Genetic Programming for Data Classification: Partitioning the Search Space

Jeroen Eggermont
Leiden Institute of Advanced Computer Science, Universiteit Leiden
P.O. Box 9512
Leiden, The Netherlands
jeggermo@liacs.nl

Joost N. Kok
Leiden Institute of Advanced Computer Science, Universiteit Leiden
P.O. Box 9512
Leiden, The Netherlands
joost@liacs.nl

Walter A. Kosters
Leiden Institute of Advanced Computer Science, Universiteit Leiden
P.O. Box 9512
Leiden, The Netherlands
kosters@liacs.nl

ABSTRACT

When Genetic Programming is used to evolve decision trees for data classification, search spaces tend to become extremely large. We present several methods using techniques from the field of machine learning to refine and thereby reduce the search space sizes for decision tree evolvers. We will show that these refinement methods improve the classification performance of our algorithms.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Systems Applications — *Information Search and Retrieval*;
I.2.8 [**Computing Methodologies**]: Artificial Intelligence — *Problem Solving, Control Methods, and Search*

Keywords

Genetic Programming, Data Classification

1. INTRODUCTION

Decision tree constructing algorithms for data classification such as ID3 [12] and C4.5 [11] are based on a common principle: *divide and conquer*. The algorithms attempt to divide a training-set T into multiple (disjoint) subsets so that each subset T_i belongs to a single target class. In its simplest form a training-set consisting of N records could be divided into N subsets $\{T_1, \ldots, T_N\}$ such that each subset is associated with a single record and target class. However, the predictive capabilities of such a classifier would be limited. Therefore algorithms like C4.5 try to construct more general decision trees by partitioning the domain of numerical attributes and thereby limiting the size of the constructed decision tree. Since finding the smallest decision tree consistent with a specific training-set is NP-complete [6], machine learning algorithms for constructing decision trees tend to be non-backtracking and greedy in nature.

Unlike the *divide and conquer* approach of machine learning algorithms, evolutionary algorithms do not directly construct a solution to a problem (e.g., a decision tree) but rather search for a solution in a space of possible solutions. The space in which such an evolutionary algorithm searches is usually very large and sometimes even infinite (at least theoretically). In this paper we continue the research started in [4] and show how the search space in which a tree-based Genetic Programming (GP) [1, 8] algorithm searches can be reduced by partitioning the domain of numerical valued attributes in a data set. By partitioning (or clustering) the numerical valued attributes the GP algorithm can concentrate on finding the right combination of attributes and decision tree shape. As we will demonstrate on a selection of classification problems this hybrid approach greatly increases the classification performance of our GP algorithms.

Note that, whereas greedy algorithms like C4.5 are aimed at locally optimizing a decision tree during construction, our GP algorithms perform a more global search through the space of possible trees. As a result our GP algorithms search in a substantially larger part of the search space.

The outline of this paper is as follows. We start by describing the decision tree format that we will be using. Next we define three decision tree representations by specifying the set of possible internal and external nodes that can occur in our trees. We then describe our experiments and results. Finally, in the last section we draw some conclusions and look at future research.

2. FULL ATOMIC REPRESENTATIONS

In this paper we use full atomic representations. A full atomic representation uses atoms in the internal and leaf nodes. Each atom is syntactically a predicate of the form (*attribute operator value*(s)), where *operator* is a function returning a Boolean value (e.g., $<$ or $=$). In the leaf nodes we have *class assignment* atoms of the form (*class* $:= C$), where C is a category selected from the domain of the attribute to be predicted. A small example tree is shown in Figure 1. A full atomic tree classifies an instance I by traversing the tree from root to leaf node. In each non-leaf node an atom is evaluated. If the result is true the right branch is traversed, else the left branch is taken. This process is repeated until a leaf node containing a *class assignment* node is reached,

SAC'04, March 14–17, 2004, Nicosia, Cyprus

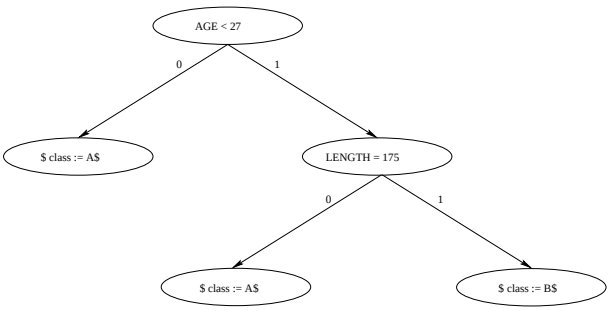

Figure 1: An example of a full atomic tree

resulting in the classification of the instance.

Multi-layered Fitness

In order to evolve trees which can be easily understood by humans, it is important to limit their size. We do this by pruning decision trees that become too large in combination with a *multi-layered* fitness. A multi-layered fitness is a fitness which consists of several fitness measures or objectives which are ranked according to their importance. Here we use a multi-layered fitness consisting of two fitness measures that we want to minimize. The primary, and most important, fitness measure is the misclassification percentage. The secondary fitness measure is the number of tree nodes. When the fitness of two individuals is to be compared we first look at the primary fitness. If both individuals have the same misclassification percentage we compare the secondary fitness measures.

2.1 A Simple Representation

By using a full atomic representation we have defined the basic shape of our decision trees. We can define the precise decision tree representation by specifying what atoms are to be used. The first representation is a simple, but powerful, decision tree representation that uses different types of atoms based on the data type of an atom's *attribute*. For non-numerical attributes we use atoms of the form ($variable_i$ = *value*) for each possible *attribute-value* combination found in the data set. For numerical attributes we define the less-than operator ($<$). Again we use atoms for each possible *attribute-value* combination found in the data set. The idea in this approach is to give the GP algorithm the most flexibility and let it decide on the best *attribute-value* combination at a given point in a tree.

2.2 Refined Representations

One of the drawbacks of the *simple* representation is the fact that it creates a possible internal node for each attribute-value combination that occurs in the data set. As a result the search space sizes become so large that it becomes difficult for the *simple* GP algorithm to achieve a decent classification performance. To reduce the huge number of possible internal nodes that are generated in this way, and thereby the search spaces, we will use ideas from machine learning. In C4.5 a single threshold value is selected to split the domain of a numerical valued attribute into two. Since C4.5 is a greedy non-backtracking algorithm it uses either the *gain* or *gain_ratio* criterion [11] to locally select the optimal threshold value for each node.

In our *refined* representations we also use the *gain* and *gain_ratio* criteria to split the domain of a numerical valued attribute into partitions. However, unlike C4.5, we determine $k-1$ threshold values resulting in k partitions. The *gain_ratio* criterion should be especially useful as it was designed to find a balance between information gained by splitting a data set into a large number of data subsets and limiting the number of subsets. Another difference with C4.5 is that we determine the set of optimal threshold values only once, based on the entire training-set, rather than for each node separately. By determining a "global" set of threshold values for each numerical valued attribute we hope to find more general trees and prevent over-fitting.

We will limit the maximum number of partitions to 5. If two sets of threshold values have the same *gain* or *gain_ratio* measures we will choose the set of threshold values containing the least values. In order to use the partitions specified by the optimal set of threshold values we need new types of atoms. If the optimal set of threshold values consists for instance of the three threshold values $threshold_1$, $threshold_2$ and $threshold_3$ we could construct atoms of the form

- $(attribute < threshold_1)$,
- $(attribute \in [threshold_1, threshold_2))$,
- $(attribute \in [threshold_2, threshold_3))$, and
- $(attribute \geq threshold_3)$.

2.3 Clustering Representation

A different method to partition the domain of numerical valued attributes is clustering. Clustering algorithms group similar collections of data together based on a measure of similarity. Unlike the partitioning methods described in Section 2.2, unsupervised clustering algorithms do not use the target class but rather divide the instances into "natural" groups. For our purposes we limit the clustering process to partitioning the domain of a single numerical valued attribute rather than clustering entire instances in the data set. Although we could use some kind of evolutionary algorithm for clustering, we use a K-means clustering algorithm [13] since it is fast and deterministic. More information about the *clustering* representation can be found in [3].

After the clustering algorithm has determined the clusters we can construct atoms based on the minimum and maximum value of each cluster. Thus, if the K-means clustering algorithm has found three clusters we could construct the following atoms:

- $(attribute \in [min_1, max_1])$,
- $(attribute \in [min_2, max_2])$, and
- $(attribute \in [min_3, max_3])$,

where min_i and max_i are the minimum and maximum value of cluster i respectively.

2.4 Example

Consider the simple example data set in Table 1. In the case of the *simple* representation we get the following atoms:

- Since attribute **A** has six possible values {1,2,3,4,5,6} and is numerical valued we use the $<$ operator: $(\mathbf{A} < 1)$, $(\mathbf{A} < 2)$, $(\mathbf{A} < 3)$, $(\mathbf{A} < 4)$, $(\mathbf{A} < 5)$ and $(\mathbf{A} < 6)$.

Table 1: Example data set

A	B	class
1	a	yes
2	b	yes
3	a	no
4	b	no
5	a	yes
6	b	yes

Table 2: The data sets used in the experiments

data set	records	attributes	classes
Australian credit (statlog)	690	14	2
German credit (statlog)	1000	23	2
Pima Indians diabetes	768	8	2
Heart disease (statlog)	270	13	2
Ionosphere	351	34	2
Iris	150	4	3

- Attribute **B** is non-numerical and thus we use the $=$ operator: $(\mathbf{B} = a)$ and $(\mathbf{B} = b)$.
- Finally for the target class we have two terminal nodes: $(class := yes)$ and $(class := no)$.

In the case of our *refined* representations we get two threshold values (3 and 5), using either the *gain* or *gain_ratio* criterion. Only two threshold values are chosen since they result in a "perfect" partitioning of attribute **A**'s domain, giving atoms: $(\mathbf{A} < 3)$, $(\mathbf{A} \in [3, 5))$, and $(\mathbf{A} \geq 5)$.

Using our K-means clustering algorithm with $k = 3$ results in three clusters for attribute **A**: $[1, 2]$, $[3, 4]$ and $[5, 6]$, respectively. Thus, in this case the following atoms are constructed: $(\mathbf{A} \in [1, 2])$, $(\mathbf{A} \in [3, 4])$, and $(\mathbf{A} \in [5, 6])$.

3. EXPERIMENTS AND RESULTS

We will compare our GP algorithms to C4.5 and a number of other evolutionary approaches using data sets from the UCI machine learning data set repository [2]. An overview of the data sets used is given in Table 2.

A single GP implementation was used for both *simple* and *refined* representations. It was programmed using the *Evolving Objects* library (EOlib) [7]. EOlib is an Open Source C++ library for all forms of evolutionary computation and is available from `http://eodev.sourceforge.net`.

In our GP system we use the standard GP mutation and recombination operators for trees. The mutation operator replaces a subtree with a randomly created subtree and the crossover operator exchanges subtrees between two individuals. Both the mutation rate and crossover rate are set to 0.9. The population was initialized using the ramped half-and-half initialization [1, 8] method to create a combination of full and non-full trees with a maximum tree depth of 6. We used a generational model (comma strategy) with population size of 100, an offspring size of 200 and a maximum of 99 generations. Parents were chosen by using 5-tournament selection. We did not use elitism as the best individual was stored outside the population. Each newly created individual, whether through initialization or recombination, was automatically pruned to a maximum number of 63 nodes.

Each algorithm is evaluated using a 10-fold cross-validation and the performance is the average misclassification error over 10 folds. In 10-fold cross-validation the total data set is divided into 10 parts. Each part is chosen once as the test set while the other 9 parts form the training set.

Table 3: Australian credit data set results.

algorithm	k	average	s.d.	best	worst	rank
clustering GP	2	**13.7**	0.8	12.5	14.8	1
clustering GP	3	14.8	0.7	13.8	16.1	3
clustering GP	4	14.8	0.4	14.3	15.7	4
clustering GP	5	15.2	0.7	13.5	15.8	8
refined GP (*gain*)	2	14.2	0.4	13.5	14.9	2
refined GP (*gain*)	3	15.1	0.8	14.9	16.4	7
refined GP (*gain*)	4	14.9	0.9	13.3	16.5	5
refined GP (*gain*)	5	15.1	0.6	13.9	16.4	6
refined GP (*gain_ratio*)	2	15.7	0.4	14.9	16.4	12
refined GP (*gain_ratio*)	3	15.5	0.1	15.4	15.7	9
refined GP (*gain_ratio*)	4	15.5	0.3	15.1	15.9	10
refined GP (*gain_ratio*)	5	15.6	0.4	15.1	16.1	11
simple GP		22.0	3.0	17.0	25.7	14
C4.5 $*$		15.9				13
Bagged C4.5		N/A				
Boosted C4.5		N/A				
CEFR-MINER		N/A				
ESIA		19.4	0.1			15

In order to compare our results to other evolutionary techniques we will also mention the results of two other evolutionary classification systems, CEFR-MINER [10] and ESIA [9], as reported in these respective papers. CEFR-MINER is a GP system for finding fuzzy decision trees and ESIA builds crisp decision trees using a genetic algorithm. Both algorithms also used a 10-fold cross-validation. To compare our results to non-evolutionary decision tree algorithms, we will also report the results of C4.5 and its Bagged and Boosted versions from [5]. We performed 10 independent runs for our GP algorithms to obtain the results. When available from the literature the results of CEFR-MINER, ESIA and C4.5 are reported. For three data sets no results were reported for C4.5 and its Bagged and Boosted versions. In those three instances, marked with a $*$, we applied C4.5 to the data set ourselves. N/A indicates that no results were available. Since C4.5 is a deterministic algorithm the standard deviation (s.d.), best and worst performance are not reported. The tables with results also contain an extra column, labeled k, to indicate the number of clusters in the case of our *clustering* GP algorithms or the maximum number of partitions in the case of the *refined* GP algorithms. The best (average) result for each data set is printed in bold font.

The Australian Credit Data Set

The results on the Australian credit data set are displayed in Table 3. The best classification performance is offered by our *clustering* GP algorithm with 2 clusters per numerical valued attribute. The second best performance is achieved by our *refined* GP algorithm using the *gain* criterion with 2 partitions per numerical valued attribute. An inspection of the atoms of the two algorithms shows that for most attributes the partitions and clusters are different. In this case the atoms created through clustering are clearly better. The classification performance of the *refined* GP algorithms using the *gain_ratio* criterion is not very good, but still better on average than C4.5 or our *simple* GP. A positive aspect of the *refined* GP algorithms using the *gain_ratio* criterion is that the standard deviations are lower than for our other algorithms.

Table 4: German credit data set results

algorithm	k	average	s.d.	best	worst	rank
clustering GP	2	27.8	0.7	26.3	28.8	4
clustering GP	3	28.0	0.8	27.0	29.8	6
clustering GP	4	27.9	0.9	26.7	29.4	5
clustering GP	5	28.4	0.8	26.9	29.5	11
refined GP (*gain*)	2	28.1	0.8	26.9	29.9	7
refined GP (*gain*)	3	27.1	0.8	26.2	29.0	2
refined GP (*gain*)	4	28.3	0.7	26.9	29.3	9
refined GP (*gain*)	5	28.2	0.6	27.1	29.3	8
refined GP (*gain_ratio*)	2	28.3	0.5	27.5	29.0	10
refined GP (*gain_ratio*)	3	28.5	0.6	27.9	29.5	12
refined GP (*gain_ratio*)	4	28.6	0.5	27.6	29.3	14
refined GP (*gain_ratio*)	5	28.5	0.5	27.8	29.3	13
simple GP		**27.1**	0.7	24.3	28.5	1
C4.5 $*$		27.2				3
Bagged C4.5		N/A				
Boosted C4.5		N/A				
CEFR-MINER		N/A				
ESIA		29.5	0.2			15

Table 5: Pima Indians diabetes data set results.

algorithm	k	average	s.d.	best	worst	rank
clustering GP	2	26.3	0.5	25.8	27.3	5
clustering GP	3	26.3	0.6	25.4	27.6	6
clustering GP	4	26.7	0.8	25.5	28.0	10
clustering GP	5	26.5	0.9	25.2	28.1	8
refined GP (*gain*)	2	27.0	0.6	26.0	28.0	11
refined GP (*gain*)	3	26.5	0.8	25.0	27.6	9
refined GP (*gain*)	4	25.9	0.8	24.2	26.8	3
refined GP (*gain*)	5	25.9	0.5	25.1	26.6	4
refined GP (*gain_ratio*)	2	27.6	0	27.6	27.6	12
refined GP (*gain_ratio*)	3	27.6	0	27.6	27.6	12
refined GP (*gain_ratio*)	4	27.7	0	27.7	27.7	14
refined GP (*gain_ratio*)	5	27.7	0	27.7	27.7	14
simple GP		26.3	1.1	24.3	28.5	7
C4.5		28.4				16
Bagged C4.5		**24.4**				1
Boosted C4.5		25.7				2
CEFR-MINER		N/A				
ESIA		29.8				17

Table 6: Heart disease data set results.

algorithm	k	average	s.d.	best	worst	rank
clustering GP	2	19.9	1.0	18.1	21.1	4
clustering GP	3	21.3	1.3	19.6	23.0	8
clustering GP	4	22.5	1.3	20.4	24.1	13
clustering GP	5	22.1	2.3	17.4	25.2	11
refined GP (*gain*)	2	19.9	2.5	15.6	24.4	3
refined GP (*gain*)	3	22.8	2.0	18.9	25.2	14
refined GP (*gain*)	4	22.1	1.5	20.4	25.2	10
refined GP (*gain*)	5	21.5	1.6	19.6	24.8	9
refined GP (*gain_ratio*)	2	18.7	0.7	17.4	20.0	2
refined GP (*gain_ratio*)	3	20.3	1.6	17.4	22.6	6
refined GP (*gain_ratio*)	4	20.6	1.1	18.9	22.6	7
refined GP (*gain_ratio*)	5	20.0	1.2	17.8	21.5	5
simple GP		25.2	2.3	22.6	31.1	16
C4.5 $*$		22.2				12
Bagged C4.5		N/A				
Boosted C4.5		N/A				
CEFR-MINER		**17.8**	7.1			1
ESIA		25.6	0.3			15

The German Credit Data Set

In Table 4 we see that only one of our GP algorithms can beat C4.5 and perform almost as well as our *simple* GP algorithm. Apparently, it is very difficult to correctly cluster or partition the domain of the numerical valued attributes of this data set. One reason for the relatively good performance of our *simple* GP algorithm on this data set might be the relatively small number of possible internal nodes when compared to Australian credit data set. The results of our *refined* GP algorithm using the *gain_ratio* criterion are again worse than those of our *clustering* and other *refined* GP algorithms.

The Pima Indians Diabetes Data Set

On the Pima Indians diabetes data set (see Table 5) the *refined* GP algorithms using the *gain* criterion are again better than those using the *gain_ratio* criterion. If we compare the results of our *clustering* GP algorithm with the results of our *refined* GP algorithms using the *gain* criterion we see that the differences are very small and not significant. On this data set none of our GP algorithms manages to perform better than our *simple* GP algorithm. Our GP algorithms are better than C4.5 but not better than its bagged and boosted versions. Although not particularly good, the results of the *refined* GP algorithms using the *gain_ratio* criterion were the most surprising. All *refined* GP algorithms using the *gain_ratio* have a standard deviation of 0. An analysis of the results showed that the misclassification rate per fold was the same for every used random seed. In the case of $k = 2$ and $k = 3$, the algorithms have exactly the same misclassification performance while using different the sets of internal nodes. The same behavior is seen for $k = 4$ and $k = 5$. In all cases the discovered decision trees differ syntactically per fold and random seed.

