Improving Induction Decision Trees with Parallel Genetic Programming

Gianluigi Folino, Clara Pizzuti and Giandomenico Spezzano
ISI-CNR
Via Pietro Bucci 41C
87036 Rende (CS), Italy
{folino,pizzuti,spezzano}@si.deis.unical.it

Abstract

A parallel genetic programming approach to induce decision trees in large data sets is presented. A population of trees is evolved by employing the genetic operators and every individual is evaluated by using a fitness function based on the J-measure. The method is able to deal with large data sets since it uses a parallel implementation of genetic programming through the grid model. Experiments on data sets from the UCI machine learning repository show better results with respect to C5. Furthermore, performance results show a nearly linear speedup.

1. Introduction

Data classification is an important *data mining* task [1] that tries to identify common characteristics in a set of N objects (tuples or examples) contained in a database and to categorize them into different groups (*classes*). Each tuple is composed of the same set of attributes and an additional known *class attribute* which identifies the class that the tuple belongs to. The classification process builds a model for each class by using the features of a subset of the available data, called *training set*. The models of each class are then applied to determine the class of the remaining data (*test set*) in the database. The most famous technique for data classification is constituted by *decision trees* and C4.5 [14] is the most popular decision tree based classification method. In the last few years several methods to builds accurate decision trees have been proposed. Among them, *genetic programming* (GP) [7] has showed to be a particularly suitable technique to deal with the task of data classification by evolving decision trees [4, 8, 9, 16, 15, 2]. In genetic programming, however, two main drawbacks limit the applicability of the approach.

First, trees are generated in a random way and evolved by applying the genetic operators. The number of trees that can be built is very high, the genetic programming algorithm needs to be endowed with a criterion to prefer a particular tree over another. The only way to provide an inductive bias to the genetic programming classification algorithm is through the fitness function. The already experimented [7, 2] fitness function that count the number of training examples classified in the correct class does not take into account the information content of a tree with respect to another, differently from C4.5 that tries to build the tree with the higher information gain at each node.

Second, when the database contains a high number of examples with many features, big populations of large decision trees are requested to accurately classify them. In data mining applications, databases with several millions of examples are common. Processing large populations of trees containing many nodes considerably degrades the execution times and requires an enormous amount of memory.

In this paper we present a parallel genetic programming algorithm to induce decision trees that uses a fitness function based on the *J-measure*, introduced by Smyth and Goodman [6] as an information theoretic approach that quantifies the information content of a rule: the higher is the J-measure, the better is the degree of interestingness of the rule. The method has been realized by using a parallel genetic programming tool based on the grid model [3]. Experimental results show good performances with respect to C5 (the latest version of C4.5) and nearly linear speedup.

The paper is organized as follows. Section 2 shows how genetic programming can be used to inductively generate decision trees. In section 3 fitness evaluation is described. In section 4 the parallel implementation of the algorithm is described. In section 5, finally, experimental results show the validity of the approach.

2. Decision tree encoding in Genetic Programming

A decision tree is a tree where the leaf nodes are labeled with the classes $C_i, i = 1, \ldots, k$ while the non leaf nodes

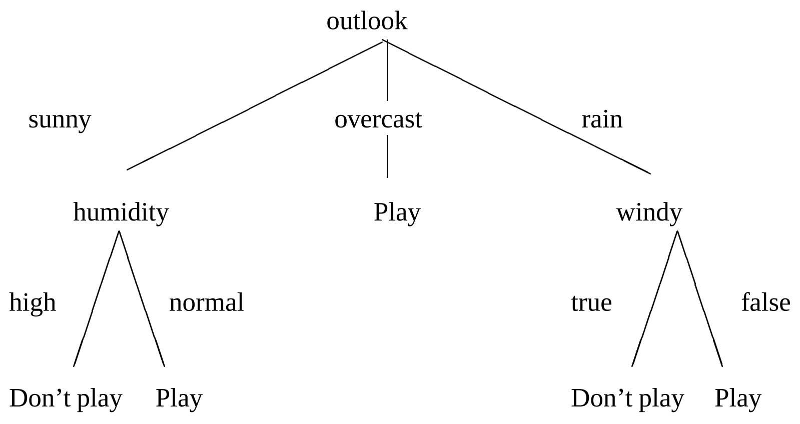

Figure 1. Decision tree for modeling play and don't play tennis.

