A Genetic Algorithm for Discovering Small-Disjunct Rules in Data Mining

Deborah R. Carvalho[1,2], Alex A. Freitas [1]

[1] *Postgraduate Program in Applied Computer Science , Pontifícia Universidade Católica do Paraná (PUCPR), Brazil*
[2] *Computer Science Dept., Universidade Tuiuti do Paraná (UTP), Brazil*

Abstract

This paper addresses the well-known classification task of data mining, where the goal is to discover rules predicting the class of examples (records of a given data set). In the context of data mining, small disjuncts are rules covering a small number of examples. Hence, these rules are usually error-prone, which contributes to a decrease in predictive accuracy. At first glance, this is not a serious problem, since the impact on predictive accuracy should be small. However, although each small disjunct covers few examples, the set of all small disjuncts can cover a large number of examples. This paper presents evidence that this is the case in several data sets. This paper also addresses the problem of small disjuncts by using a hybrid decision-tree/genetic algorithm approach. In essence, examples belonging to large disjuncts are classified by rules produced by a decision-tree algorithm (C4.5), while examples belonging to small disjuncts are classified by a genetic algorithm specifically designed for discovering small-disjunct rules. We present results comparing the predictive accuracy of this hybrid system with the prediction accuracy of three versions of C4.5 alone in eight public domain data sets. Overall, the results show that our hybrid system achieves better predictive accuracy than all three versions of C4.5 alone.

Keywords: data mining, classification, genetic algorithm, rule discovery, small disjuncts

1. Introduction

In essence, data mining consists of extracting knowledge from data. The basic idea is that, intuitively, real-world databases contain hidden knowledge useful for decision making, if this hidden knowledge can be discovered. For instance, data about the previous sales of a company might contain hidden, implicit knowledge about which kind of product each kind of customer tends to buy. Hence, by analyzing that data, one can discover knowledge potentially useful for increasing the sales of the company.

Data mining is actually an interdisciplinary field, since there are many kinds of algorithms, derived from several different research areas (arguably, mainly machine learning and statistics), which can be used to extract knowledge from data. In this paper we follow a machine learning approach (rather than a statistical approach) for data mining [16]. More precisely, we are interested in discovering knowledge that is not only accurate, but also comprehensible for the user. The discovery of comprehensible knowledge is facilitated by the use of a high-level, rule-based knowledge representation – see below.

The data mining task addressed in this paper is classification. In this task each example (or database record) belongs to a class, which is indicated by the value of a goal attribute. This attribute can take on a small number of discrete values, each of them corresponding to a class. The goal of the classification algorithm is to predict the class of an example, based on the value of the other attributes (called predictor attributes) for that example. For a comprehensive review of the classification task the reader is referred to [10], while for a comprehensive review of data mining in general the reader is referred to [25]. A specialized review of data mining with evolutionary algorithms can be found in [7].

In the context of the classification task, the discovered knowledge is often expressed as a set of IF-THEN prediction rules of the form: IF <conditions> THEN <class>. From a logical viewpoint, typically the discovered rules are in disjunctive normal form, where each rule represents a disjunct and each rule condition

[1] R. Imaculada Conceição, 1155, Curitiba PR 80215-901
E-mail addresses: deborah@utp.br, alex@ppgia.pucpr.br
www.ppgia.pucpr.br/~alex
[2] Av. Comendador Franco, 1860, Curitiba PR. 80215-090

represents a conjunct. A small disjunct can be defined as a rule that covers a small number of training examples [11].

In general rule induction algorithms have a bias that favors the discovery of large disjuncts, rather than small disjuncts. This is due to the belief that specializations in the training set are unlikely to be valid in the test set. Hence, at first glance small disjuncts should not be included in the discovered rule set, since they tend to be error prone [11]. However, although each disjunct covers a small number of examples, the set of all small disjuncts can cover a large number of examples. For instance [4] reports 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.

In our previous work [1,2] we have proposed 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 genetic algorithm specifically designed for discovering small-disjunct rules.

However, that previous work had two limitations. First, it reported results for only two data sets. Second, it compared the performance of the hybrid system only with the default version of C4.5 [19]. This paper extends our previous work in three directions:

(a) we report more extensive results on eight data sets;
(b) we compare the performance of the hybrid system not only with the default version of C4.5, but also with two other versions of C4.5 more adapted to cope with small disjuncts; and
(c) we present an analysis of the number of small disjuncts found in each data set, showing that small disjuncts occur more often than one might think at first glance.

The remainder 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, focusing on the genetic algorithm, which was specifically designed for discovering small-disjunct rules. Section 4 reports the results of experiments evaluating the performance of our system on eight data sets. Finally, Section 5 presents conclusions and future research directions.

2. Related Work

Liu et al. present a new technique for organizing discovered rules in different levels of detail [12]. The algorithm consists of two steps. The first one is to find top-level general rules, descending down the decision tree from the root node to find the nearest nodes whose majority classes can form significant rules. They call these rules the top-level general rules. The second is to find exceptions, exceptions of the exceptions and so on. They determine whether a tree node should form an exception rule or not using two criteria: significance and simplicity. Some of the exception rules found by this method could be considered as small disjuncts. However, unlike most of the projects discussed below, the authors do not try to discover small-disjuncts rules with greater predictive accuracy. Their method was proposed only as a form of summarizing a large set of discovered rules. By contrast our work aims at discovering new small-disjunct rules with greater predictive power than the corresponding rules discovered by a decision tree algorithm. Next we discuss other research more related to our work.

Weiss and Hirsh present a quantitative measure for evaluating the effect of small disjuncts on learning [24]. The authors reported experiments with a number of data sets to assess the impact of small disjuncts on learning, especially when factors such as training set size, pruning strategy, and noise level are varied. Their results confirmed that small disjuncts do have a negative impact on predictive accuracy in many cases. However, they did not propose any solution for the problem of small disjuncts.

Holte et al. investigated three possible solutions for eliminating small disjuncts without unduly affecting the discovery of "large" (non-small) disjuncts [11], 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. Performance is estimated by using a statistical significance test. (c) Using a specificity bias for small disjuncts (without altering the bias for large disjuncts).

Ting proposed the use of a hybrid data mining method to cope with small disjuncts [21]. 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. Similarly, Lopes and Jorge discuss two techniques for rule and case integration[13]. Case-based learning is used when the rule base is exhausted. Initially, all the examples are used to induce a set of rules with satisfactory quality. The examples that are not covered by these rules are then handled by a case-based learning method. In this proposed approach, the paradigm is shifted from rule learning to case-based learning when the quality of the rules gets below a given threshold. If the initial examples can be covered with high-quality rules the case-based approach is not triggered. In a high level of abstraction, the basic idea of these two methods is similar to our hybrid decision-tree/genetic algorithm method. However, these two methods have the disadvantage that the IBL (or CBL) 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 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 [22]. However, these results have a limited utility in practice, since the analysis of this interaction was made possible by using artificially generated data sets. 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 did experiments showing that, when noise is added to real-world data sets, small disjuncts contribute disproportional and significantly for the total number of classification errors made by the discovered rules [23].

3. A Hybrid Decision-Tree/Genetic-Algorithm System for Rule Discovery

As mentioned in the Introduction, we present 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 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. Indeed, one of the drawbacks of decision tree-building algorithms is the fragmentation problem [8], where the set of examples belonging to a tree node gets smaller and smaller as the depth of the tree is increased, making it difficult to induce reliable rules from deep levels of the tree.

On the other hand, genetic algorithms are robust, flexible algorithms, which tend to cope well with attribute interactions [6,7,18]. Hence, they can be more easily tailored for coping with small disjuncts, which are associated with large degrees of attribute interaction [20,17].

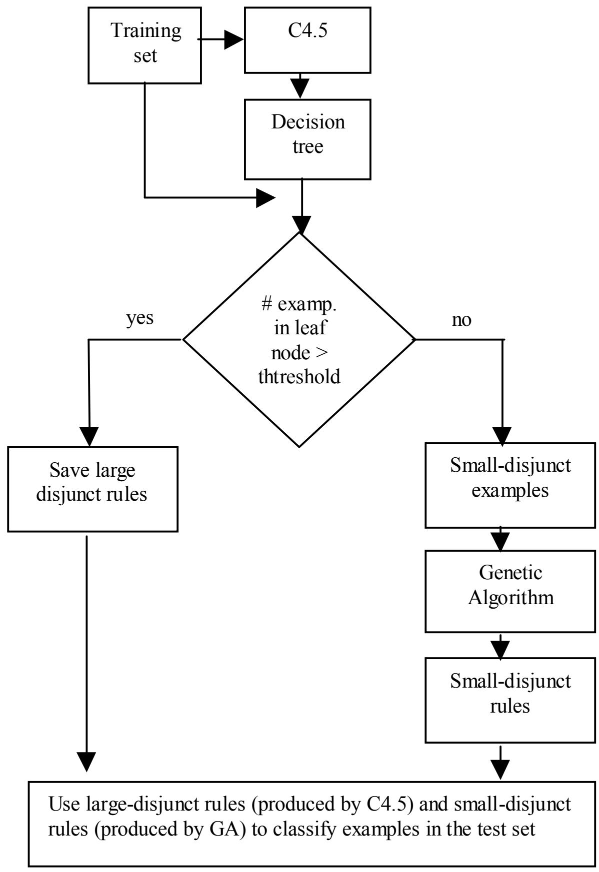


Figure 1. Overview of our hybrid Decision-Tree/Genetic-Algorithm system

The proposed system discovers rules in two training phases. In the first phase we run C4.5, a well-known decision tree induction algorithm [19]. 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. A large-disjunct rule is simply saved in order to classify new examples in the future (examples of the test set). By contrast, the small-disjunct rules discovered by C4.5 are not saved. Rather, the corresponding small-disjunct examples (i.e., the examples belonging to leaf nodes considered small disjuncts) are passed to a genetic algorithm (GA), triggering the second training phase. This second phase consists of using the GA to discover rules covering those small-disjunct examples. We have developed a GA specifically for this phase, as described in detail below. The small-disjunct rules discovered by this GA are then used to classify new examples in the future (examples of the test set). The basic idea of the entire process is shown in Figure 1.

3.1 Overview of a GA for Discovering Small-Disjunct Rules

In this section we describe our genetic algorithm (GA) developed for discovering small-disjunct rules – i.e., rules covering the examples in a decision tree's leaf nodes deemed small disjuncts, as explained above. A detailed description of this GA can also be found in [1,2]. In any case, the below description is intended to make this paper self-contained.

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 later. The GA also has a rule-pruning operator that favors the discovery of shorter, more comprehensible rules, as will be seen in Section 3.5.

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 that entire 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 i-th run of the GA, $i = 1,..., c$, discovers a rule predicting the i-th class.

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 comprising a given rule antecedent.

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 on 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 heuristic to select the subset of attributes that is used to comprise rule conditions.

Our heuristic is based on the fact that different small disjuncts identified by the decision-tree algorithm can have several ancestral 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 ancestral nodes of those two nodes. Then the two corresponding rule antecedents have k - 1 conditions in common. Therefore, it does not make sense to use these common conditions in the rules to be discovered by the GA, since they will not be good at discriminating between the two corresponding small disjuncts. Hence, for each small disjunct, the genome of a GA individual contains only the 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. 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_i is a logical/relational operator compatible with attribute A_i – see Figure 2. B_i represents an active bit, 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).

As an example, the i-th rule condition encoded in the genome could be a condition such as "Sex = female", where attribute A_i is "Sex", the operator Op_i is "=" and the value V_{ij} is "female". As another example, the i-th rule condition encoded in the genome could be "Salary < 20,000", where the operator Op_i is "<". In the former example the operator "=" was used because Sex is a categorical (nominal) attribute, whereas in the latter example the operator "<" was used because Salary is a continuous (real-valued) attribute.

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

Figure 2. Structure of the genome of an individual.

3.3 Fitness Function

Assume, without loss of generality, that there are two classes. Let positive ("+") class be the class predicted by a given rule, and let negative ("-") class be any class other than the class predicted by the rule. For instance, suppose that there are three classes, c_1, c_2, and c_3, and suppose that the rule being evaluated predicts class c_1. Then, class c_1 is considered the positive class, and classes c_2 and c_3 are grouped into a single negative class.

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

$$Fitness = (TP / (TP + FN)) * (TN / (FP + TN)) \qquad (1)$$

where

TP (true positives) = number of "+" examples that were correctly classified as "+" examples;

FP (false positives) = number of "-" examples that were wrongly classified as "+" examples;

FN (false negatives) = number of "+" examples that were wrongly classified as "-" examples;

TN (true negatives) = number of "-" examples that were correctly classified as "-" examples.

For a comprehensive discussion about this and related rule-quality measures in general, independent of genetic algorithms, the reader is referred to [10]. 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.

Note that our fitness function does not take into account rule comprehensibility. However, our GA has a rule-pruning operator that fosters the discovery of shorter, more comprehensible rules, as discussed in section 3.5.

3.4 Conventional Genetic Operators

We use the well-known tournament method for selection, with tournament size of 2 [14]. 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 [9].

The action of one-point crossover in our previously-described individual representation is illustrated in Figure 3, where the crossover point, denoted by a vertical line ("|"), fell between the second and the third genes. In each gene (corresponding to a rule condition) the number 1 or 0 between brackets denotes the value of the active bit flag, as explained in Section 3.2. Note that crossover points can fall only between genes, and not inside a gene. Hence, crossover swaps entire rule conditions between individuals, but it cannot produce new rule conditions.

Sex=male(0)	Age<25(1)	Employed=no(1)	Salary>50,000(0)
Sex=fem (1)	Age<25(1)	Employed=yes(1)	Salary>30,000(0)

(a) Parent individuals (before crossover)

Sex=male (0)	Age<25(1)	Employed=yes(1)	Salary>30,000(0)
Sex=fem (1)	Age<25 (1)	Employed=no(1)	Salary>50,000(0)

(b) Offspring (after crossover)

Figure 3: One-Point Crossover in our Individual Representation

The creation of new rule conditions is accomplished by the mutation operator, which replaces the attribute value of a condition (the element V_{ij} of Figure 2) with a new randomly-generated value belonging to the domain of the corresponding attribute. For instance, suppose the condition "Marital_Status = single" is encoded into the genotype of an individual. A mutation could modify this condition into the new condition "Marital_Status = divorced".

3.5 Rule Pruning Operator

We have developed an operator especially designed for improving the comprehensibility of candidate rules. The basic idea of this operator, called the 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 and machine learning literature (although details of the method vary a lot among algorithms). Our rule-pruning operator is applied to every individual of the population, right after the individual is formed as a result of crossover and mutation operators.

Unlike the usually simple operators of conventional GAs, our rule-pruning operator is an elaborate procedure based on information theory [3]. This procedure can be regarded as a way of incorporating a classification-related heuristic into a GA for rule discovery. The heuristic in question is to favor the removal of rule conditions with low information gain, while keeping the rule conditions with high information gain.

The rule-pruning operator works in an iterative fashion. 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 [19,3]:

$$\text{InfoGain}(\text{cond}_i) = \text{Info}(G) - \text{Info}(G|\text{cond}_i), \qquad (2)$$

where

$$\text{Info}(G) = -\sum_{j=1}^{c} (|G_j|/|T| * \log_2(|G_j|/|T|)) \qquad (3)$$

$$\text{Info}(G|\text{cond}_i) = \qquad (4)$$
$$-\ (|V_i|/|T|) \sum_{j=1}^{c} ((|V_{ij}|/|V_i|) * \log_2(|V_{ij}|/|V_i|))$$
$$-\ (|\neg V_i|/|T|) \sum_{j=1}^{c} ((|\neg V_{ij}|/|\neg V_i|) * \log_2(|\neg V_{ij}|/|\neg V_i|))$$

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.

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 heuristic 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.

```
/* n = number of genes = number of attributes available to compose
rule antecedent */
/* The i-th position of vectors Info_Gain_Cond[] contains the infor-
mation 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. , it does not occur in the rule)
   END WHILE
END
```

Figure 4: Rule-pruning procedure applied to GA individuals

A more detailed description of our rule-pruning procedure is shown in Figure 4. 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.

3.6 Classifying Examples in the Test Set

Recall that in our system each leaf node of the induced decision tree is deemed a small disjunct or a large disjunct, and that our method induces c rules for each of the d small disjuncts, where c is the number of classes and d is the number of small disjuncts.

Once all d x c runs of the GA are completed, examples in the test set are classified as follows. For each test example, we push the example down the decision tree until it reaches a leaf node. If that node is a large disjunct, the example is classified by the decision tree algorithm – i.e., it is assigned the majority class of the examples in the leaf node. Otherwise – i.e., the leaf node covering the example is a small disjunct – we try to classify the example by using one of the *c* rules discovered by the GA for the corresponding small disjunct.

At this point there are three possible outcomes. First, there might be more than one small-disjunct rule covering the example, since the coverage of the rules discovered by the GA can overlap with each other. In this case the example is classified by the highest-quality rule among all the rules covering the example, as estimated by the fitness function computed by the genetic algorithm. Second, there might be a single small-disjunct rule covering the example. In this case the example is simply classified by that rule. Third, there might be no small-disjunct rule covering the test example. In this case the example is classified by a default rule. In an earlier version of our system [1,2] we have experimented with two strategies for defining the default rule:

(1) Using a global default rule, which simply predicts the majority class among all small-disjunct examples. We call that a global default rule because the majority class in chosen considering all examples in all small disjuncts;

(2) Using a local default rule for each small disjunct. That rule simply predicts the majority class among the examples belonging to the current small disjunct.

For this paper we are only adopting the second strategy, using the local default rule. We decided to use only the local default rule because the difference between the results of both strategies was quite small, and adopting the local default rule has the advantage that the comparison of our system's results with C4.5's results is fairer, since C4.5 effectively uses a "local default rule".

4. Computational Results

We have evaluated our system on eight public domain data sets from the well-known data set repository at UCI (University of California at Irvine): Adult, Connect, CRX, Hepatitis, Segmentation, Splice, Voting and Wave. These data sets are available at the: http://www.ics.uci.edu/~mlearn/MLRepository.html.

Table 1 summarizes the data sets used in our experiments. The first column shows the data set identification, while the remaining columns report the number of attributes, the number of examples and the number of classes of the data set, respectively.

In our experiments we have used the predefined division of the Adult data set into a training and a test set. For the Connect data set, we have randomly partitioned this data set into a training set and a test set. In the case of these two data sets the use of a single training/test set partition is acceptable due to the relatively large size of the data. Since the other six data sets are not so large as the Adult and Connect data sets, to make the results more reliable we have run a tenfold cross-validation procedure for each of those six data sets. In other words, the data set was divided into ten partitions and our hybrid decision tree / genetic algorithm system was run ten times. In each run one of the ten partitions was chosen as the test set and all the other nine partitions were merged to compose the training set. The results reported below for those six data sets are an average over the ten runs.

Table 1. Main characteristics of data sets used in our experiment

data set	# atributes	# examples	# classes
Wave	21	5000	3
Hepatitis	19	155	2
Adult	14	45222	2
CRX	15	690	2
Voting	16	506	2
Connect	42	67557	3
Splice	60	3190	3
Segmentation	19	2310	7

As described in Section 3, our hybrid decision-tree/GA rule discovery system uses 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 system will be significantly dependent on the definition of a 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 report here experiments with two different values for the parameter S, namely $S = 10$ and $S = 15$.

We chose the values of S=10 and S=15 for our experiments for the following reasons. First, it is desirable that the value of S is not too small, since in this case there would always be too few examples for training the GA. Intuitively, values of S considerably smaller than 10 would have the disadvantage of producing disjuncts that are too small, with too few examples for inducing reliable rules. On the other hand, a considerably larger value of S is not desirable either, due two reasons. First, the GA was designed to discover rules covering only few examples, which has simplified its design, since each run of the GA has to discover only one rule per class. As the number of examples belonging to a small disjunct increases, ideally the GA should probably discover more than one rule per class for each small disjunct. This would require a significant chance in the design of the GA. In the current GA an individual represents a single rule, and there is no mechanism to enforce population diversity (such as niching). If one wanted the GA to discover more rules per class, such a mechanism would have to be added. Second, and perhaps even more important, a large value of *S* would beg the question of the meaning of a “small” disjunct. Our work is based on the assumption that in general C4.5 can correctly classify large-disjunct examples. The GA is called to classify only small-disjunct examples. If a leaf node in the tree produced by C4.5 has considerably more than 15 examples, intuitively C4.5 has enough examples to perform a good classification (otherwise it would probably have split on that node, producing a larger tree). Hence, in our experiments we used the values S =10 and S = 15 as reasonable trade-offs between the above-described conflicting problems. We emphasize, however, that we make no claim that the values S = 10 or S = 15 are optimal. (An optimization of the value of *S* could be done in future work, but the optimal value would probably be dependent on the data set.)

For each of these two *S* values, we have done ten different experiments, varying the random seed used to generate the initial population of individuals of the GA. The results reported below, for each value of S, are an arithmetic average of the results over these ten different experiments. Therefore, the total number of experiments is 20 (2 values of S * 10 different random seeds).

Note that the actual number of GA runs is much more than 20 for each data set. In each experiment we run the GA $c * d$ times, where c is the number of classes and d is the number of small disjuncts.

As the decision-tree component of our hybrid algorithm we have used C4.5, a well-known decision-tree algorithm [19]. We 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. More precisely, in all the experiments, in each run of the GA the population size is 200 individuals, the GA is run for 50 generations, and the probabilities of crossover and mutation are 80% and 1%, respectively.

We now report results evaluating the performance of our hybrid C4.5/GA system. We start by comparing the accuracy rate of our hybrid system against the accuracy rate of C4.5 alone. The results are shown in Table 2. The first column of this table indicates the data sets. The second column shows the accuracy rate on the test set achieved by C4.5, classifying both large-disjunct and small-disjunct examples. Note that the accuracy rate of C4.5 alone is independent of the definition of small disjunct (value of *S*). The next two columns report the overall accuracy rate on the test set achieved by our hybrid C4.5/GA system – i.e., using C4.5 to classify large-disjunct examples and our GA to classify small-disjunct examples. Each of those two columns reports the results for a specific value of *S* (small disjunct size). The values between brackets are standard deviations.

Table 2. Accuracy Rate of C4.5 and our hybrid C4.5/GA system

		Hybrid C4.5/GA	
data set	C4.5	S=10	S=15
Wave	75.54	**79.60 (2.0) +**	**80.30 (2.0) +**
Hepatitis	83.64	**84.97 (6.0)**	82.25 (5.0)
Adult	79.21	**79.83 (0.1) +**	**79.55 (0.1) +**
CRX	84.53	**86.12 (4.0)**	**86.37 (2.0)**
Voting	94.63	92.30 (1.0) -	91.72 (2.0)
Connect	72.6	**75.93 (0.8) +**	**74.62 (0.2) +**
Splice	45.98	**46.45 (0.9)**	**47.70 (2.0)**
Segmentation	97.67	93.62 (0.8) -	93.16 (1.0) -

Let us now discuss the results reported in Table 2. In the last two columns of this table the cells where the hybrid C4.5/GA system achieved a higher accuracy rate than C4.5 alone are shown in bold. Note that when S = 10 the hybrid system outperforms C4.5 alone in 6 out of the 8 data sets, and when S = 15 the former outperforms the latter in 5 out of the 8 data sets. In total, the hybrid system outperforms C4.5 alone in 11 out of 16 cases – i.e., in 68.8% of the cases. However, not all results are statistically significant. In the last two columns of Table 2 the cases where the better (worse) accuracy of the hybrid system is statistically significant is indicated by the “+” (“-“) symbol. A result was considered as statistically significant when the difference between the accuracy rate of the hybrid system and of C4.5 was larger than two standard deviations. Note that when S = 10 the hybrid system is significantly better than C4.5 in 3 data sets and the reverse is true in 2 data sets. When S = 15 the hybrid system is significantly better than C4.5 in 3 data sets, and the reverse is true in one data set. In total, the results of the hybrid system are significantly better than the results of C4.5 in 6 out of 16 cases (i.e., 37.5% of the cases), whereas the reverse is true in only 3 out of 16 cases (i.e., 18.8% of the cases).

Overall, the results of Table 2 show that the hybrid system has obtained somewhat better accuracy rates than the default version of C4.5 alone. However, to better evaluate the performance of the hybrid it is also important to compare it against other method(s) to cope with small disjuncts. In particular, we wanted to compare the hybrid system against another method that induces rules or trees (which can be straightforwardly converted to rules). In this case the kind of knowledge representation used by the systems being compared is the same, and the difference in the results will reflect mainly differences in search strategies. Hence, we can compare the evolutionary search strategy of the GA against the local, greedy search strategy of a rule induction or decision tree algorithm.

Within this spirit we now report the results of another experiment comparing our hybrid C4.5/GA system against a "double run" of C4.5. The later is a new way of using C4.5 to cope with small disjuncts, as follows. The main idea of our "double run" of a decision-tree algorithm is to build a classifier running the algorithm C4.5 twice. The first run considers all examples in the original training set, producing a first decision-tree. Once identified which examples belong to small disjuncts, the system groups all the examples belonging to small disjuncts (according to the first decision tree) into a single example subset. This can be thought of as a second training set. Then C4.5 is run again on this second, reduced training set producing a second decision tree.

In order to classify a new example, the rules discovered by both runs of C4.5 are used as follows. First, the system checks whether the new example belongs to a large disjunct of the first decision tree. If so, the class predicted by the corresponding leaf node is assigned to the new example. Otherwise (i.e. the example belongs to one of the small disjuncts of the first decision tree), the new example is classified by the second decision tree.

The motivation for this more elaborate use of C4.5 was an attempt to create a simple algorithm that was more robust to cope with small disjuncts.

The results comparing the accuracy rate (on the test set) of our hybrid system against the accuracy rate of "double C4.5" – denoted by C4.5(2) - are shown in Table 3.

Analogously to Table 2, the numbers between brackets in the third and fifth columns of Table 3 are standard deviations, the cells where the hybrid system outperforms "double C4.5" are shown in bold, and the cells where the hybrid system's results are significantly better (worse) than double C4.5's results contain the symbol "+" ("-"). Again, a result was considered statistically significant when the difference between the accuracy rate of the hybrid system and of double C4.5 was larger than two standard deviations.

As can be observed in the Table 3, in all 16 cases the accuracy rate of the hybrid system was better than the accuracy rate of double C4.5. Furthermore, the difference is statistically significant in 9 out of the 16 cases – i.e., in 56.3% of the cases.

Table 3. Accuracy Rate of "double C4.5" (denoted C4.5 (2)) and our hybrid C4.5/GA system

	S = 10		S = 15	
data set	C45(2)	C4.5/GA	C45(2)	C4.5/GA
Wave	69.97	**79.60 (2.0) +**	71.34	**80.30 (2.0) +**
Hepatitis	81.90	**84.97 (6.0)**	82.10	**82.25 (5.0)**
Adult	79.19	**79.83 (0.1) +**	78.81	**79.55 (0.1) +**
CRX	85.50	**86.12 (4.0)**	86.10	**86.37 (2.0)**
Voting	89.50	**92.30 (1.0) +**	90.60	**91.72 (2.0)**
Connect	49.90	**75.93 (0.8) +**	56.80	**74.62 (0.2) +**
Splice	46.33	**46.45 (0.9)**	46.84	**47.70 (2.0)**
Segmentation	44.10	**93.62 (0.8)+**	55.80	**93.16 (1.0) +**

Finally, we report the results of a third experiment, where we have compared the accuracy rate of our hybrid system against C4.5 without tree pruning. It is well known that in general (but not always) the results of C4.5 with pruning are better than the results of C4.5 without pruning. However, in the context of our work there is a motivation for evaluating the results of C4.5 without pruning. We are looking for small-disjunct rules, which tend to be more specific rules. Turning off C4.5 pruning does lead to more specific rules. Of course, there is a danger that C4.5 without pruning will overfit the data, and this approach will produce more specific rules not only for small-disjunct examples, but also for large-disjunct examples. In any case, it is worth trying C4.5 without pruning as a possible solution for the problem of small disjuncts, since this is a simple approach serving as a baseline solution for the problem of small disjuncts.

Hence, we report in Table 4 results comparing the accuracy rate (on the test set) of our hybrid system with the accuracy rate of C4.5 without pruning.

Analogously to Table 2 and Table 3, the numbers between brackets in the third and fourth columns of Table 4 are standard deviations, the cells where the hybrid system outperforms C4.5 without pruning are shown in bold, and the cells where the hybrid system's results are significantly better (worse) than the results of C4.5 without pruning contain the symbol "+" ("-"). Again, a result was considered statistically significant when the difference between the accuracy rate of the hybrid system and of C4.5 without pruning was larger than two standard deviations.

As can be observed in the Table 4, in 14 cases out of 16 – i.e., in 87.5% of the cases – the accuracy rate of the hybrid system was better than the accuracy rate of C4.5 without pruning, and the opposite was true in only 2 cases – i.e., in 12.5% of the cases. The results of the hybrid system were significantly better in 6 cases – i.e., in 37.5% of the cases; whereas the results of C4.5 without pruning were significantly better in only 2 cases – i.e., in only 12.5% of the cases.

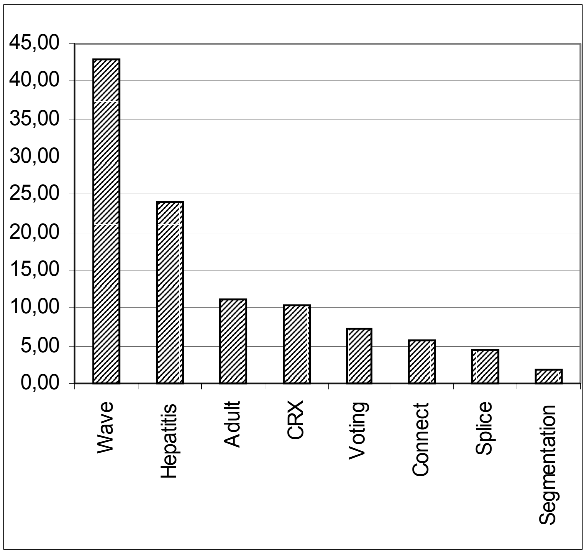
Table 4. Accuracy rate of our hybrid C4.5/GA system and C4.5 without pruning

data set	C4.5 without pruning	S = 10 C4.5/GA	S = 15 C4.5/GA
Wave	74.31	**79.60 (2.0) +**	**80.30 (2.0) +**
Hepatitis	77.67	**84.97 (6.0)**	**82.25 (5.0)**
Adult	76.50	**79.83 (0.1) +**	**79.55 (0.1) +**
CRX	85.04	**86.12 (4.0)**	**86.37 (2.0)**
Voting	91.04	**92.30 (1.0)**	**91.72 (2.0)**
Connect	69.10	**75.93 (0.8) +**	**74.62 (0.2) +**
Splice	46.30	**46.45 (0.9)**	**47.70 (2.0)**
Segmentation	97.08	93.62 (0.8) -	93.16 (1.0) **-**

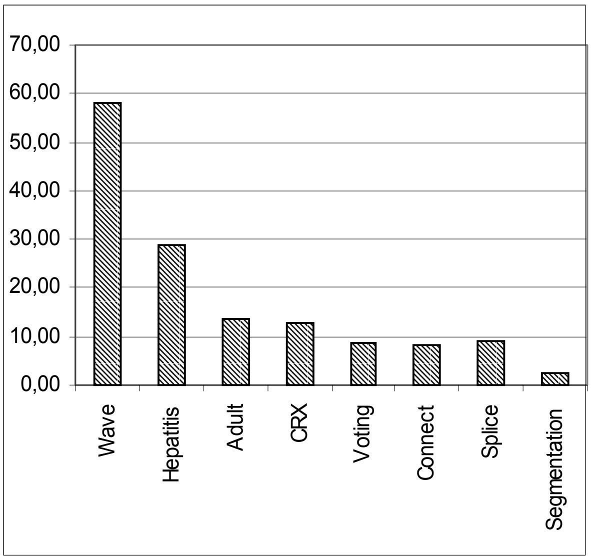
Figure 5 shows the percentage of training examples which belong to small disjuncts for S=10 and S=15. As discussed in the introduction, the percentage of small-disjunct examples is representative, particularly when S=15. Specifically, in the Wave data set more than 50% of the examples belong to small disjuncts.

Figure 5 provides a possible explanation for the relatively bad performance of our hybrid C4.5/GA method on the Segmentation data set (see the last rows of Tables 2 and 4). As shown in Figure 5, out of the 8 data sets used in our experiment, this is the one having the smallest proportion of examples belonging to small disjuncts. Hence, it is possible that C4.5/GA did not perform well on this data set because there were relatively few examples belonging to small disjuncts, hindering the performance of the GA component of the method. Further research is necessary to confirm if this remark can lead to a general pattern or if it holds only for the Segmentation data set.

We now turn to the issue of computational efficiency. Each run of the GA is relatively fast, since it uses a training set with just a few examples. However, recall that in order to discover all small disjunct rules we need to run the GA $c * d$ times, where c is the number of classes and d is the number of small disjuncts. Hence, it is important to know how long the entire set of $c * d$ GA runs takes. This processing time obviously varies with the number of small disjuncts, which depends on both the data set and on the definition of small disjunct (the value of the parameter S). In our experiments, the processing time taken by all $c * d$ runs of the GA was about 1 hour for the largest data set, Connect, and the largest number of small disjuncts, associated with the value $S = 15$. The experiments were performed on a Pentium II with 128 Mb of RAM and 166 MHz of clock. One hour seems to us a reasonable processing time, especially considering that in real-world applications the time to run a data mining algorithm is typically just about 10% or 20% of the total time spent with the knowledge discovery process [15].



(a) for $S = 10$



(b) for $S = 15$

Figure 5. Relative frequency of the small disjuncts found in the data sets used in our experiments

In addition, scalability to larger data sets does not seem a problem so serious as one might think at first glance. Most of the processing time of our hybrid system 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 system – 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.

Finally, if necessary the processing time taken by all the $c * d$ GA runs can be considerably reduced by using parallel processing techniques [5]. Actually, our method greatly facilitates the exploitation of parallelism in the discovery of small disjunct rules, since each GA run is completely independent from the others and it needs to have access only to a small data set, which surely can be kept in the local memory of a simple processor node.

5 Conclusions and Future Research

In this paper we have described a hybrid decision-tree (C4.5)/GA system, 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 developed specifically for discovering small disjunct rules.

More precisely, we have compared our hybrid C4.5/GA system with 3 algorithms based on the use of C4.5 alone, namely: (a) the default version of C4.5; (b) a "double run of C4.5", where all the small-disjunct examples found by the first run of C4.5 are grouped together into a "second training set", which is then used to build another tree in the second run of C4.5; (c) C4.5 without pruning. Overall, the hybrid system obtained better accuracy rates than *all* the 3 above-mentioned versions of C4.5 alone. More precisely, the accuracy rates obtained by the hybrid system were:

(a) somewhat better than the ones obtained both by default version of C4.5 and by C4.5 without pruning;
(b) considerably better than the ones obtained by a double run of C4.5.

We have also presented evidence that small disjuncts are more common in data sets than one might think at first glance.

There are several possible directions for future research. We are currently developing a new version of our hybrid system where all the small-disjunct examples are merged together into a second training set, but instead of running C4.5 on that set, we run a GA on that set. As mentioned in Section 4, this requires the GA to be extended with some niching method. In the long term, we envisage two research directions. An important one is to evaluate the performance of our hybrid C4.5/GA system for different kinds of definition of small disjunct, e.g. relative size of the disjunct (rather than absolute size, as considered in this paper). Another interesting research direction would be to compare the results of our system against rules discovered by the GA only, although in this case the design of the GA would have to be somewhat modified - e.g. by using a niching mechanism, as mentioned in Section 4.

References

[1] Carvalho, D.R; Freitas, A.A. A hybrid decision tree/genetic algorithm for coping with the problem of small disjuncts in Data Mining. *Proc 2000 Genetic and Evolutionary Computation Conf. (Gecco-2000), 1061-1068*. Las Vegas, NV, USA. July 2000.

[2] Carvalho, D.R.; Freitas A.A. A genetic algorithm-based solution for the problem of small disjuncts. Principles of Data Mining and Knowledge Discovery (Proc. 4th European Conf., PKDD-2000. Lyon, France). Lecture Notes in Artificial Intelligence 1910, 345-352. Springer-Verlag, 2000.

[3] Cover, T.M.; Thomas, J.A. *Elements of Information Theory*. John Wiley &Sons. 1991.

[4] Danyluk, A.P.; Provost, F.J. Small Disjuncts in Action: Learning to Diagnose Errors in the Local Loop of the Telephone Network, *Proc. 10th International Conference Machine Learning*, 81-88. Morgan Kaufmann, 1993.

[5] Freitas, A.A.; Lavington, S.H. *Mining Very Large Databases with Parallel Processing*. Kluwer. 1998.

[6] Freitas, A.A. Evolutionary Computation. Chapter of forthcoming *Handbook of Data Mining and Knowledge Discovery*. Oxford University Press, 2002.

[7] Freitas, A.A.. Data Mining and Knowledge Discovery with Evolutionary Algorithms. (Forthcoming book) Springer-Verlag, 2002.

[8] Friedman, J.H.; Kohavi, R.; Yun, Y. Lazy decision trees. *Proc. 1996 Nat. Conf. of the AAAI (AAAI-1996)*.

[9] Goldberg, D.E. Genetic Algorithms in Search, Optimization, and Machine Learning. New York: Addison-Wesley, 1989

[10] Hand, D. J. *Construction and Assessment of Classification Rules*, John Wiley & Sons. 1997.

[11] Holte, R.C.; Acker, L.E.; Porter, B.W. Concept Learning and the Problem of Small Disjuncts, *Proc. IJCAI – 89*, 813-818. 1989.

[12] Liu, B.; Hu, M.; Hsu, W. Multi-level Organization and Summarization of the Discovered Rules, *Proc. 6th ACM SIGKDD Int. Conf. On Knowledge Discovered & Data Mining (KDD– 2000)*, 208-220. ACM Press, 2000.

[13] Lopes, A.A.; Jorge, A. Integrating Rules and Cases in Learning via Case Explanation and Paradigm Shift, *Advances in Artificial Intelligence. (Proc. IBERAMIA – SBIA 2000)*. Lecture Notes in Artificial Intelligence 1952, 33-42. Springer Verlag 2000.

[14] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. 3rd Ed. Springer-Verlag. 1996.

[15] Michalski, R.; Kaufman, K. Data Mining and Knowledge Discovery: A Review of Issues and Multistrategy Approach, In: Ryszard S. Michalski; Ivan Bratko and Miroslav Kubat (Eds.). *Machine Learning and Data Mining Methods and Applications,* John Wiley & Sons Ltd., 1998.

[16] Mitchell, T. *Machine Learning*. McGraw-Hill, 1997.

[17] Nazar, K.; Bramer, M.A. Estimating concept difficulty with cross entropy. In: Bramer, M.A. (Ed.) *Knowledge Discovery and Data Mining*, 3-31. London: The Institution of Electrical Engineers, 1999.

[18] Noda, E.; Lopes, H.S.; Freitas, A.A. Discovering interesting prediction rules with a genetic algorithm. *Proc. Congress on Evolutionary Computation (CEC-99)*, 1322-1329. IEEE Press, 1999.

[19] Quinlan, J.R. *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publisher. 1993.

[20] Rendell, L.; Seshu, R. Learning hard concepts through constructive induction: framework and rationale. *Computational Intelligence 6*, 247-270. 1990.

[21] Ting, K. M. The Problem of Small Disjuncts: its remedy in Decision Trees, *Proc. 10^{th} Canadian Conference on AI*, 91-97. 1994.

[22] Weiss, G.M. Learning with Rare Cases and Small Disjuncts, *Proc. 12^{th} International Conference on Machine Learning (ICML-95)*, 558-565. Morgan Kaufmann, 1995.

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

[24] Weiss, G.M.; Hirsh, H. A Quantitative Study of Small Disjuncts, *Proc. of Seventeenth National Conference on Artificial Intelligence.* (AAAI-2000) 665-670 Austin, Texas. 2000.

[25] Witten, I.H.; Frank, E. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 2000.