Complete Expression Trees for Evolving Fuzzy Classifier Systems with Genetic Algorithms and Application to Network Intrusion Detection

Jonatan Gómez
Division of Computer Sciences
Mathematical Sciences Department
The University of Memphis and
Universidad Nacional de Colombia
jgomez@memphis.edu

Dipankar Dasgupta
Division of Computer Sciences
Mathematical Sciences Department
The University of Memphis
Memphis, TN 38152.
ddasgupt@memphis.edu

Olfa Nasraoui
Dept. of Electrical & Computer Engineering
The University of Memphis
onasraou@memphis.edu

Fabio Gonzalez
Division of Computer Sciences
Mathematical Sciences Department
The University of Memphis and
Universidad Nacional de Colombia
fgonzalz@memphis.edu

Abstract

We propose a new linear representation scheme for evolving fuzzy rules using the concept of complete binary tree structures. *We also use special genetic operators such as* gene addition, gene deletion, and variable length crossover. *Results show that using these special operators along with the common mutation operator produce useful and minimal structure modifications to the fuzzy expression tree represented by the chromosome. The proposed method (*representation and operators*) is tested with a number of benchmark data sets including the KDDCup'99 Network Intrusion Detection data.*

1. Introduction

The task of generating classifiers based on rules (fuzzy and classical) with genetic algorithms and other evolutionary techniques has been studied for years [1], [2], [3]. One of the main problems in the evolution of rules is the way that a rule is codified in a chromosome. This problem arises from the fact that the condition part of the rule can be an arbitrary logical expression. In general, the condition part of a rule has a tree structure and there is no natural way to represent it with a linear structure. There are basically three approaches to deal with the problem of representing a rule's condition part as a linear string:

1. Conjunctions of simple terms: In this approach the condition part is restricted to be a conjunction of one or more simple logic terms [1], [3], [4], [5], [6], and [7]. The problem with this codification is that a single rule is not enough to characterize a single class.
2. Fixed condition structure: In this approach a condition structure is predefined as a template and only the simple logic terms and the operators are varied [8]. The problem with this approach is the template has to be developed according to the problem to solve.
3. Linear-Tree with precedence representation: In this approach a precedence value for the logic operators in the condition is codified together with the operator [9]. In this way any logic expression can be codified, because the precedence value defines the order of the operators evaluation. The problem with this approach is that the genetic operators can significantly disrupt the tree structure of the condition, for example, if a precedence value is modified.

The purpose of the work presented in this paper is to explore a linear representation scheme for fuzzy rules based on the concept of complete binary trees. For this representation, we use some special genetic operators such as gene addition, gene deletion, and variable length crossover, along with the common mutation operator. It is shown that the genetic operators make minimal tree structure modifications, i.e., they cause little disruption over the fuzzy expression tree structure space. Besides, the evaluation of the represented fuzzy rule conditions can be performed in an efficient way, $O(n)$.

The subsequent sections are organized as follows. Section 2 presents the concept of complete binary trees. Section 3 describes the classification process using fuzzy classifiers. Section 4 presents the proposed method to evolve complete fuzzy rules. Section 5 shows some experimental results and the analysis performed. Section 6 draws some conclusions.

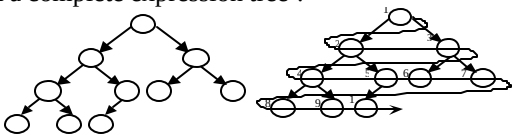
2. Complete binary trees

A binary tree is called ***h*-completely full** if it has height h and $2^{h+1}-1$ nodes (NITS).

A binary tree is called ***h*-complete** if one of the following conditions is satisfied:
1. If the left tree is (h-1)-completely full and the right tree is (h-1)-complete
2. If the left tree is (h-1)-complete and the right tree is (h-2) completely full
3. If h=0 and the tree is empty

Intuitively, a h-complete tree is a binary tree that is filled completely on all the levels $0, 1, \ldots, h$, except possibly the level

h that is filled from left to right. This structure is usually called **heap** [10]. Figure 1.a shows a complete binary tree sample. Nodes in a complete binary tree can be enumerated in a simple way, starting from the node at the root of the tree and traversing the tree level by level from left to right, as is shown in figure 1.b. This enumeration is called **level enumeration**. It can be proved that each expression tree can be represented with a complete expression tree[1].



(a) **(b)**

Figure 1: (a) Complete binary tree. (b) Binary tree level enumeration (similar to breadth first search).

2. Fuzzy Classifiers

An atomic fuzzy expression is an expression: *attribute is [not] fuzzyset*, where, *attribute* is a characteristic attribute, and *fuzzyset* is the fuzzy set [11] name that has been defined by a fuzzy membership function. The truth-value is the degree of membership of the attribute to the fuzzy set. Fuzzy rules have the form: *IF* fuzzy expression *THEN* consequence, where *fuzzy expression* is a logic expression, which uses fuzzy logic operators and atomic fuzzy expressions, and *consequence* is an atomic expression. A fuzzy classifier can be represented by a set of *m* rules, where *m* is the number of different classes (a rule per class):

R_1: IF condition$_1$ THEN data is class$_1$

....

R_m: IF condition$_m$ THEN data is class$_m$.

There are several defuzzification techniques to determine the class label. One of these techniques is the maximum defuzzification technique, which classifies an input data into the conclusion class of the rule with maximum fuzzy value in its condition part, i.e.:

$$class_k \Leftrightarrow k = \arg\max_{1 \le i \le m} \{eval(condition_i)\}$$

4. Proposed Approach

We evolve a rule for a specific class with one run of the genetic algorithm. In this way, if *m* is the number of different classes, we run the GA *m* times. Only the condition part has to be codified as a linear chromosome, with variable length. This can be represented using a binary tree (expression tree), where the leaf nodes are atomic expressions, and the intermediate nodes the logic operators (and, or).

4.1. Linear tree representation of complete-tree fuzzy rules

In our approach, we only evolved complete-tree fuzzy rules **(CTree)**, i.e., fuzzy rules with condition expressions that have an *h*-complete binary expression tree, for some natural number *h*. To establish the process of linear representation of *h*-complete fuzzy rules, we used the following grammar (in Backus Normal Form) for a free parenthesis logical expression:

<EXP> → <EXP><OPER><ATOMIC> | <ATOMIC>

<ATOMIC> → variable is [not] set

<OPER> → or | and

Applying repeatedly the previous definition, the following logical expression can be obtained:

A or B and C and D or E

where *A*, *B*, *C*, *D*, and *E* are atomic expressions. There are several processes to assign a meaning to a free parenthesis logical expression, i.e., to build an expression tree for the expression. We defined a process to build an *h*-complete tree from a given free parenthesis logical expression that exploits the advantages of the expression trees: leaf nodes are atomic expressions, intermediate nodes are logical operators, and if there are *n* operators then there are *n+1* atomic expressions. The proposed expression tree building process is defined recursively as:

$$Tree(Exp) = \begin{cases} < NIL, A_1, NIL > & if\ Exp = A_1 \\ replace(Tree(Exp - \{A_{n+1}, O_n\}), O_n, A_{n+1}) & otherwise \end{cases}$$

where *replace(t, o , a)*, replaces the first leaf node in the tree *t* (*first_leaf(t)*), according to the level enumeration process, by the node (*first_leaf(t), o, a*). A_{n+1}, is the last atomic expression in the argument expression *Exp*, and, O_n, is the last operator in the argument expression *Exp*. Figure 2, shows the execution of the proposed process for the sample free parenthesis logic expression *A or B and C and D or E*.

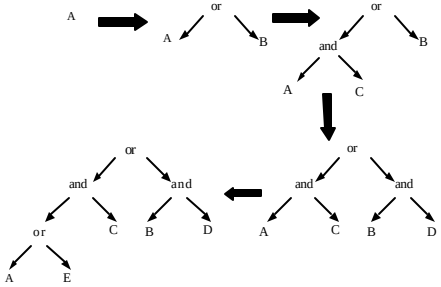


Figure 2: Execution of the proposed tree building process

Therefore, the free parenthesis logical expression:

A or B and C and D or E

Represents the logical expression:

(((A or E) and C) or (B and D))

The time that is expended in the evaluation of a codified condition is linear *O(n)*, because this process can be performed using a circular queue. We codified a logic expression without parenthesis of *n* logic operators in a chromosome of *n+1* genes, where the *i*-th gene is composed by the atomic expression A_i and the logic operator O_i. In this way the last gene has an unused logic operator. Figure 3 shows the chromosome for an expression tree of *n* logic operators.

Gen$_1$				...	Gen$_n$				Gen$_{n+1}$			
ac$_1$			op$_1$	...	ac$_n$			op$_n$	ac$_{n+1}$			*
var$_1$	ro$_1$	set$_1$		...	var$_n$	ro$_n$	set$_n$		var$_{n+1}$	ro$_{n+1}$	set$_{n+1}$	*

Figure 3: Chromosome for a complete rule condition

[1] The proof uses recursive duplication of trees that do not have brothers, and the commutative property of logic operators.

The number of bits used to codify the variable and set part of each atomic expression depends on the number of attributes and the number of defined fuzzy sets. In our approach, the number of bits is approximated by $log_2(n)$ for the variable part (n is the number of attributes), and by $log_2(m)$ for the fuzzy set part (m is the number of defined fuzzy sets). The relation operator part is codifed with only *one* bit as the logic operator is either *and* or *or*.

4.2. Genetic operators

One of the main problems with the linear representation of fuzzy rules is that the genetic operators produce global structure modifications over the expression tree codified. A genetic operator has the local structure modification property if the child expression tree has "similar" structure to the parent expression tree, i.e., the genetic operator preserves the tree structure. With the proposed linear representation, the genetic operators shown here have the local modification property.

4.2.1. Variable length crossover. When the crossover operator is applied, a crossover point is chosen between 1 and the minimum of the lengths of the two parent chromosomes. The code is exchanged as usual and the child with the highest fitness is chosen to replace the chromosome *p*. Figure 4, shows the effect of the crossover operation. Only one node (atomic or operator) is modified, other nodes are exchanged or kept. Because the crossover point was selected inside nodes *C* and *Y*, then these nodes interchange their code in the normal way (as in binary strings) and create the new fuzzy expressions *H* and *M*, the other nodes do not change. Also, the children preserve the tree structure of the parents.

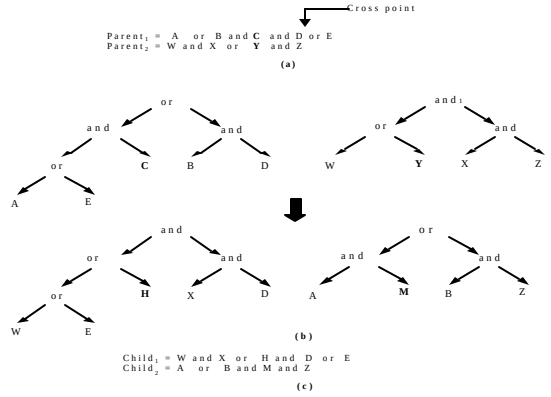


Figure 4: Effect of the crossover operator. (a) Parent genotypes, (b) Phenotypes code interchange, (c) Children genotypes.

4.2.2. Gene addition and deletion. The gene *addition* operator adds a random gene to the chromosome end. The gene *deletion* operator eliminates the last gene in the chromosome. Figure 5 shows the effect of the gene addition and deletion operators. It is clear that these operators preserve the tree structure.

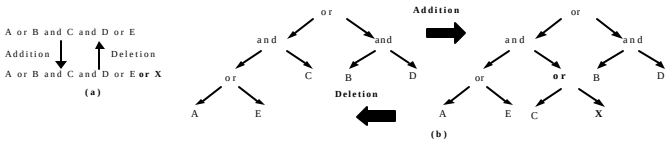


Figure 5: Effect of the addition and deletion operators. (a) Genotype modification, (b) Phenotype modification

In these operators, only one node is modified and the children preserve the tree structure of the parents.

4.2.3. Mutation. A randomly chosen bit is changed as used in simple GAs. Figure 6 shows the local modification effect of the mutation operator. This operator only modifies one node.

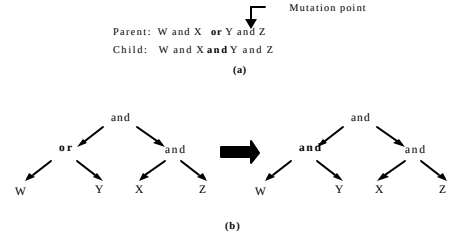


Figure 6: Effect of the mutation operator. (a) Genotype modification. (b) Phenotype modification.

4.3. Fitness evaluation

We opt to seek the best classification rule for each class separately because this leads to a much *faster* and *simpler* search, has the potential to yield *simpler* rules, and also because this yields an approach that can be easily *parallelized* on several independent processors, especially in the presence of many classes. Because we run the genetic algorithm once for each class and we want compressible rules, the optimization problem is a three-goal objective function: maximize the *sensitivity*, maximize the *specificity*, and minimize the *rule length*. In this paper we used the fuzzy confusion matrix to calculate the fitness of a chromosome (rule) instead of the confidence threshold with classical confusion matrix. In the fuzzy confusion matrix, the condition fuzzy truth degree and the fuzzy negation operator are used directly. Although there are different ways to deal with multi-objective optimization problems (Fonseca, 97), we chose the weighted sum technique. The fitness of a chromosome is evaluated according to the following equations:

$$TP = \sum_{i=1}^{p} \min\{real(data_i), predicted(data_i)\}$$

$$TN = \sum_{i=1}^{p} \min\{1 - real(data_i), 1 - predicted\ (data_i)\}$$

$$FP = \sum_{i=1}^{p} \min\{1 - real(data_i), predicted\ (data_i)\}$$

$$FN = \sum_{i=1}^{p} \min\{real(data_i), 1 - predicted(data_i)\}$$

$$sensitivity = \frac{TP}{TP + FN}, \qquad specificity = \frac{TN}{TN + FP}$$

$$length = 1 - \frac{chrom\_length}{10}$$

$$fitness = w_1 * sensitivity + w_2 * specificity + w_3 * length$$

Here,

- *TP*, *TN*, *FP*, and *FN* are the true positive, true negative, false positive, and false negative values for the codified rule respectively
- *real* is a function that returns 1.0 when the data sample belongs to the training class and 0.0 in other case,
- *predicted* is a function that returns the confidence value of the condition part of the codified rule
- *p* is the number of samples in the training data set, and,
- w_1, w_2, and w_3 are the assigned weight for each rule characteristic respectively.

5. Experimentation

In order to evaluate the performance of the proposed approach, tests were conducted using the IRIS, WINE, and VOTE data sets [12], and 10% of the KDDCup'99 network intrusion data [13]. The data sets are described in Table 1

.

Table 1 Testing data sets

Data Set	Size	Classes	Attributes
IRIS	150	3	4
WINE	178	3	13
VOTE	435	2	16
KDDCup'99	492021	23	41

In order to perform a complete comparison between different codification schemes in other approaches and the proposed approach, we defined a set of codifications schemes that cover many representations proposed in the literature:

- Conjunctive Normal Form: Order 1 (CNF-1), Order 2 (CNF-2), and Order 3 (CNF-3)
- Disjunctive Normal Form: Order 1 (DNF-1), Order 2 (DNF-2), and Order 3 (DNF-3)
- Operator Precedence (OpPr)

CNF-1 is the codification scheme proposed in [1], [3], [4], [5], [6], and [7], DNF-1 is the dual version of the CNF-1. CNF-2, CNF-3, DFN-2, and DNF-3 are a representative set of codification schemes proposed in [8]. The last strategy was proposed in [9]. The first six codification schemes were extended in order to be able to apply the addition and deletion operators. The extension is straightforward: Suppose we are codifying in a chromosome a CNF-*k* fuzzy rule, and the chromosome has *m* atomic expressions: A_0, A_1, ..,A_{m-1}. Then the codified rules in CNF-*k* is:

$(A_0 \vee A_2 \vee .. \vee A_{k-1}) \wedge (A_k \vee A_{k+1} \vee .. \vee A_{2k-1}) \wedge \ldots \wedge (A_p \vee A_{p+1} \vee .. A_{m-1})$

Where $p = k * \lfloor m/k \rfloor$, ($\lfloor . \rfloor$ indicates the *integer part*). We only included in the fuzzy classifier the best rule evolved for a given class to have a common point of comparison.

5.1. Data preprocessing

For each data set, ten fold experiments were employed [14]. The accuracy of the trained classifier was calculated as the average of these ten tests. This process was repeated five times for each data set and the average score was computed.

5.2. Experimental settings

A genetic algorithm with competitive genetic operators was employed to evolve the fuzzy classifier [15]. The algorithm uses a hybrid co-evolutionary learning rule strategy that updates the operator probabilities in a competitive way according to the applied operator and its effect over the chromosome fitness. The selection process uses the neighborhood concept to choose the parents. We used this type of genetic algorithm in order to eliminate the dependency of the results on the choice of the operator probabilities. The following parameters were used: Population size: 200, Maximum number of iterations: 200, Selection strategy: Tournament size 4, Initial chromosome length: Between 1 and 6 (randomly). We used five linguistic values for each (normalized) attribute (Low, ML, Med, MH, High), for the IRIS, WINE, and KDDCup'99 data sets. In the case of the VOTE data set, we used three crisp sets: YES, NO, and MARK, and for the KDDCup'99 categorical values we used the categorical values a crisp sets as explained in [16]. We tested three strategies to define the value of the fitness weights. The first strategy uses the constant values $w_1=0.45$, $w_2=0.45$, and $w_3=0.1$, as reported in [9]. The second strategy assigns proportional random values to the weight in order to give more importance to the specificity and sensitivity terms, i.e., the weight w_1, w_2, and w_3 are assigned random values such that $w_1, w_2 \geq 4*w_3$. The third strategy assigns random values to each weight. In strategies 2 and 3, the weights are scaled such that the weights sum to one.

5.3 Results and Analysis

Table 3 shows the properties of tested codification schemes.

Table 3 Induced properties of the fuzzy rules linear codification

CODIFICATION	REPRESENTABILITY	DISRUPTION
CNF-k, DNF-k	Low	Low
MOpPr	High	High
CTree	Medium	Low

It is clear that with the MOpPr scheme, it is possible to represent a big number of different fuzzy rule trees (the limitation depends on the maximum priority that can be assigned to the operators). The CNF-*k* and DNF-*k* schemes are very restricted and these cannot represent all the expression trees. The proposed approach **CTree** is more restricted than the MOpPr scheme, but is less restrictive than the CNF-*k* and DNF-*k* schemes. Besides, the disruption induced by the genetic operators over the tree structure is higher in the MOpPr scheme than in the others. For example, the mutation genetic operator can modify the priority of one operator and then change the tree structure of the fuzzy rule codified significantly. For the CNF-k, DNF-k, and CTree, the disruption induced by the genetic operators is small.

5.3.1. IRIS, WINE, and VOTE data sets. The average accuracy % reached for the genetic algorithm using the tested codification schemes for the IRIS, WINE, and VOTE data sets are shown in tables 4, 5, and 6 respectively[2] (next page). We

[2] The ESIA row in table 4 shows the reported performance of the strategy proposed in [3]. Because the average length in that paper is not shown, we estimate it as the number of fuzzy rules generated (6.4) over the number of classes (3) (in this way we supposed that each rule has a length of one, which is maximally optimistic). The OpPr row shows the results obtained in the paper [9], i.e., using a priority operator strategy, a genetic algorithm with fixed genetic operator probabilities, and a deletion operator that can delete *any* gene, not only the last. MOpPr uses the

ranked the codification schemes according to average performance over three tests. Our results fared well with the ones reported in the literature.

In the case of the WINE data set, the best result was obtained using constant fitness weight and predefined pattern DNF-3, i.e., when the chromosome codifies fuzzy rules of the form: *(A Ù B Ù C) Ú (D Ù E Ù F)*. Although the best performance was reached for DNF-3 this scheme was ranked 5th, because its performance depends on the fitness weight assignment strategy: when these were proportionate randomly assigned or totally random assigned, the performance decreased rapidly. The codification scheme CNF-1 was ranked 1st, because its average performance was better. According to these results, the WINE data set can be well modeled by a conjunction of atomic terms, while the worst codification schemes were the other predefined codification schemes. Therefore, the performance of using predefined patterns depends strongly on the high level knowledge used to generate them. The proposed approach was ranked in second place, indicating that it performs well without high level knowledge, and better than most of the tested codification schemes.

Table 4: Performance of the proposed approach for the IRIS data set

		TEST			RULE
COD	**RANK**	1	2	3	**LENGTH**
CTree	**4 (94.84)**	**93.73**	**95.33**	**95.46**	**1.78**
MOpPr	6 (94.39)	92.53	95.06	95.60	1.78
CNF-1	5 (94.52)	92.26	95.46	95.86	1.74
CNF-2	2 (95.24)	93.86	95.86	96.00	1.82
CNF-3	1 (95.46)	94.93	96.00	95.46	1.77
DNF-1	3 (95.19)	95.20	94.66	95.73	1.76
DNF-2	8 (93.95)	92.80	93.46	95.60	1.72
DNF-3	7 (93.99)	91.86	94.26	95.86	1.77
OpPr	94.5	-	-	-	2.10
ESIA	95.33	-	-	-	2.13

Table 5: Performance of the proposed approach for the WINE data set

		TEST			RULE
COD	**RANK**	**1**	**2**	**3**	**LENGTH**
CTree	**2 (92.22)**	**94.11**	**93.33**	**89.22**	**2.43**
MOpPr	3 (91.81)	94.11	92.88	88.44	2.39
CNF-1	1 (92.59)	94.44	93.11	90.22	2.19
CNF-2	6 (89.70)	91.44	90.22	87.44	2.51
CNF-3	7 (87.92)	88.55	88.22	87.00	1.92
DNF-1	8 (87.14)	87.11	87.00	87.33	1.46
DNF-2	4 (90.69)	91.88	90.22	90.00	1.76
DNF-3	5 (90.55)	94.55	88.22	88.88	2.01
OpPr	93.9	-	-	-	7.63
HGBML	92.7	-	-	-	1.70

same codification of OpPr but relies on the proposed genetic operators instead. The HGBML row shows the performance of the proposed approach in [7]. The results for this approach were taken from the table 2, when the number of generated rules was 3 and the reached performance with three rules was bigger.

Table 6: Performance of the proposed approach for the VOTE data set

		TEST			RULE
COD	**RANK**	**1**	**2**	**3**	**LENGTH**
CTree	**5 (95.42)**	**95.59**	**95.09**	**95.59**	**1.00**
OP	7 (95.40)	95.59	95.04	95.59	1.02
CNF-1	1 (95.48)	95.59	95.27	95.59	1.02
CNF-2	1 (95.48)	95.59	95.40	95.45	1.02
CNF-3	6 (95.41)	95.59	95.13	95.54	1.02
DNF-1	3 (95.46)	95.59	95.22	95.59	1.02
DNF-2	8 (95.37)	95.59	94.95	95.59	1.02
DNF-3	4 (95.43)	95.59	95.13	95.59	1.02
OpPr	94.7	-	-	-	1.1

In general, the performance of the genetic algorithm is best when the fitness weights are assigned as in [9], and worst when these are assigned randomly. In this case the performance of the evolved fuzzy classifier is greatly affected by the fitness weight assignment strategy. The performed test showed that the genetic algorithm is able to evolve simple fuzzy rules for these data sets. The evolution of the average fuzzy rule length for the WINE data set is shown in figure 8.

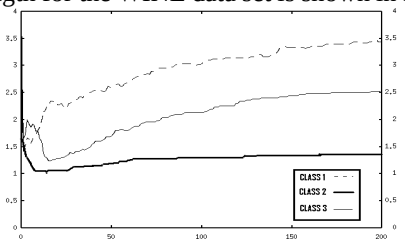


Figure 8. Evolution of the rule length for the WINE data set

5.3.2. KDDCup'99 data set. In this large data set (492021 samples), 42 attributes characterize network traffic behavior [13], and 22 different types of attacks than can be classified in four main intrusion classes, are shown in table 7. The classification accuracy of the fuzzy classifier evolved using **CTree** is shown in table 8 as well as those of the winner group in the KDDCup'99 contest [17]. The results are not directly comparable because they are obtained from different data sets, and are shown for reference. Table 9 shows that our **CTree**

approach fares well with other intrusion detection methods. A complete description of the performance of **CTree** on this task can be found in [16].

Table 7: Classes in the 10 % of the KDDCup 99 data set

CLASS	SUB-CLASSES	SAMPLES
NORMAL		95278 (19.3%)
U2R	buffer_overflow, loadmodule, multihop, perl, rootkit	59 (0.01%)
R2L	ftp_write, guess_passwd, imap, phf, spy, warezclient, warezmaster	1119 (0.23%)
DOS	back, land, Neptune, pod, smurf, teardrop	391458(79.5%)
PRB	Ipsweep, nmap, portsweep, satan	4107 (0.83%)

Table 8: Accuracy of CTree and winner in the KDDCup' 99 contest

CLASS	CTree	WINNER ENTRY
Normal	92.78%	94.50%
U2R	88.13%	13.2%
R2L	7.41%	8.4%
DOS	98.91%	97.10%
PRB	50.35%	83.30%

Table 9: Comparison between the proposed approach and others

Algorithm	FA %	DR %	Complexity
EFRID	**7.0**	**98.15**	**O(n)**
RIPPER-Artificial Anomalies [19]	2.02	94.26	$O(n*\log^2 n)$
SMARTSIFTER [20]	-	82.0	$O(n^2)$

6. Conclusions

Our experiments showed that the proposed representation approach was able to evolve fuzzy classifiers with accuracy comparable to the ones reported in the literature, and that the fitness weight assignment strategy affects fuzzy classifier evolution. Moreover, the proposed approach was able to evolve *simpler variable-length* fuzzy rules. Simpler fuzzy rules have a clear advantage in real applications. First they yield rules that are easier to interpret, hence scoring high on *interpretability*. Second they yield a classifier that is *faster* in deployment. This is especially *crucial* for data involving a *large number of attributes*. Our approach is practical regardless of the type of feature used, i.e. numerical or categorical, and even attributes resulting from *vague human linguistic* expressions, by virtue of *fuzzy sets*. It is therefore promising for a variety of data mining applications, particularly in Web [18,19] and Text mining. For such applications, the number of features can rise to explosive proportions resulting in rule lengths that can explode exponentially, and thus make not only the *training and search* process tedious, but also the *actual real-time classification* extremely slow. In online settings such as on the World Wide Web, both efficiency and time complexity are crucial. We plan to investigate the viability of the proposed techniques in implementing fast online Recommendation engines [18,19], and possibly search engines. The importance of fast response is even more crucial in *network security* applications: In automatic Intrusion Detection systems, a *fast* classifier can mean all the difference between being able to *detect and stop* the intruder's behavior *in time*, as compared to detecting the user's behavior, but responding *too late* because of a *slow* detection process. We emphasize that training is the most time consuming part of our system. Classification, on the other hand, is extremely *fast (not only because of the structure and linear codification of the expression tree, but also because of the simpler rules generated!)*, and is a viable method to other techniques used in machine learning, such as decision trees. One might wonder *whether fast* classification is more important than *accurate* classification. In our work, we do not claim this to be true. In fact, our use of *different weights* to weigh different criteria in accordance to their *importance depending* on the particular application and problem, is the best answer. For example, sensitivity is much more crucial than specificity in Intrusion Detection Systems. Thus we may prefer to choose the weights accordingly, hence resulting in simpler (fast to evaluate) rules with high sensitivity for example. In the case of providing online Web recommendations, simplicity is almost as important as accuracy because accurate recommendations are useless if they require a long time to generate, since the user will most likely have navigated to a different Web site, by the time recommendations are ready.

Acknowledgments

This work was supported by the Defense Advanced Research Projects Agency (F30602-00-2-0514) and National Sciences Foundation (NSF-EIA-9818323). Nasraoui is supported by an NSF CAREER Award. (NSF-IIS-0133948).

7. References

[1] K. De Jong and W. Spears (1991). Learning Concept Classification Rules Using Genetic Algorithms. *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pp. 651-656.

[2] C.E. Bojarczuk, H.S. Lopes and A.A. Freitas (1999). Discovering comprehensible classification rules using genetic programming: a case study in a medical domain. *Proc. Genetic and Evolutionary Computation Conference GECCO99*, Morgan Kaufmann, 1999, pp. 953-958.

[3] J. Liu and J. Kwok (2000). An extended genetic rule induction algorithm. *Proceedings of the Congress on Evolutionary Computation (CEC)*, pp.458-463.

[4] M.V. Fidelis, H.S. Lopes and A.A. Freitas (2000). Discovering comprehensible classification rules with a genetic algorithm. *Proc. Congress on Evolutionary Computation (CEC)*, pp. 805-810.

[5] A. Gonzalez and R. Prez (1998). Completeness and consistency conditions for learning fuzzy rules. *Fuzzy Sets and Systems*, 96: 37-51.

[6] H. Ishibuchi, K. Nozaki, N. Yamamoto and H. Tanaka (1995). Selecting fuzzy if-then rules for classification problems using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, **3**(3):260-270.

[7] H. Ishibuchi and T. Nakashima (2000). Linguistic Rule Extraction by Genetics-Based Machine Learning. *Proceedings*

of the Genetic and Evolutionary Computation Conference GECCO'00, 195-202. Morgan Kaufmann.
[8] A. Giordana and L. Saitta (1993). Regal: an integrated system for learning relations using genetic algorithms. *Proceedings of the Second International Workshop on Multistrategy Learning*. R.S. Michalski et G. Tecuci (eds), pp. 234-249.
[9] D. Dasgupta and F. Gonzalez (2001). Evolving Complex Fuzzy Classifier Rules Using a Linear Tree Genetic Algorithm. *Proceedings of the Genetic and Evolutionary Computation Conference, GECOO 2001*, pp. 299-305.
[10] T. Cormen, C. Leiserson and R Rivest (1990). Introduction to algorithms. Mac Graw Hill.
[11] Zadeh, L.A., "Fuzzy sets" in Information and Control, 8: 338-352, 1965
[12] C.L. Blake, and Merz, C.J. (1998). UCI Repository of machine learning databases Irvine, CA: University of California, Department of Information and Computer Science. http://www.ics.uci.edu/~mlearn/ MLRepository.html
[13] KDD-cup data set. http://kdd.ics.uci.edu/databases/ kddcup99/kddcup99.html
[14] T. Lim and W. Loh (1997). A Comparison of Prediction, Accuracy, Complexity, and Training Time of Thirty-Three Old and New Classification Algorithms. Technical Report, Department of Statistics, University of Wisconsin-Madison, No. 979.
[15] J. Gomez, and D. Dasgupta, "Using Competitive Operators and a Local Selection Scheme in Genetic Search", Submitted as Late-breaking paper to the Evolutionary Computation Conference GECCO02, 2002
[16] J. Gomez, and D. Dasgupta, "Evolving Fuzzy Rules for Intrusion", To appear in the proceedings of the Third Annual IEEE Information Assurance Workshop 2002 Conference. June 2002
[17] Results of the KDD' 99 Classifier learning contest, http://www-cse.ucsd.edu/users/elkan/clresults.html.
[18] Nasraoui O., Krishnapuram R., Joshi A., and Kamdar T. "Automatic Web User Profiling and Personalization using Robust Fuzzy Relational Clustering," in "E-Commerce and Intelligent Methods" in the series "Studies in Fuzziness and Soft Computing", J. Kacprzyk, