Searching the Forest: Using Decision Trees as Building Blocks for Evolutionary Search in Classification Databases

SE Rouwhorst
Department of Informatics and Mathematics
Vrije Universiteit of Amsterdam
The Netherlands
serouwho@cs.vu.nl

AP Engelbrecht
Department of Computer Science
University of Pretoria
South Africa
engel@driesie.cs.up.ac.za

Abstract- A new evolutionary search algorithm, called BGP, to be used for classification tasks in data mining, is introduced. It is different from existing evolutionary techniques in that it does not use indirect representations of a solution, such as bit strings or grammars. The algorithm uses decision trees of various sizes as individuals in the populations and operators, e.g. crossover, are performed directly on the trees. When compared to C4.5 and CN2 on a benchmark of problems, BGP shows very good results.

1 Introduction

The extraction of useful information from databases - commonly known as knowledge discovery in databases (KDD) - has become a field of study widely investigated. Large amounts of data can be analyzed in order to find patterns, which can help to predict future values, or classify future instances, among other tasks. Traditional methods have shown to be insufficient for many real-world problems. There is a demand for intelligent methods, which can deal with the enormous amount of available data. It is therefore not surprising that many of the recently developed methods, have their origins in artificial intelligence and machine learning [6]. These new methods combine and refine approaches, such as logic, statistics, artificial neural networks and evolutionary based algorithms to perform the task of KDD.

The research in this article involves classification tasks only. Several techniques exist, e.g. C4.5 [7], AQ15 [5], which take a dataset containing classification data as input and return a (nearly) optimal decision tree or a set of rules for this dataset. The goal of this research was to develop a new technique and to perform a preliminary comparison with C4.5 and CN2 [1] on a benchmark of classification databases. The technique is based on genetic programming where decision trees are used as representation scheme of the individuals in the evolving population. The idea of using genetic algorithms (GA's) or genetic programming for classification tasks in KDD is not new. In GABIL [2] a Genetic Algorithm is used so that the bit string represents a set of rules. Other approaches to data mining using evolutionary computing include REGAL [4] and GA-MINER [3]. The main difference between existing evolutionary approaches to data mining and the new algorithm presented in this paper is that we use a direct representation of solutions compared to an indirect representation. The decision trees are considered the 'building blocks' of the final outcome of the algorithm. The name for the algorithm therefore was chosen to be 'Building block approach to Genetic Programming' (BGP). BGP starts with small decision trees and performs operators, such as crossover and mutation directly on the trees. Over generations of evolving decision trees, new conditions are gradually added to the trees. This process is repeated until a tree is found to be a 'good' solution to the given problem according to some criterion. In this way the resulting trees are kept as small as possible.

The BGP algorithm is described in section 2. In section 3 the experiments that were done are described. Section 4 contains a discussion of the results of these experiments. Finally, conclusions and some ideas for future work are presented in section 5.

2 Methods

2.1 Assumptions

In order to avoid confusion about terminology, some concepts are explained first. For this research, a database is seen as a matrix, where each row represents an instance and each column represents an attribute. For example, we could have a medical database with 4 attributes: 'Bloodsugar level', 'Age', 'Diet', 'Pregnant'. Databases can hold almost any kind of information, but for this algorithm it is assumed that the data can be either *continuous*, *discrete*, *nominal* or *Boolean*. 'Bloodsugar level' can then be seen as a continuous attribute, 'Age' as a discrete attribute and 'Pregnant' as a Boolean attribute. Suppose the attribute 'Diet' can have the following values: 'sugar-free diet', 'salt-free diet' and 'no diet', then this attribute is of type *nominal*. This information about the attributes is assumed to be known prior to execution of the algorithm. There are two more assumptions about data: First, there are no unknown or missing values and second, the classification of all instances is known. For example, in the above medical database the instances are patients and for all patients it is known whether they are 'Healthy' or 'Not Healthy'.

0-7803-6375-2/00/$10.00 ©2000 IEEE.

2.2 Decision trees

As said before, the building blocks for the search algorithm are (parts of) decision trees, so it is wise to describe what kind of decision tree was used in this research. Decision trees classify instances by sorting them down the tree from the root to a leaf node, which provides the classification of the instance. Each node in the tree specifies a condition of some attribute of the instance. When an instance is parsed through the tree, the left child of a node is chosen if the condition in the node is true for that instance, else the right child is chosen.

