A hybrid decision tree/genetic algorithm for coping with the problem of small disjuncts in data mining

Deborah R. Carvalho

Pontificia Universidade Catolica do Parana (PUCPR)
Postgraduate program in applied computer science
R. Imaculada Conceicao, 1155. Curitiba – PR
80215-901. Brazil

Universidade Tuiuti do Parana (UTP)
Computer Science Dept.
Av. Comendador Franco, 186. Curitiba-PR
80215-090 Brazil
deborah@utp.br

Alex A. Freitas

Pontificia Universidade Catolica do Parana (PUCPR)
Postgraduate program in applied computer science
R. Imaculada Conceicao, 1155. Curitiba – PR
80215-901. Brazil
alex@ppgia.pucpr.br
http://www.ppgia.pucpr.br/~alex
Tel./Fax: (55) (41) 330-1669

Abstract

The problem of small disjuncts is a serious challenge for data mining algorithms. In essence, small disjuncts are rules covering a small number of examples. Due to their nature, small disjuncts tend to be error prone and contribute to a decrease in predictive accuracy. This paper proposes a hybrid decision tree/genetic algorithm method to cope with the problem of small disjuncts. The basic idea is that examples belonging to large disjuncts are classified by rules produced by a decision-tree algorithm, while examples belonging to small disjuncts (whose classification is considerably more difficult) are classified by rules produced by a genetic algorithm specifically designed for this task.

1 INTRODUCTION

In the context of the well-known classification task of data mining, the discovered knowledge is often expressed as a set of IF-THEN rules, since this kind of knowledge representation is intuitive for the user. From a logical viewpoint, typically the discovered rules are in disjunctive normal form, where each rule represents a disjunct and each rule condition represents a conjunct. A small disjunct can be defined as a rule which covers a small number of training examples (Holte et al. 1989).

In general rule induction algorithms have a bias that favors the discovery of large disjuncts, rather than small disjuncts. This preference is due to the belief that it is better to capture generalizations rather than specializations in the training set, since the latter are unlikely to be valid in the test set (Danyluk & Provost 1993).

Hence, at first glance, small disjuncts are not important, since they tend to be error prone. However, a deeper study of the issue of small disjuncts reveals that in fact they are quite interesting in the context of data mining, for the following reasons:

(a) Although each disjunct covers a small number of examples, the set of all small disjuncts can cover a large number of examples. For instance (Danyluk & Provost 1993) report a real-world application where small disjuncts cover roughly 50% of the training examples. Therefore, if the rule induction algorithm ignores small disjuncts and discovers only large disjuncts, classification accuracy will be significantly degraded.

(b) Some small disjuncts cover examples that represent rare cases in the application domain, which constitutes an interesting concept to be discovered. Actually, bearing in mind that one of the goals of data mining is to discover previously-*unknown* rules, small-disjunct rules tend to be more interesting than large-disjunct rules, since the latter are more likely to be previously-known by the user (Provost & Aronis 1996).

In this paper we propose a hybrid decision tree/genetic algorithm method for rule discovery that copes with the problem of small disjuncts. The basic idea is that examples belonging to large disjuncts are classified by rules produced by a decision-tree algorithm, while examples belonging to small disjuncts (whose classification is considerably more difficult) are classified by rules produced by a new genetic algorithm, specifically designed for discovering small-disjunct rules.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 describes our hybrid decision tree/genetic algorithm method for rule discovery.

This section assumes that the reader is familiar with decision-trees, which is a very well-known knowledge discovery paradigm in data mining, and focus on a detailed description of a new genetic algorithm proposed in this paper. Section 4 reports the results of experiments evaluating the performance of the proposed method on a case study dataset. Finally, section 5 presents the conclusions and some future research directions.

2 RELATED WORK

(Holte et al. 1989) investigated three possible solutions for coping with small disjuncts, namely: (a) Eliminating all rules whose number of covered training examples is below a predefined threshold. In effect, this corresponds to eliminating all small disjuncts, regardless of their estimated performance. (b) Eliminating only the small disjuncts whose estimated performance is poor. (c) Using a specificity bias for small disjuncts (while using a generality bias for large disjuncts). The third approach turned out to be partly (but not entirely) successful.

(Ting 1994) proposed the use of a hybrid data mining method to cope with small disjuncts. His method consists of using a decision-tree algorithm to cope with large disjuncts and an instance-based learning (IBL) algorithm to cope with small disjuncts. The basic idea of this hybrid method is that IBL algorithms have a specificity bias, which should be more suitable for coping with small disjuncts. In a high level of abstraction, the basic idea of this method is similar to our hybrid decision-tree/genetic algorithm method. However, Ting's method has the disadvantage that the IBL algorithm does not discover any high-level, comprehensible rules. By contrast, we use a genetic algorithm that does discover high-level, comprehensible small-disjunct rules, which is important in the context of data mining.

(Weiss 1995) investigated the interaction of noise with rare cases (true exceptions) and showed that this interaction led to degradation in classification accuracy when small-disjunct rules are eliminated. However, these results have a limited utility in practice, since the analysis of this interaction was made possible by using artificially generated data set. In real-world data sets the correct concept to be discovered is not known a priori, so that it is not possible to make a clear distinction between noise and true rare cases. (Weiss 1998) did experiments showing that, when noise is added to real-world data sets, small disjuncts contribute disproportionaly and significantly for the total number of classification errors made by the discovered rules.

3 A HYBRID DECISION-TREE / GENETIC-ALGORITHM METHOD FOR RULE DISCOVERY

As mentioned in the introduction, we propose a hybrid method for rule discovery that combines decision trees and genetic algorithms. The basic idea is to use a well-known decision-tree algorithm to classify examples belonging to large disjuncts and use a new genetic algorithm to discover rules classifying examples belonging to small disjuncts. This approach tries to combine the best of both worlds. Decision-tree algorithms have a bias towards generality that is well suited for large disjuncts, but not for small disjuncts. On the other hand, genetic algorithms are robust, flexible algorithms which tend to cope well with attribute interactions (Freitas 2000), (Noda et al. 1999), and can be more easily tailored for coping with small disjuncts.

The proposed method discovers rules in two training phases. In the first phase we run C4.5, a well-known decision tree induction algorithm (Quinlan 1993). The induced, pruned tree is transformed into a set of rules in the usual way - that is, each path from the root to a leaf node corresponds to a rule predicting the class specified in the corresponding leaf node. Hence, a decision tree with *d* leaves is transformed into a rule set with *d* rules (or disjuncts). Each of these rules is considered either as a small disjunct or as a "large" (non-small) disjunct, depending on whether or not its coverage (the number of examples covered by the rule) is smaller than or equal to a given threshold.

The second phase consists of using a genetic algorithm to discover rules covering the examples belonging to small disjuncts. We have developed a new genetic algorithm for this phase, which will be described in detail below.

Once the second training phase is over, examples in the test set are classified as follows. For each test example, we first check whether the example is covered by some large-disjunct rule. If so, the example is classified by the corresponding rule, which is one of the rules induced by the decision tree algorithm. Otherwise the example is classified by some small-disjunct rule, which is one of the rules discovered by our genetic algorithm.

It should be noted that the small-disjunct rules discovered by our genetic algorithm can overlap each other. Therefore, if a test example is to be classified by some small disjunct rule, there might be one of the following two kinds of rule conflict.

First, there might be more than one small-disjunct rule covering the test example. If this is the case, the example is classified by the highest-quality rule among all small-disjunct rules covering the examples. The quality of a rule is measured by the value of the fitness function computed by the genetic algorithm - described in section 3.3. Second, there might be no small-disjunct rule covering the test example. If this is the case, the example is classified by a default rule. This is a rule which simply predicts the majority class of the target dataset. A similar procedure is also used in several rule induction algorithms.

Finally, note also that these kinds of rule conflict cannot occur if the test example is to be classified by a large-disjunct rule, since these rules have mutually exclusive and exhaustive coverage, due to that fact that they were directly generated from the induced decision tree.

3.1 OVERVIEW OF A GA FOR DISCOVERING SMALL-DISJUNCTS RULES

In this section we describe our genetic algorithm (GA) developed for discovering small-disjunct rules - i.e. rules covering the examples in leaf nodes of a decision tree considered to be a small disjunct, as explained above.

The first step in the design of a GA for rule discovery is to decide what an individual (candidate solution) represents. In our case, each individual represents a small-disjunct rule. The genome of an individual consists of the conditions in the antecedent (IF part) of the rule. The goal of the GA is to evolve rule conditions that maximize the predictive accuracy of the rule, as evaluated by a fitness measure - described below. The consequent (THEN part) of the rule, which specifies the predicted class, is not represented in the genome. Rather, it is fixed for a given GA run, so that all individuals have the same rule consequent during all that run.

Each run of our GA discovers a single rule (the best individual of the last generation) predicting a given class for examples belonging to a given small disjunct. Since we need to discover several rules to cover examples of several classes in several different small disjuncts, we run our GA several times for a given dataset. More precisely, we need to run our GA $d * c$ times, where d is the number of small disjuncts and c is the number of classes to be predicted. For a given small disjunct, the k-th run of the GA, $k = 1,...,c$, discovers a rule predicting the k-th class.

At first glance this is a computationally expensive approach for rule discovery. However, note that in our approach each GA run will use as its training set - for the purpose of fitness computation - only the few (about 10) examples belonging to a given small disjunct. Therefore, we avoid the well-known bottleneck of most GAs for rule discovery, which is the long time taken to evaluate an individual's fitness when mining large datasets. Indeed, the computational results reported in section 4 confirm that the total processing time associated with all the runs of our GA is relatively short.

The next subsections describe in detail the individual representation, the fitness function, and the genetic operators used in our GA.

3.2 INDIVIDUAL REPRESENTATION

In our GA each individual represents the antecedent (IF part) of a small-disjunct rule. More precisely, each individual represents a conjunction of conditions composing a given rule antecedent. Each condition is an attribute-value pair - see below.

The rule antecedent contains a variable number of rule conditions, since one does not know a priori how many conditions will be necessary to compose a good rule. In practice, for implementation purposes, one has to specify both a lower limit and an upper limit in the number of conditions of a rule antecedent. In our GA the minimum number of rule conditions is 2. Although this number could be set to 1, recall that our GA is searching for small-disjunct rules. It is very unlikely that a rule with a single condition can accurately predict the class of an example belonging to a small disjunct, so a lower limit of 2 seems to make sense.

The maximum number of rule conditions is more difficult to determine. In principle, the maximum number of rule conditions could be m, where m is the number of predictor attributes in the dataset. However, this would have two disadvantages. First, it could lead to the discovery of very long rules, which goes against the desire to discover comprehensible rules. Second, it would require a long genome to represent individuals, which tends to increase processing time. To avoid these problems, we use a heuristics to select the subset of attributes that is used to compose rule conditions.

Our heuristics is based on the fact that different small disjuncts identified by the decision-tree algorithm can have several rule conditions in common. For instance, suppose that two sibling leaf nodes of the decision tree were deemed small disjuncts and let k be the number of ancestor nodes of these two leaf nodes. Then the two corresponding rule antecedents have k - 1 conditions in common. Therefore, it does not make much sense to use these common conditions in the rules to be discovered by the GA. Rather, for each small disjunct, the genome of a GA individual contains only attributes that were *not* used to label any ancestor of the leaf node defining that small disjunct.

To represent a variable-length rule antecedent (phenotype) we use a fixed-length genome, for the sake of simplicity. Recall that each GA run discovers a rule associated with a given small disjunct and that each small disjunct is identified by a decision tree leaf node. For a given run of the GA, the genome of an individual consists of n genes, where $n = m - k$, where m is the total number of predictor attributes in the dataset and k is the number of ancestor nodes of the decision tree leaf node identifying the small disjunct in question.

Each gene represents a rule condition (phenotype) of the form A_i Op_i V_{ij}, where the subscript i identifies the rule condition, $i = 1,...,n$; A_i is the i-th attribute; V_{ij} is the j-th value of the domain of A_i; and Op is a logical/relational operator compatible with attribute A_i - see below. To encode this phenotype the internal representation of a gene consists of four elements, as follows:

(a) identification of a given predictor attribute, A_i, $i = 1,...,n$.

(b) identification of a logical/relational operator Op_i. Two cases are possible here. If attribute A_i is a categorical (nominal) attribute, the operator Op_i is "*in*", which will produce rule conditions such as "A_i *in* $\{V_{i1},...,V_{ik}\}$", where $\{V_{i1},...,V_{ik}\}$ is a subset of the values of the domain of A_i. By contrast, if A_i is a continuous (real-valued) attribute, the operator Op_i is either "≤" or ">", which will produce rule conditions such as "$A_i \leq V_{ij}$", where V_{ij} is a value belonging to the domain of A_i.

(c) identification of a set of attribute values $\{V_{i1},...,V_{ik}\}$, if the attribute A_i is categorical, or a single attribute value V_{ij}, if the attribute A_i is continuous, as explained in the previous item.

(d) a flag, called the active bit B_i, which takes on the value 1 or 0 to indicate whether or not, respectively, the i-th condition is present in the rule antecedent (phenotype).

The overall structure of the genome of an individual is illustrated in Figure 1.

A_1 $Op_1\{V_{1j}..\}$	B_1	...	A_i $Op_i\{V_{ij}..\}$	B_i	...	A_n $Op_n\{V_{nj}..\}$	B_n

Figure 1: Structure of the genome of an individual.

3.3 FITNESS FUNCTION

To clarify the definition of the fitness function used in our GA, we show in Figure 2 a well-known 2x2 confusion matrix for domains with two classes - arbitrarily called "positive" (+) and "negative" (-) classes. This matrix divides the classifications made by discovered rules predicting the "+" class on the test set into four categories, depending on whether or not a test example is covered by a rule predicting the "+" class, corresponding to the first and second rows, respectively; and on whether or not the test example has the "+" class, corresponding to the first and second columns, respectively (Hand 1997). The labels of each of the four quadrants of the matrix have the following meaning: TP = number of "+" examples that were correctly classified as "+" examples; FP = number of "-" examples that were wrongly classified as "+" examples; FN = number of "+" examples that were wrongly classified as "-" examples; TN = number of "-" examples that were correctly classified as "-" examples.

		True Class	
		"+"	"-"
Predicted	"+"	TP (true positive)	FP (false positive)
Class	"-"	FN (false negative)	TN (true negative)

Figure 2: A 2x2 confusion matrix for a domain with two classes

To evaluate the quality of an individual (candidate rule), our GA uses the following fitness function:

$Fitness = (TP / (TP + FN)) * (TN / (FP + TN))$.

For a comprehensive discussion about this and related rule-quality measures in general, independent of genetic algorithms, the reader is referred to (Hand 1997). Here we briefly mention that, in the above formula, the term $(TP / (TP + FN))$ is often called *sensitivity*, whereas the term $(TN / (FP + TN))$ is often called *specificity*. These two terms are multiplied to force the GA to discover rules that have both high sensitivity and high specificity, since it would be relatively simple to maximize one of these terms by reducing the other.

3.4 GENETIC OPERATORS

We use the well-known tournament method for reproduction, with tournament size of 2. We also use standard one-point crossover with crossover probability of 80%, and mutation probability of 1%. Furthermore, we use elitism with an elitist factor of 1 - i.e. the best individual of each generation is passed unaltered into the next generation.

In addition to the above standard genetic operators, we have also developed a new operator especially designed for improving the comprehensibility of rules. The basic idea of this operator, called rule-pruning operator, is to remove several conditions from a rule to make it shorter. In a high level of abstraction, removing conditions from a rule is a common way of rendering a rule more comprehensible, in the data mining literature. This operator is applied to every individual of the population, right after the individual is formed as a result of crossover and mutation operators.

We have devised a rule pruning procedure based on information theory. In essence, this procedure works as follows. First of all, it computes the information gain of each of the *n* rule conditions (genes) in the genome of the individual – see below. Then the procedure iteratively tries to remove one condition at a time from the rule. The smaller the information gain of the condition, the earlier its removal is considered and the higher the probability that it will be actually removed from the rule.

More precisely, in the first iteration the condition with the smallest information gain is considered. This condition is kept in the rule (i.e. its active bit is set to 1) with probability equal to its normalized information gain (in the range 0..1), and is removed from the rule (i.e. its active bit is set to 0) with the complement of that probability. Next the condition with the second smallest information gain is considered. Again, this condition is kept in the rule with probability equal to its information gain, and is removed from the rule with the complement of that probability. This iterative process is performed while the number of conditions occurring in the rule is greater than the minimum number of rule conditions - at present set to 2, as explained earlier - and the iteration number is smaller than or equal to the number of genes (maximum number of rule conditions) *n*.

The information gain of each rule condition $cond_i$ of the form $<A_i\ Op_i\ V_{ij}>$ is computed as follows (Quinlan 1993), (Cover & Thomas 1991):

$$InfoGain(cond_i) = Info(G) - Info(G|cond_i), \text{ where}$$

$$Info(G) = - \sum_{j=1}^{c} (|G_j|/|T| * \log_2(|G_j|/|T|))$$

$$Info(G|cond_i) =$$

$$\left[- (|V_i|/|T|) \sum_{j=1}^{c} ((|V_{ij}|/|V_i|) * \log_2(|V_{ij}|/|V_i|)) \right.$$

$$\left. - (|\neg V_i|/|T|) \sum_{j=1}^{c} ((|\neg V_{ij}|/|\neg V_i|) * \log_2(|\neg V_{ij}|/|\neg V_i|)) \right]$$

where G is the goal (class) attribute, c is the number of classes (values of G), $|G_j|$ is the number of training examples having the j-th value of G, $|T|$ is the total number of training examples, $|V_i|$ is the number of training examples satisfying the condition $<A_i\ Op_i\ V_{ij}>$, $|V_{ij}|$ is the number of training examples that both satisfy the condition $<A_i\ Op_i\ V_{ij}>$ and have the j-th value of G, $|\neg V_i|$ is the number of training examples that do not satisfy the condition $<A_i\ Op_i\ V_{ij}>$, and $|\neg V_{ij}|$ is the number of training examples that do not satisfy $<A_i\ Op_i\ V_{ij}>$ and have the j-th value of G.

```
/* n = number of genes = number of attributes available to compose rule
antecedent */
/* The i-th position of vectors Info_Gain_Cond[] contains the
information gain of the i-th condition. This is used as the probability that
the condition is active */
/* The i-th position of vector Sorted_Cond[] contains the id of the
   condition with the i-th smallest information gain */
BEGIN
   Min_N_Cond = 2;      /* Minimum number of conditions */
   FOR i = 1 TO n
       compute Info_Gain_Cond[i];   /* see text */
   END FOR
   sort the n conditions in increasing order of Info_Gain_Cond[i];
   FOR i = 1 TO n
       Sorted_Cond[i] = Id of condition with
            the i-th smallest information gain;
   END FOR
   Iteration_Id = 1;
   N_Act_Cond = number of active conditions (with active bit = 1) in
genome;
   WHILE (N_Act_Cond > Min_N_Cond) AND (Iteration_Id < n)
        Random_N = randomly-generated number in the range 0..1;
        IF Random_N < Info_Gain_Cond[Sorted_Cond[Iteration_Id]]
          THEN condition whose Id is Sorted_Cond[Iteration_Id]
                 is active (i.e. it occurs in the rule)
            ELSE condition whose Id is Sorted_Cond[Iteration_Id]
                 is not active (i.e. does not occurs in the rule)
END WHILE
END
```

Figure 3: Rule-pruning procedure applied to GA individuals

The use of the above rule-pruning procedure combines the stochastic nature of GAs, which is partly responsible for their robustness, with an information-theoretic heuristics for deciding which conditions compose a rule antecedent, which is one of the strengths of some well-known data mining algorithms. As a result of the action of this procedure, our GA tends to produce rules that have both a relatively small number of attributes and high-information-gain attributes, whose values are estimated to be more relevant for predicting the class of an example.

A more detailed description of our rule-pruning procedure is shown in Figure 3. As can be seen in this Figure, the above-described iterative mechanism for removing conditions from a rule is implemented by sorting the conditions in increasing order of information gain. From the viewpoint of the GA, this is a logical sort, rather than a physical one. In other words, the sorted conditions are stored in a data structure completely separated from the individual's data structure, so that there is no modification in the actual order of the conditions in the genome of the individual.

4 COMPUTATIONAL RESULTS

As a case study for evaluating the proposed GA, we have used the adult dataset, which is one of the largest public domain datasets in the well-known data repository of the UCI (University of California at Irvine), available at http://www.ics.uci.edu/~mlearn/MLRepository.html.

This dataset contains information about the USA census. The goal (class) attribute indicates whether or not the average annual salary of a person exceeds 50K dollars. This dataset contains 48842 examples and 14 attributes, out of which 6 are continuous and 8 are categorical.

In our experiments we have used the predefined division of the dataset into a training and a test set, with the former having 32561 examples and the latter having 16281 examples. The examples that had some missing value were removed from the data set. As a result, the number of examples was slightly reduced to 30162 and 15060 examples in the training and test set, respectively.

In each run of the GA, the population size is 200 individuals, and the GA is run for 50 generations. We used these parameter values because they are relatively common in the literature. We made no attempt to optimize these parameter values.

As described in section 3, our hybrid decision-tree/GA rule discovery method consists of using a GA to discover rules for classifying small-disjunct examples only - recall that large-disjunct examples are classified by the decision tree. Intuitively, the performance of our method will be significantly dependent on the definition of small disjunct.

In our experiments we have used a commonplace definition of small disjunct, based on a fixed threshold of the number of examples covered by the disjunct. The general definition is: "A decision-tree leaf is considered a small disjunct if and only if the number of examples belonging to that leaf is smaller than or equal to a fixed size *S*." We have done experiments with four different values for the parameter *S*, namely $S = 3$, $S = 5$, $S = 10$ and $S = 15$.

For each of these four S values, we have done five different experiments, varying the random seed used to generate the initial population of individuals. The results reported below, for each value of S, is an arithmetic average of the results over these five different experiments. Therefore, the total number of experiments is 20 (4 values of S * 5 different random seeds).

This methodology makes the results somewhat more reliable, since in GAs, as in other stochastic methods, the quality of the results may be somewhat influenced by the random seed.

Note that the actual number of GA runs is much more than 20. Indeed, in each of these 20 experiments we run the GA twice for each small disjunct, since the target dataset has two classes. In any case, each run of the GA is relatively fast, as argued in section 3. In reality, each of these 20 experiments took a processing time on the order of one hour on a Pentium II of 266 MHz with 64Mb of RAM.

We now report results comparing the performance of the proposed GA against C4.5, a well-known decision-tree induction algorithm (Quinlan 1993). We have used the default parameters of C4.5. To make the comparison fair, we have made no attempt to optimize GA parameters such as population size, number of generations, and probabilities of crossover and mutation. We used relatively common parameter values suggested in the literature, as described above.

Table 1: Results comparing our hybrid C4.5/GA against C4.5

	accuracy rate of C4.5 only			Accuracy rate of C4.5 / GA		
disjunct size (S)	large disjuncts	small disjuncts	over-all	large disjuncts	small disjuncts	over-all
3	0.80	0.51	0.786	0.80	0.49	0.784
5	0.81	0.52	0.786	0.81	0.49	0.783
10	0.84	0.52	0.786	0.84	0.77	0.826
15	0.84	0.53	0.786	0.84	0.86	0.844

The results are shown in Table 1. The first column of this Table indicates the size threshold *S* used to define small disjuncts. The next three columns report results produced by C4.5 alone, without using the GA. More precisely, the second and third columns of the Table report the accuracy rate on the test set achieved by C4.5 separately for examples classified by large-disjunct rules and small-disjunct rules. The fourth column reports the overall accuracy rate on the test achieved by C4.5, classifying both large- and small-disjunct examples. Note that the figures in this column are of course constant across all the rows, since its results refer to the case where all test examples are classified by C4.5 rules, regardless of the definition of small disjunct.