(decision nodes) with the attributes $A_i, i = 1, \ldots, m$ of the training set. The branches leaving a node represent a test on the attribute values of the form $A_i = v_j$, where v_j is one of the possible values A_i can assume. The path from the root to a leaf represents a set of conditions attribute-value which describes the class C_i labeling that leaf. In figure 1 the well known decision tree built for the well known small training set given in [14] and consisting of four attributes (*outlook, temperature, humidity and wind*) and two classes (*play, don't play*) is shown. The tree gives the weather condition under which it is better to play tennis or not. For example, the path from the root to the leftmost node, expressed as *outlook = sunny* $\wedge$ *humidity = high than Don't play* means that if the weather outlook is sunny and the humidity is high then don't play.

Genetic programming is a variation of genetic algorithms in which the evolving individuals are themselves computer programs instead of fixed length strings from a limited alphabet of symbols [7]. Programs are represented as trees with ordered branches in which the internal nodes are *functions* and the leaves are so-called *terminals* of the problem. The GP approach evolves a population of trees by using the genetic operators of *reproduction, recombination* and *mutation*. The reproduction operator copies individual trees of the current population into the next generation with a probability proportionate to their fitness. The recombination operator generates two new individuals by crossing two trees at randomly chosen nodes and exchanging the subtrees. The two individuals participating in the crossover operation are again selected proportionate to fitness. The mutation operator replaces one of the nodes with a new randomly generated subtree. Each tree represents a candidate solution to a given problem and it is associated with a *fitness value* that reflects how good it is, with respect to the other solutions in the population. There is also the possibility for preventing trees to become too deep and for simplifying them by using the parsimony.

Genetic programming can be used to inductively generate decision trees for the task of data classification. Decision trees can be interpreted as composition of functions where the function set is the set of attribute tests and the terminal set are the classes. The function set can be obtained by converting each attribute into an attribute-test function. Thus there are as many functions as there are attributes. For each attribute A, if $A_1, \ldots A_n$ are the possible values A can assume, the corresponding attribute-test function f_A has arity n and if the value of A is A_i then $f_A(A_1, \ldots A_n) = A_i$. When a tuple has to be evaluated, the function at the root of the tree tests the corresponding attribute and then executes the argument outcoming from the test. If the argument is a terminal, then the class name for that tuple is returned, otherwise the new function is executed.

The function set is $f_{outlook}(sunny, overcast, rain)$, $f_{humidity}(high, normal)$, $f_{windy}(true, false)$ and the terminal set is $\{play,\ don't\ play\}$. The fitness of an individual measures the goodness of a tree in classifying the data set. This measure can be done in several ways. The most common approach is to define it as the number of training examples classified in the correct class. This definition comes in a natural way from the classification problem and in genetic programming it is called *raw fitness*. However, the raw fitness is a very simple measure that does not take into account the information content of a tree with respect to another. A key point during the construction of a decision tree is the choice of the attribute that best discriminates the training examples. C4.5 of Quinlan [14] uses a statistical property that measures how well an attribute separates the examples, known as *information gain*. The information gain of an attribute is the expected reduction in entropy caused by partitioning the examples according to this attribute [11]. The entropy in information theory is interpreted as the minimum number of bits of information needed to encode the classification of a tuple and it is considered as a measure of the impurity in a collection of training examples. The higher is the information gain, the better is the prediction of the class attribute. In genetic programming trees are generated in a random way and evolved by applying the genetic operators. There are no criteria to prefer an attribute with respect to another. Since the number of trees that can be built is very high, the genetic programming algorithm needs to be endowed with an inductive bias to prefer a particular tree over the offspring after that crossover took place. The only way to provide an inductive bias to the genetic programming classification algorithm is through the fitness function. The inductive bias of ID3 [13], the precursor of C4.5, approximately consists in preferring shorter trees over longer trees and trees that place higher information gain attributes close to the root, are preferred over those that do not [11]. In genetic programming the former request can be realized by using the concept of *parsimony*: individuals having a high number of nodes are penalized over those having a low number of nodes by worsening their fitness value of a given factor. The preference on attributes having high information gain can not be realized in GP because trees are not built step by step. However, the utilization of a fitness function based on the J-measure [6], that determines the information content of a tree, can give a preference criterion to find the decision tree that classifies a set of tuples in the best way. The J-measure has already successfully been used by Wang et al. [18] to merge intervals for maximizing the interestingness of a set of association rules and by de Araujo et al. [5] to discover rules in databases. In the next, the definition of J-measure is given.