The conditions that were used in the decision trees are:

- *For all attributes*: == (equal to) and ! = (not equal to)
- *For continuous and discrete attributes also*: < (less than), <= (less than or equal to), > (greater than), >= (greater than or equal to)

BGP counts each of the relational operators as one condition. This means that, for example, >= is treated as one condition, contrary to C4.5 and CN2 where two conditions are used to represent this relation.

The left hand side of a condition is an attribute from the database. The right hand side can be an attribute or a threshold value. By allowing the right hand side of a condition to be an input attribute, the BGP gains more representation power than what is offered by C4.5 and CN2 (and most other data mining tools for that matter). Three example conditions are:
IF 'Age' >= 'Bloodsugar level'
IF 'Pregnant' == 'True'
IF 'Attribute1' < 'Attribute2'

Four different operators are performed directly on the trees or on their nodes:

Crossover: Given two 'parent' trees, randomly select two of their non-leaf nodes and swap the subtrees starting at these nodes.

Mutation on relational operator: Given a condition, change the relational operator (==, ! =, <, <=, > or >=) into a randomly selected one such that the semantics are not violated.

Mutation on right hand side: Given a condition, change the right hand side of the condition into a randomly chosen (allowed) new one. This can be either a value of the attribute or another attribute.

Prune: Given a tree, randomly select one of its non-leaf nodes and delete the subtree starting at this node. The selected node will become a leaf node and will contain a classification.

A probability is defined for each operator. Suppose the probability for crossover is 0.6, then approximately 60% of the trees in the population will undergo crossover. The probabilities for the two mutation operators work on the nodes instead of the trees. This means that if the probability for 'mutation on the relational operator' is 0.3, then approximately 30% of the nodes in all trees will undergo this kind of mutation.

The decision tree described so far can easily be transformed into a set of rules. The number of rules equals the number of leaf nodes in the tree. A rule consists of all conditions in the path from the root to a leaf node concatenated by 'AND' and followed by the classification taken from the leaf node.

2.3 The Search Algorithm - BGP

In order to perform the search, the available data is first divided into two sets of instances: training instances and test instances. The BGP algorithm is then applied to the training instances. The pseudo-code of the algorithm is:

```
Algorithm BGP
    read information from database
    randomly initialize population
    while not 'best_tree_is_good_enough' do
        if 'add_nodes_condition' then
          add nodes to trees in population
        end if
        for i=1 to 'population size'
            tournament select individual(s)
            perform operators
        end for
        if 'found_new_best_tree' then
          store copy of new best tree
        end if
    end while
    extract rules from 'best_tree'
end Algorithm
```

The algorithm starts by reading information about the database, such as the type of attributes, the values of attributes and classification of the training instances. The initial population will then be randomly generated. All trees in the population have just one condition at this moment. The population size was usually a hundred individuals. During the entire search a copy of the best decision tree found so far is stored separately. Every cycle starts with checking whether this tree is the 'best' within the population according to the following test:

$$\text{IF } \left(MinRuleAcc > e^{\left(c\left(\frac{trainsize}{T(0)}\right) - c\left(\frac{trainsize}{T(t)}\right)\right)} \right)$$

THEN 'best tree'

where $MinRuleAcc$ is the lowest accuracy of all the rules in the tree, c is a parameter for the algorithm, $trainsize$ is the number of training instances, T(t) is a temperature function that starts by returning the initial temperature $T(0)$ and decreases its output with time until it reaches 0. The temperature function used in these experiments was very simple: $T(t) = T(0) - t$, where t is the generation number. See figure 1 for an example of this function. If discovered that the