The Heart Disease Data Set

The results on the Heart disease data set are displayed in Table 6. All our GP algorithms show a large improvement in misclassification performance over our *simple* GP algorithm. In all but two cases our GP algorithms are also better than C4.5. On this data set our *refined* GP algorithms using the *gain_ratio* criterion perform a little better than the *gain* criterion variant. CEFR-MINER is still the best performing algorithm, but the difference with our best *(refined) full atomic* GP algorithm using the *gain_ratio* criterion is very small (less than 1%) and our algorithm has a much smaller standard deviation. When we look at the number of clusters or maximum number of partitions we see that a maximum of 2 clusters or partitions is clearly the best for this data set.

The Ionosphere Data Set

If we look at the results on the Ionosphere data set in Table 7 we see that using the *gain_ratio* instead of the *gain* criterion with our *refined* GP algorithms greatly improves the classification performance. Only our *refined* GP algorithms using the *gain_ratio* criterion also manages to beat C4.5 regardless of the maximum number of partitions we use. The classification performance of our *clustering* GP algorithms is disappointing as only our *clustering* GP algorithm with 3 clusters per numerical valued attribute manages to really outperform our *simple* GP but still performs much worse than C4.5.

The Iris Data Set

If we look at the results of our GP algorithms on the Iris data set in Table 8 we see that by far the best performance is achieved by our *clustering* GP algorithm with 3 clusters per numerical valued attribute. On this data set the *refined* GP algorithms using the *gain_ratio* criterion splits the domain

Table 7: Ionosphere data set results

algorithm	k	average	s.d.	best	worst	rank
clustering GP	2	13.1	0.9	11.4	14.2	1
clustering GP	3	10.5	1.2	8.8	13.4	9
clustering GP	4	12.1	1.3	9.4	14.0	14
clustering GP	5	13.3	2.1	10.8	17.4	16
refined GP (*gain*)	2	8.3	1.0	7.1	10.8	5
refined GP (*gain*)	3	10.5	1.1	9.1	12.5	10
refined GP (*gain*)	4	10.8	0.6	9.9	12.0	11
refined GP (*gain*)	5	11.6	1.7	8.8	15.1	13
refined GP (*gain_ratio*)	2	7.7	0.7	6.8	9.1	3
refined GP (*gain_ratio*)	3	8.1	0.8	7.1	9.4	4
refined GP (*gain_ratio*)	4	8.3	0.9	6.5	10.0	5
refined GP (*gain_ratio*)	5	9.1	1.0	7.1	10.2	8
simple GP		12.4	1.8	8.0	14.3	15
C4.5		8.9				7
Bagged C4.5		6.2				2
Boosted C4.5		**5.8**				1
CEFR-MINER		11.4	6.0			12
ESIA		N/A				

Table 8: Iris data set results

algorithm	k	average	s.d.	best	worst	rank
clustering GP	2	21.1	0.3	20.7	21.3	13
clustering GP	3	**2.1**	0.2	2.0	2.7	1
clustering GP	4	5.2	0.7	4.0	6.0	7
clustering GP	5	6.0	0.8	4.7	7.3	10
refined GP (*gain*)	2	29.6	0.3	29.3	30.0	14
refined GP (*gain*)	3	6.3	0.3	6.0	6.7	11
refined GP (*gain*)	4	5.1	0.7	4.0	6.0	6
refined GP (*gain*)	5	6.5	1.0	5.3	8.0	12
refined GP (*gain_ratio*)	2	31.7	0.9	30.7	32.7	15
refined GP (*gain_ratio*)	3	31.7	0.9	30.7	32.7	15
refined GP (*gain_ratio*)	4	31.7	0.9	30.7	32.7	15
refined GP (*gain_ratio*)	5	31.7	0.9	30.7	32.7	15
simple GP		5.6	1.1	3.3	7.3	8
C4.5		5.9				9
Bagged C4.5		5.0				4
Boosted C4.5		5.0				4
CEFR-MINER		4.7	7.1			2
ESIA		4.7	0.0			2