3. Fitness evaluation

Let T be a decision tree that classifies a data set of N tuples in $C_1, \ldots, C_k$ classes. Let P_j^i be a path from the root to a leaf node labelled with the class C_i. P_j^i is a set of attribute-value conditions that classify a part of the training examples as belonging to that class. Each class C_i is thus described by the disjunction $Y_i = P_1^i \vee \ldots \vee P_h^i$ of all the paths having C_i as leaf node, where h is the number of leaf nodes labelled C_i. Let the rule

(1) **if** Y_i **then** C_i

denote the decision tree restricted to the class C_i and $p(Y_i)$ the fraction of tuples satisfying the condition Y_i, $p(C_i)$ the fraction of tuples belonging to the class C_i and $p(C_iY_i)$ the fraction of tuples both satisfying the condition Y_i and belonging to the class C_i. Furthermore, $p(C_i \mid Y_i) = p(C_iY_i)/p(Y_i)$. The *J-measure* is defined as

$$(2) \quad \sum_{i=1}^{k} p(Y_i)p(C_i \mid Y_i)log(\frac{p(C_i \mid Y_i)}{p(C_i)})$$

In this formula $p(Y_i)$ measures the generality of the rule (1), the following term measures the discriminatory power of Y_i on C_i, that is it measures the distance between our a priori belief $p(C_i)$ about the class C_i and our a posteriori belief $p(C_i \mid Y_i)$ about C_i. Each product in the summation can be considered as a measure of the generality and the goodness of fit of the rule (1) with regard to the class C_i. Thus the overall sum provides the goodness to fit of all the tree. The higher is the values of the sum the better is the predictive accuracy of the tree. It is worth to note that formula (2) is an extension of the original formula of Smyth and Goodman that takes into account that a decision tree classifies tuples in a generic number of classes.

For example, let us consider the decision tree in figure 1. The rules for $Play$ and $Don't\ Play$ are :

if $(outlook = sunny \wedge humidity = normal) \vee (outlook = overcast) \vee (outlook = rain \wedge windy = false)$ **then** $Play$

if $(outlook = sunny \wedge humidity = high) \vee (outlook = rain \wedge windy = True)$ **then** $Don't\ Play$

Thus, if Y_1 and Y_2 denote the bodies of rules $Play$ and $Don't\ Play$, respectively, then

$p(Play)$=9/14=0.642857,

$p(Don't\ Play)$ = 5/14=0.357143,

$p(Y_1)$ = 9/14 = 0.642857,

$p(Y_2)$= 5/14= 0.357143,

$p(Play \mid Y_1)$ = $p(Don't\ Play \mid Y_2)$ =1

and the J-measure is

0.642857 * 1 * log(1/0.642857) + 0.357143 * 1 * log(1/0.357143) = 0.123355 + 0.159699 = 0.283054.

Notice that this tree is a perfect tree and it is the one generated by C4.5. If we consider any other tree which does not perfectly classifies the tuples, the value of the J-measure is lower.

```
Let $pc, pm$ be the crossover and mutation probability
for each point i in grid do in parallel
generate a random individual $ti$
evaluate the fitness of $ti$
end parallel for
while not MaxNumberOfGeneration do
for each point $i$ in grid
do in parallel
    generate a random probability p
     if ( p $< pc$) then
       select the cell $j$, in the neighborhood of $i$,
       such that $tj$ has the best fitness
       produce the offspring by crossing $ti$ and $tj$
       replace $ti$ with the best of the two offspring
    else
      if (p $< pm + pc$) then
       mutate the individual
      else
       copy the current individual in the population
      end if
     end if
  end parallel for
end while
```

Figure 2. Pseudo-code of the parallel genetic algorithm.

4. Parallel implementation

The genetic programming classification algorithm has been implemented by using a parallel genetic programming tool [3] based on the $grid$ model [12], also known as $cellular$ model. In this model each individual has a spatial location on a low-dimensional grid and the individuals interact locally within a small neighborhood. This 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. Fitness evaluation is done simultaneously for all the individuals and selection, reproduction and mating take place locally within the neighborhood. At the beginning, for each cell, an individual is randomly generated and its fitness is evaluated. Then, at each generation, every tree undergoes one of the genetic operators (reproduction, crossover, mutation) depending on

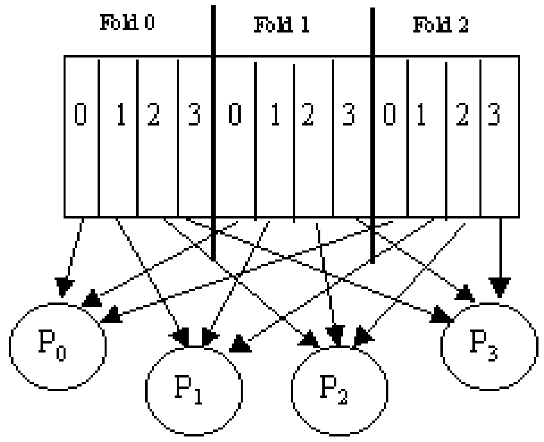

Figure 3. Load balancing strategy: each fold is divided in four strips.

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. The parallel algorithm on a 2-dimensional toroidal grid can be described by the pseudo-code shown in figure 2

The implementation has been realized on a general purpose distributed-memory parallel computer that uses a partitioning technique based upon domain decomposition in conjunction with the *Single-Program-Multiple-Data* ($SPMD$) programming model. According to this model, an application on N processing elements (PEs) is composed of N similar processes, each of which operates on a different set of data. For an effective implementation, data should be partitioned in such a way that communication takes place locally and the computation load be shared among the PEs in a balanced way. In this way, instead of assigning only one individual to a processor, the individuals are grouped by *slicing up* the grid and assigning a $slice$ of the population to a node. No coordinator process is necessary because the computational model is completely decentralized. Each slice process, with contains a strip of elements of the grid, runs on a single processing element of the parallel machine and executes the code on each subgrid point, thus updating all the individuals of the subpopulation. The size of the subpopulation of each slice process is calculated by dividing the population for the number of the processors of the parallel machine. Each slice process updates sequentially the individuals belonging to its subgrid.

We used a one-dimensional domain decomposition (in the x direction) of the grid that contains the population and

Table 1. Databases description

DATABASE	ATTR.	CLASSES	TUPLES
Australian	14	2	690
Breast-Cancer	9	2	286
German	20	2	1000
Hypo	29	5	3772
Monk1	6	2	124
Monk2	6	2	169
Monk3	6	2	122
Mushroom	22	2	8124
Segment	19	7	2310

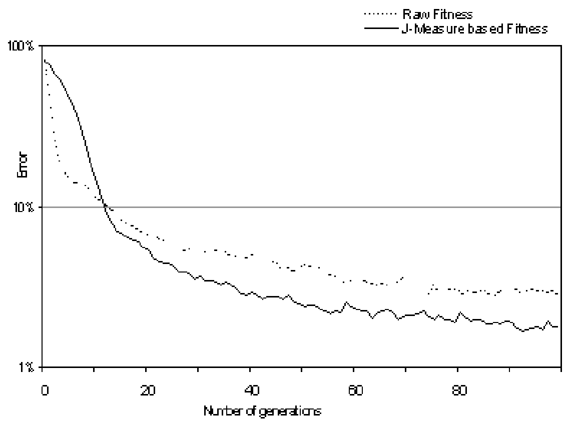

Figure 4. J-measure based fitness and raw fitness for the Hypo data set.

the explicit message passing (MPI library) to exchange information between these domains. In order to equally distribute the computational load among the processing nodes we introduce an intelligent partitioning of the grid. The partitioning strategy is a form of *block-cyclic* decomposition. The idea is to split the grid virtually in a number of folds and assign equal parts of each fold to each of the processes as shown in figure 3.

This can lead to load balancing provided that the resulting granules (further referred to as strip) are fine enough to assure that uneven load distribution across folds is statistically insignificant across processes. It should be noted, that the number of folds and processes should be chosen with caution, since the more strips are used, the bigger is the communication overhead among the processing elements. For a detailed description of the parallel implementation of genetic programming through the grid model and its comparison and performances with respect to the other parallel implementations of genetic programming see [3].

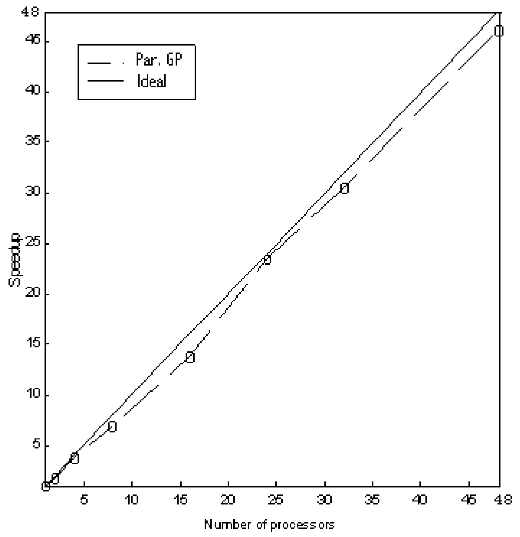

Figure 5. Speedup measure Segment data set.

5. Experimental Results

In this section we present the experiments and results obtained by implementing the method on a Linux Cluster having 4 nodes each with 2 800-Mhz Pentium III processors, 256 Mbytes of memory per processor, running under Redhat Linux 7.0. Experiments have been executed by running the method 20 times for 100 generations on real data sets contained in the UCI Machine Learning Repository [10]. Table 1 contains the description of these databases.

The maximum depth of the new generated subtrees is 4 for the step of population initialization, 6 for crossover and 2 for mutation. The population size has been set to 400 elements. In table 2 the results generated by C5 with pruning and the our method are presented. For the latter the best result over the 20 runs is reported together with the average error in the brackets. The size of the tree is relative to the tree having the lowest misclassification error on the test set, while in parenthesis the average tree size is reported.

The table shows that the trees generated by the genetic programming algorithm with respect to C5 have a misclassification error on both the training set and the test set lower than C5, except for Segment. In particular for the Monk1 and Mushroom problems our algorithm finds the exact tree for almost always the 20 runs, for Monk3 a null error on the test set. In figure 4 the values of the fitness computed with the J-measure and the raw fitness are compared for the Hypo problem and show the better convergence of the former. Finally in figure 5 the speedup results are showed for the Segment data set on a Meiko CS-2 parallel machine with

Table 2. Results generated by Parallel GP

	C5			Parallel GP		
DB	**Size**	**Train set**	**Test set**	**Size**	**Train set**	**Test set**
Australian	17	10.7	11.3	34 (24)	9.130 (12.231)	10.435 (11.961)
Breast-Cancer	21	19.9	25.3	29 (55)	12.042 (24.554)	20.000 (24.657)
German	65	15.3	26.0	51 (32)	21.021 (26.592)	25.444 (26.715)
Hypo	17	0.6	1.0	37 (25)	0.577 (1.092)	0.795 (1.393)
Monk1	17	12.1	25.7	37 (45)	0 (0.040)	0 (0.139)
Monk2	20	23.7	35.0	9 (32)	17.160 (22.267)	34.574 (36.091)
Monk3	9	6.6	2.8	22 (23)	4.098 (5.291)	0 (0.927)
Mushroom	23	0	0.0004	30 (38)	0 (0.246)	0 (0.441)
Segment	93	4.9	8.3	58 (68)	10.455 (12.522)	10.909 (12.345)

48 130-Mhz processors. The performance results that we have measured with the parallel implementation of the algorithm are very interesting since they give good scalability and show a nearly linear speedup.