The next three columns report results produced by our hybrid C4.5/GA method. More precisely, the fifth column reports the accuracy rate on the test set for large-disjunct rules. The figures in this column are, of course, exactly the same as the figures in the second column, since our hybrid method also uses the C4.5 rules for classifying examples belonging to large disjuncts. In any case, we included this redundant column in the Table for the sakes of comprehensibility and completeness. The sixth column reports the accuracy rate on the test set for the small-disjunct rules discovered by the GA. Finally, the seventh column reports the overall accuracy rate on the test achieved by our hybrid C4.5/GA method, classifying both large- and small-disjunct examples.

Comparing the third column against the sixth column we can note two distinct patterns of results. Consider first the case where a disjunct is considered as small if it covers ≤ 3 or ≤ 5 examples. This case corresponds to the first and second rows of Table 1. In this case the performance of the rules produced by the GA is slightly inferior to the performance of the rules produced by C4.5. In any case, this small reduction of performance referring to small-disjunct rules has an even smaller, virtually-negligible impact on the overall accuracy rate, as can be seen by comparing the fourth and seventh columns of Table 1. For instance, in the second row the overall accuracy rate of C4.5 is 78.6%, while the overall accuracy of our hybrid C4.5/GA is 78.3%.

A different picture emerges when we consider the case where a disjunct is considered as small if it covers ≤ 10 or ≤ 15 examples. This case corresponds to the third and fourth rows of Table 1. Now the performance of the small-disjunct rules produced by the GA is much better than the performance of the small-disjunct rules produced by C4.5. For instance, in the fourth row the C4.5 rules have an accuracy rate of 53% whereas the GA rules have an accuracy rate of 86%. This improved accuracy has a considerable impact on the overall accuracy rate. For instance, in the fourth row the overall accuracy of C4.5 is 78.6%, while the overall accuracy of our hybrid C4.5/GA is 84.4%.

A possible explanation for these results is as follows. In the first case, where a disjunct is considered as small if it covers ≤ 3 or ≤ 5 examples, there are very few training examples available for each GA run. With so few examples the estimate of rule quality computed by the fitness function is far from perfect, and the GA does not manage to do better than C4.5. On the other hand, in the second case, where a disjunct is considered as small if it covers ≤ 10 or ≤ 15 examples, the number of training examples available for the GA is considerable higher - although still relatively low. Now the estimate of rule quality computed by the fitness function is significantly better. As a result, the characteristics of GA which make them suitable for discovering good small-disjunct rules - such as robustness, flexibility, and ability to cope well with attribute interaction - lead to the discovery of small-disjunct rules much better than the corresponding rules discovered by C4.5.

Despite the good results reported above, the current version of our method has a limitation. It implicitly assumes that each small disjunct contains examples of both positive and negative classes. This is necessary in order to discover rules from small disjuncts. This condition is satisfied by adult dataset. However, in some datasets some small disjuncts can contain only positive examples, with no negative example to support rule discovery. In these cases the current version of our method should not be directly applied to the small disjuncts in question. We are currently working on a new version of our system that solves this problem.

5 CONCLUSIONS AND FUTURE RESEARCH

The problem of how to discover good small-disjunct rules is very difficult, since these rules are error-prone due to the very nature of small disjuncts. Ideally, a data mining system should discover good small-disjunct rules without sacrificing the goodness of discovered large-disjunct rules.

Our proposed solution to this problem was a hybrid decision-tree/GA method, where examples belonging to large disjuncts are classified by rules produced by a decision-tree algorithm and examples belonging to small disjuncts are classified by rules produced by a genetic algorithm. In order to realize this hybrid method we have used the well-known C4.5 decision-tree algorithm algorithm and developed a new genetic algorithm tailored for the discovery of small-disjunct rules.

The proposed hybrid method was evaluated in a case study using the adult dataset. The performance of our new GA and corresponding hybrid C4.5/GA method depends significantly on the definition of small disjunct. In a nutshell, the results shows that: (a) there is no significant difference in the quality of the rules discovered by C4.5 alone and the rules discovered by our C4.5/GA method when a disjunct is considered as small if it covers ≤ 3 or ≤ 5 examples; (b) the quality of the rules discovered by our C4.5/GA method is considerably better than the quality of the rules discovered by C4.5 alone when a disjunct is considered as small if it covers ≤ 10 or ≤ 15 examples.

A disadvantage of our hybrid C4.5/GA method is that it is much more computationally expensive than the use of C4.5 alone. More precisely, in a training set with about 30000 examples our hybrid method takes on the order of one hour, while C4.5 alone takes on the order of a few seconds. However the extra processing is not too excessive, and it seems a small price to pay for the considerable increase in the predictive accuracy of the discovered rules.

In addition, scalability to larger datasets does not seem a problem so serious as one might think at first glance. Most of the processing time of our hybrid method is taken by the GA. However, the length of time taken by each GA run depends essentially on the definition of disjunct size, rather than on the size of the entire dataset. It is true that larger datasets tend to have a larger number of small disjuncts, which in turn would increase the processing time of our C4.5/GA method - due to an increase in the number of GA runs. However, this is a problem for any algorithm specifically designed for coping with small disjuncts. The point is that the processing time taken *per small disjunct* is relatively short even when using a genetic algorithm, since there are just a few examples in the training set of a small disjunct.

There are several possible directions for future research. An important one is to evaluate the performance of the proposed hybrid C4.5/GA method for different kinds of definition of small disjunct, e.g. relative size of the disjunct (rather than absolute size, as considered in this paper). It would also be useful to evaluate the performance of the proposed method in other datasets, to further validate the results presented in this paper. Another interesting research direction would be to compare the results of the proposed C4.5/GA method against rules discovered by the GA only, although in this case the design of the GA would have to be somewhat modified - e.g. the heuristics of attribute selection described in section 3.2 could not be used.

BIBLIOGRAPHY

K. Ali (1995). *Learning Probabilistic Relational Concept Descriptions*, PhD thesis, chapter 5. University of California, Irvine. USA.

T.M. Cover and J. A. Thomas (1991), *Elements of Information Theory*, John Wiley & Sons.

A. Danyluk and F. Provost (1993). Small Disjuncts in Action: Learning to Diagnose Errors in the Local Loop of the Telephone Network, *Proc. 10th International Conference Machine Learning*, 81-88.

A.A. Freitas (2000) Evolutionary Algorithms. Chapter of forthcoming *Handbook of Data Mining and Knowledge Discovery.* Oxford University Press, 2000.

D. Hand (1997). *Construction and Assessment of Classification Rules*, John Wiley & Sons.

R. Holte; L. Acker and B. Porter (1989). Concept Learning and the Problem of Small Disjuncts, *Proc. IJCAI – 89*, 813-818.

E. Noda; H.S. Lopes and A.A FREITAS (1999) Discovering interesting prediction rules with a genetic algorithm. *Proc. CEC-99*, 1322-1329.

F. Provost and J.M. ARONIS (1996). Scaling up inductive learning with massive parallelism. *Machine Learning 23(1)*, Apr. 1996, 33-46.

J. R Quinlan (1993). *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publisher.

K.M. Ting (1994). The Problem of Small Disjuncts: its remedy in Decision Trees, *Proc. 10th Canadian Conference on AI*, 91-97.

G.M. Weiss (1995). Learning with Rare Cases and Small Disjuncts, *Proc. 12th International Conference on Machine Learning*, 558-565.

G.M. Weiss (1998). The Problem with Noise and Small Disjuncts, *Proc. Int. Conf. Machine Learning (ICML – 98)*, 1998, 574-578.