of the numerical valued attributes into 2 partitions regardless of the maximum allowed number of partitions. This is probably the reason for the bad misclassification rate of these algorithms as both refined representations also classify badly when the domain of the numerical valued attributes is split into two partitions or clusters. If we compare our *clustering* GP algorithms with the *refined* GP algorithm using the *gain* criterion we must conclude that in this case clustering performs better.

4. CONCLUSIONS

The results of our experiments show a clear indication that the choice of which nodes are going to be used when evolving decision trees can be crucial. With the exception of the German credit data set our *clustering* and *refined* GP algorithms manage to outperform our *simple* GP algorithm on all data sets. In the case of the Australian credit and Heart disease data set our new algorithms are always better regardless of the maximum number of partitions or clusters we use. On the other data sets our new algorithms are only occasionally better but the improvements can be significant (like for instance on the Iris and Ionosphere data set).

With regards to future research we are looking at several areas. In [3] we combined clustering, Genetic Programming and fuzzy logic to evolve fuzzy decision trees. These fuzzy decision trees offer better classification performance than similar non-fuzzy trees. Similarly, our *refined* GP algorithms using the *gain* and *gain_ratio* criteria can be transformed into fuzzy decision tree evolvers. We are also studying various methods to reduce the time spend on fitness evaluations while maintaining classification accuracy. Another area of interest is improving the non-evolutionary part by looking at heuristics other than the *gain* and *gain_ratio*.

5. REFERENCES

[1] W. Banzhaf, P. Nordin, R. Keller, and F. Francone. *Genetic Programming: An Introduction*. Morgan Kaufmann, 1998.

[2] C. Blake and C. Merz. UCI repository of machine learning databases, 1998.

[3] J. Eggermont. Evolving fuzzy decision trees with genetic programming and clustering. In J. Foster, E. Lutton, J. Miller, C. Ryan, and A. Tettamanzi, editors, *Proceedings on the Fifth European Conference on Genetic Programming (EuroGP'02)*, volume 2278 of *LNCS*, pages 71–82. Springer-Verlag, 2002.

[4] J. Eggermont, J. Kok, and W. Kosters. Genetic programming for data classification: Refining the search space. In T. Heskes, P. Lucas, L. Vuurpijl, and W. Wiegerinck, editors, *Proceedings of the 15th Belgium/Netherlands Conference on Artificial Intelligence (BNAIC'03)*, pages 123–130, 2003.

[5] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Proc. 13th International Conference on Machine Learning*, pages 148–146. Morgan Kaufmann, 1996.

[6] L. Hyafil and R. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[7] M. Keijzer, J. J. Merelo, G. Romero, and M. Schoenauer. Evolving objects: A general purpose evolutionary computation library. In P. C. et al., editor, *Proceedings of Evolution Artificielle'01*, volume 2310 of *LNCS*, pages 231–244. Springer Verlag, 2001.

[8] J. Koza. *Genetic Programming*. MIT Press, 1992.

[9] J. Liu and J. Kwok. An extended genetic rule induction algorithm. In *Proc. of the 2000 Congress on Evolutionary Computation*, pages 458–463, Piscataway, NJ, 2000. IEEE Service Center.

[10] R. Mendes, F. Voznika, A. Freitas, and J. Nievola. Discovering fuzzy classification rules with genetic programming and co-evolution. In L. de Raedt and A. Siebes, editors, *5th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'01)*, volume 2168 of *LNAI*, pages 314–325. Springer Verlag, 2001.

[11] J. Quinlan. C4.5: Programs for machine learning, 1993.

[12] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

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