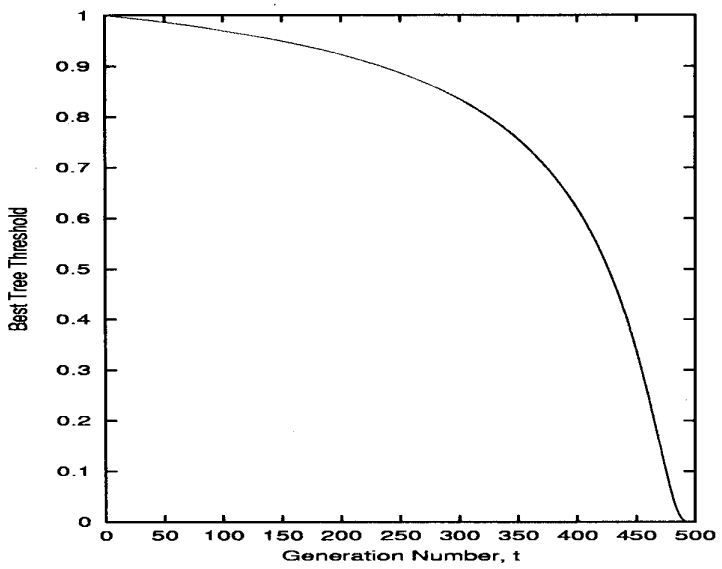

Figure 1: A plot of the function that is used to decide whether a decision tree is the best: $e^{\left(c(\frac{trainsize}{T(0)})-c(\frac{trainsize}{T(t)})\right)}$ for $c = 0.1$, $T(0) = 500$ and $trainsize = 600$.

lowest rule accuracy of the tree exceeds the calculated value, then the rules are extracted from this tree and the algorithm stops running.

The next three steps will be repeated if a satisfactory tree has *not* been found yet. First a check is performed to see if conditions should be added to the trees in the population. The new conditions to be added to the decision trees are generated randomly. For this check, the average width and average depth of the trees in the population are determined. These averages are compared to the averages of the previous generation. If the difference is below a user-specified value L, then conditions are added to the trees:

$$\text{IF } \Big((av\_depth + av\_width)_{(t)} - (av\_depth + av\_width)_{(t-1)} < L \Big) \text{ THEN 'add\_conditions'}$$

The second step is to perform the four operators, i.e. crossover and mutation. The individuals are chosen via tournament-selection. The number of elements in the tournament is also a parameter for the algorithm. The last step is to check whether the new population contains a tree with a higher accuracy on the training instances. If so, a copy of this tree will be saved as the 'best_tree' found so far.

3 Experiments

The main goal of this paper is to present BGP, and to perform a preliminary comparison with established data mining tools. For this purpose four databases were taken from the UCI Machine Learning archive[1] available to anyone on the WWW:

Ionosphere which has 351 instances, each using 34 continuous attributes. There are two classes, which are unevenly distributed among the instances.

Iris which uses four attributes of which two are low and two are highly correlated to the classification. All attributes are continuous. The number of instances is 150 and there are three evenly distributed classes.

Monks which is an artificial dataset. There are six nominal attributes: two with two discreet values, three with three discreet values and one with four values. The data set consists of all 432 possible instances given these attributes. Monks-1 was generated using the rule:
if ((attr1 == attr2) or (attr5==1)) then Monk

Monks-2:
if exactly two of ((attr1==1), (attr2==1), (attr3==1), (attr4==1), (attr5==1), (attr6==1)) then Monk
Monks-3:
if ((attr5==3 and attr4==1) or (attr5!=4 and attr2!=3)) then Monk

The Monks-3 problem is provided with a training set that has 5% class noise and a test set without noise. To use this training set to generate thirty new representative training sets, was not possible. Only the provided training and test set were used on the three algorithms. Therefore it was not possible to determine confidence intervals for the results of this problem.

[1] http://www.ics.uci.edu/~mlearn

Pima-diabetes This is a fairly large dataset compared to the others. It consists of 768 instances using eight attributes (six discrete and two continuous). There are two unevenly distributed classes.

For each problem a training set of instances was drawn randomly from the available instances. The remaining instances were used as the test set. This process was repeated until thirty training and test sets were generated, all to be used by CN2, C4.5 and BGP. For CN2 and C4.5 the default parameters were used. The output of C4.5 is a decision tree, but the algorithm is accompanied by a program to convert the decision tree into an optimal set of rules. This program first generates a rule for each leaf node, using all conditions in the path from the root to that leaf node. This is not different from the way we convert the decision tree generated by BGP to a rule set. The C4.5 rule program then applies pruning methods to get the smallest possible set of rules with the same or higher accuracy as the decision tree. For each algorithm and each of the thirty couples of training and test sets, the accuracy on this test set was determined by dividing the number of correctly classified instances by the total number of instances. The number of rules and the average number of conditions in the rules were also used as performance criteria. We assumed that the obtained accuracies were distributed normally around the 'real' accuracy. With 95% probability the confidence intervals for the accuracies could then be calculated using the following formula:

$$\bar{a} \pm 1.96\sqrt{\frac{\bar{a}(1-\bar{a})}{n}}$$

where $\bar{a}$ is the average of the thirty accuracies and $n = 30$ is the number of runs on different couples of training and tests set. Also an estimate of the difference in accuracies between two algorithms was calculated, namely between BGP and CN2 and between BGP and C4.5. The confidence intervals for the difference at 95% probability was calculated using the following formula:

$$\bar{d} \pm \left(t_{(95\%,n-1)} \sqrt{\frac{1}{n(n-1)} \sum_{i=1}^{n} (d_i - \bar{d})^2} \right)$$

$$\text{where} \qquad \bar{d} = \frac{1}{n} \sum_{i=1}^{n} d_i$$

d_i is the accuracy of BGP on the i^{th} couple of training and test sets, minus the accuracy of the other algorithm on the same training and test set, $n = 30$ is the number of runs on different couples of training and test sets and $t_{95\%,n-1} = 2.045$.

4 Results

Table 1 shows the average accuracies over thirty different test sets for the three algorithms CN2, C4.5 and BGP. The table also shows the average number of rules and the average number of conditions per rule for each algorithm. For the Monks-3 problem no confidence intervals were calculated, as each algorithm was only performed on one training and test set. Table 3 summarizes the optimal BGP parameters used for experiment.

On the Iris, Monks-1 and Pima databases BGP has slightly lower accuracies than CN2 and C4.5. When looking at the confidence intervals in table 2 the BGP algorithm is not *significantly* worse. Table 2 shows the average difference between two algorithms and the confidence intervals at 95% probability. The only database on which BGP has a significantly worse accuracy is on the Ionosphere database. On the other databases BGP has a similar or significantly higher accuracy. The biggest difference however is not in the accuracies but in the average number of rules. BGP produced almost consistently less rules than the other algorithms. Especially in the Monks-2 problem the difference between CN2 and BGP is striking. The difference can be explained partially by the fact that BGP uses a wider variety of relational operators in the conditions than the other two algorithms ($\{==, !=, <, <=, >, >=\}$ for BGP, $\{==, <, >\}$ for CN2 and $\{==, <=, >\}$ for C4.5). Another reason for the difference is that in BGP the right hand side of a condition is allowed to be an attribute or a threshold value, whereas in CN2 and C4.5 the right hand side of the condition can only be a threshold value. It is clear that the results of C4.5 on the number of rules is better than CN2, because it prunes the rules when converting a decision tree to a set of rules. BGP still produced significantly less rules than C4.5, though, with similar accuracies.

While the results above represent a preliminary comparison of BGP with C4.5 and CN2, using default values as provided by C4.5 and CN2, further investigations will perform a comparison with C4.5 and CN2 using optimal parameter values.

5 Conclusions

This paper discusses a new approach to classification tasks in data mining, called 'BGP', short for 'Building block approach to Genetic Programming'. It is an evolutionary search method that takes decision trees of various sizes as individuals. Four different operators, namely cross-over, pruning and mutation on the threshold and relational operators, were performed directly on the decision trees. This approach is different from other evolutionary based techniques in that it does not use indirect representations of a solution, e.g. bitstrings or grammars. BGP was compared to two standard machine learning algorithms, CN2 and C4.5, on four benchmark problems (Iris, Ionosphere, Monks and Pima-diabetes). The accuracies of BGP were similar or better to the accuracies of CN2 and C4.5, except on the Ionosphere database. The main difference with the C4.5 and especially CN2 is that BGP produced these accuracies using almost consistently less rules.

It should be emphasized that this work was just a prelimi-

Task		BGP	CN2	C4.5
Iono	Test Accuracy	0.892	0.921	0.931
	Average # rules	4.70	17.07	8.57
	Avg. # cond. in rules	2.39	2.35	2.15
Iris	Test Accuracy	0.941 ± 0.085	0.943 ± 0.083	0.944 ± 0.082
	Average # rules	3.73	5.33	4.10
	Avg. # cond. in rules	2.02	1.64	1.60
Monks-1	Test Accuracy	0.993 ± 0.029	1.00 ± 0.000	1.00 ± 0.000
	Average # rules	4.37	18.0	21.5
	Avg. # cond. in rules	2.22	2.37	2.73
Monks-2	Test Accuracy	0.684 ± 0.167	0.626 ± 0.173	0.635 ± 0.172
	Average # rules	6.00	122.8	13.9
	Avg. # cond. in rules	2.96	4.53	3.01
Monks-3	Test Accuracy	0.972	0.907	0.963
	# rules	3	22	12
	Avg. # cond. in rules	1.67	2.17	2.77
Pima	Test Accuracy	0.725 ± 0.160	0.739 ± 0.157	0.734 ± 0.158
	Average # rules	3.70	35.8	12.73
	Avg. # cond. in rules	1.97	2.92	3.90

Table 1: For each problem the first entry states average accuracies on test set over 30 runs and confidence intervals at 95% probability. The second entry states the average number of rules over the 30 runs. The third entry states the average number of conditions in the rules.