6. Conclusion

A new approach to induce decision trees for the data mining task of classification has been presented. The method builds trees through genetic programming and evaluates their accuracy by using a fitness function based on the J-measure. The method is able to deal with large data sets since it uses a parallel implementation of genetic programming through the grid model. Experiments on data sets from the UCI machine learning repository show better results with respect to C5.

References

[1] U.M. Fayyad, G. Piatesky-Shapiro and P. Smith (1996). From Data Mining to Knowledge Discovery: an overview. In U.M. Fayyad & al. (Eds) *Advances in Knowledge Discovery and Data Mining*, pp.1-34, AAAI/MIT Press.

[2] G. Folino, C. Pizzuti and G. Spezzano (1999). A Cellular Genetic Programming Approach to Classification. *Proc. Of the Genetic and Evolutionary Computation Conference GECCO99*, Morgan Kaufmann, pp. 1015-1020, Orlando, Florida.

[3] G. Folino, C. Pizzuti and G. Spezzano (2000). CAGE : A Tool for Parallel Genetic Programming Applications. *Proc. Of the European Genetic Programming Conference EuroGP01*, Como, 15-16 April, to appear.

[4] A.A. Freitas (1997). A Genetic Programming Framework for two Data Mining Tasks: Classification and Generalised Rule Induction. *GP'97: Proc. 2nd Annual Conference*, pp.96-101, Stanford University, CA, USA.

[5] D.L.A. de Araujo, H.S. Lopes and A.A. Freitas (2000). A Parallel Genetic algorithm for Rule Discovery in Large Databases. *Data Mining with Evolutionary Algorithms*, Freitas, Hart, Krasnogor, Smith eds., GECCO-2000 workshop, Las Vegas, Nevada, USA pp.89-94.

[6] P. Smyth and R.M. Goodman (1992). An Information Theoretic approach to Rule Induction From Databases. *IEEE Transaction on Knowledge and Data Engineering*, vol. 4, N. 4, pp.301-316.

[7] J. R. Koza (1992). *Genetic Programming: On Programming Computers by Means of Natural Selection and Genetics*, MIT Press.

[8] N.I. Nikolaev and V. Slavov (1997). Inductive Genetic Programming with Decision Trees. *Proceedings of the 9th International Conference on Machine Learning*, Prague, Czech Republic, April 1997.

[9] R.E. Marmelstein and G.B. Lamont (1998). Pattern Classification using a Hybbrid Genetic Program - Decision Tree approach. *Proceedings of the Third Annual Conference on Genetic Programming*, Morgan Kaufmann.

[10] C.J. Merz and P.M. Murphy (1996). UCI repository of Machine Learning. *http://www.ics.uci/mlearn/MLRepository.html*.

[11] Tom M. Mitchell (1997). *Machine Learning.* McGraw-Hill Int. Editions.

[12] C. C. Pettey (1997), Diffusion (cellular) models, in T. Back, D.B. Fogel, Z. Michalewicz (eds.), *Handbook of evolutionary Computation*. IOP Publishing and Oxford University Press.

[13] J. Ross Quinlan (1986). Induction of decision trees. *Machine Learning*, 1(1),81-106.

[14] J. Ross Quinlan (1993). *C4.5 Programs for Machine Learning.* San Mateo, Calif., Morgan Kaufmann.

[15] M.D. Ryan and V.J. Rayward-Smith (1998). The Evolution of Decision Trees. *Proceedings of the Third Annual Conference on Genetic Programming*, Morgan Kaufmann.

[16] W.A. Tackett (1993). Genetic Programming for feature discovery and image discrimination. *Proceedings of the Fifth International Conference on Genetic Algorithms*.

[17] T. Toffoli and N. Margolus (1986). *Cellular Automata Machines A New Environment for Modeling*. The MIT Press, Cambridge, Massachusetts.

[18] K. Wang, S. Tay and B. Liu (1998). Interestingness-Based Interval Merger for Numeric Association *Proc. of 4th Int.Conf. on Knowledge Discovery and Data Mining*, pp.121-127, AAAI/MIT Press.