can be obtained by using a sample of reasonable size at a much lower computational cost, and that sample size influences the achievable accuracy.

References

1. Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, (36):105–139, 1999.
2. Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
3. U.M. Fayyad, G. Piatesky-Shapiro, and P. Smith. From data mining to knowledge discovery: an overview. In U.M. Fayyad & al. (Eds), editor, *Advances in Knowledge Discovery and Data Mining*, pages 1–34. AAAI/MIT Press, 1996.
4. G. Folino, C. Pizzuti, and G. Spezzano. A cellular genetic programming approach to classification. In *Proc. Of the Genetic and Evolutionary Computation Conference GECCO99*, pages 1015–1020, Orlando, Florida, July 1999. Morgan Kaufmann.

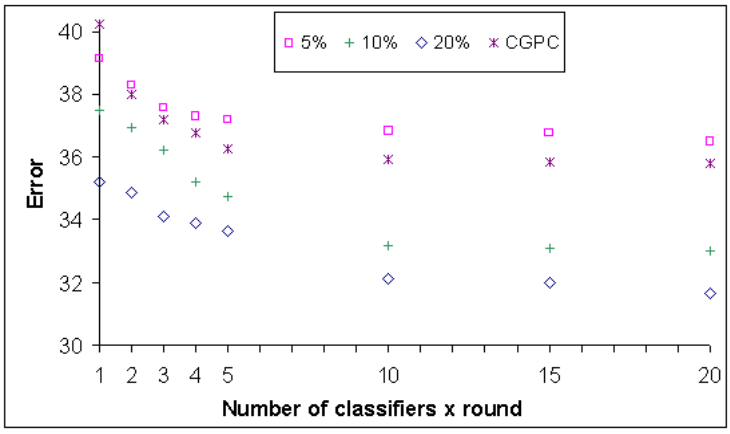

Fig. 3. Mean error for different sample sizes of the training set vs number of classifiers for round. (Covtype dataset)

5. G. Folino, C. Pizzuti, and G. Spezzano. Ensemble techniques for parallel genetic programming based classifiers. In E. Costa C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, editor, *Proceedings of the Sixth European Conference on Genetic Programming (EuroGP-2003)*, volume 2610 of *LNCS*, pages 59–69, Essex, UK, 2003. Springer Verlag.
6. G. Folino, C. Pizzuti, and G. Spezzano. A scalable cellular implementation of parallel genetic programming. *IEEE Transaction on Evolutionary Computation*, 7(1):37–53, February 2003.
7. A.A. Freitas. A genetic programming framework for two data mining tasks: Classification and generalised rule induction. In *Proceedings of the 2nd Int. Conference on Genetic Programming*, pages 96–101. Stanford University, CA, USA, 1997.
8. Y. Freund and R. Scapire. Experiments with a new boosting algorithm. In *Proceedings of the 13th Int. Conference on Machine Learning*, pages 148–156, 1996.
9. Hitoshi Iba. Bagging, boosting, and bloating in genetic programming. In *Proc. Of the Genetic and Evolutionary Computation Conference GECCO99*, pages 1053–1060, Orlando, Florida, July 1999. Morgan Kaufmann.
10. J. R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Selection*. MIT Press, Cambridge, MA, 1992.
11. R.E. Marmelstein and G.B. Lamont. Pattern classification using a hybbrid genetic program - decision tree approach. In *Proceedings of the Third Annual Conference on Genetic Programming*, Morgan Kaufmann, 1998.
12. C.J. Merz and P.M. Murphy. In *UCI repository of Machine Learning*, http://www.ics.uci/mlearn/MLRepository.html, 1996.
13. N.I. Nikolaev and V. Slavov. Inductive genetic programming with decision trees. In *Proceedings of the 9th International Conference on Machine Learning*, Prague, Czech Republic, 1997.
14. J. Ross Quinlan. Bagging, boosting, and c4.5. In *Proceedings of the 13th National Conference on Artificial Intelligence AAAI96*, pages 725–730. Mit Press, 1996.

15. M.D. Ryan and V.J. Rayward-Smith. The evolution of decision trees. In *Proceedings of the Third Annual Conference on Genetic Programming*, Morgan Kaufmann, 1998.
16. R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.
17. M. Sreenivas, K. AlSabti, and S. Ranka. Parallel out-of-core decision tree classifiers. In H. Kargupta and P. Chan, editors, *in Advances in Distributed Data Mining*, Menlo Park, 1996. AAAI Press.
18. A. Srivastava, E. Han, V. Kumar, and V. Singh. Parallel formulations of decision tree classification algorithms. *Data Mining and Knowledge Discovery*, 3(2):237–261, 1999.
19. M. Tomassini. Parallel and distributed evolutionary algorithms: A review. In P. Neittaanmki K. Miettinen, M. Mkel and J. Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, J. Wiley and Sons, Chichester, 1999.
20. M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. Parallel algorithms for discovery of association rules. *Data Mining and Knowledge Discovery, special issue on Scalable High Performance Computing*, 1(4):343–373, 1997.