Task	BGP & CN2	BGP & C4.5
Iono	**-0.0286 ± 0.0263**	**-0.0385 ± 0.0142**
Iris	-0.00267 ± 0.0237	-0.00400 ± 0.0115
Monks-1	-0.00657 ± 0.00933	-0.00657 ± 0.00933
Monks-2	**+0.0576 ± 0.0165**	**+0.0485 ± 0.0154**
Pima	-0.0132 ± 0.0160	-0.00844 ± 0.0190

Table 2: Comparison between BGP and CN2, and BGP and C4.5 using t-tests over 30 training and test sets to determine confidence intervals at 95%. A '+'means that BGP showed better results than the algorithm it is compared to and '−' means BGPs results wore worse. The bold font indicates that one method is significantly better than the other.

nary investigation into the possibilities of our approach. What will follow is a study of optimal operator probabilities, different selection methods for individuals, and a comparison with optimal C4.5, CN2 and other evolutionary approaches to data mining. Given the difficulties BGP had with the Ionosphere problem, it may be fruitful to use a different method of generating new conditions. In this research, whenever a new (part of a) condition had to be generated, is was done in a random way. It is worth investigating, whether local search could be used to generate new (parts of) conditions.

A comparison of the distribution of relational operators used by each approach will be conducted to investigate the importance of the set of operators used by each algorithm.

6 Acknowledgements

Thanks to Frans van den Bergh for suggestions and feedback.

Bibliography

[1] Clark, P., Niblett, R. The CN2 induction algorithm. From *Machine Learning*, 3, pp. 261-284. 1989.

[2] De Jong, K.A., Spears, W.M., Gordon, D.F. Using Genetic Algorithms for Concept Learning. From *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI-1991)*, pp. 651-656. 1991

[3] Flockhart, I.W., Radcliffe, N.J. GA-MINER: parallel data mining with hierarchical genetic algorithms - final report. From *EPCC-AIKMS-GA-MINER-Report 1.0*. University of Edinburgh. 1995.

[4] Giordana, A., Saitta, L., Zini, F. Learning Disjunctive Concepts by means of Genetic Algorithms. From *Proceedings on the 11th International Conference on Machine Learning (ICML-1994)*, pp. 96-104. Morgan Kaufmann, 1994.

[5] Michalski, R.S., Mozetic, I., Hong, J. Lavrac, H. The multi-purpose incremental learning system AQ15 and

Parameter	Data Set					
	Iniosphere	Iris	Monks1	Monks2	Monks3	Pima-diabetes
Training set	300	120	420	300	300	500
Test set	51	30	30	132	132	268
c	0.1	0.1	0.1	0.1	0.1	0.1
L	0.0	0.0	0.0	0.0	0.0	0.0
Tournament Size	20	10	10	10	10	20
$T(0)$	2000	300	200	1500	500	2000
Probability that threshold is attribute	0.1	0.2	0.7	0.2	0.1	0.1
Threshold Mutation Probability	0.2	0.7	0.7	0.4	0.3	0.2
Relational Operator Mutation Probability	0.4	0.2	0.2	0.2	0.3	0.4
Pruning Probability	0.5	0.2	0.2	0.5	0.5	0.5
Cross-over Probability	0.5	0.5	0.8	0.5	0.5	0.5
Population Size	100	100	100	100	100	100

Table 3: BGP system parameters

its testing application to three medical domains. From *Proceedings of the Fifth National Conference on AI*, pp. 1041-1045. Philadelphia. Morgan Kaufmann, 1986

[6] Mitchell, T.M. *Machine Learning*. McGraw-Hill, USA. 1997.

[7] Quinlan, J.R. From *C4.5: Programs for Machine Learning*. San Mateo, CA. Morgan Kaufmann, 1993.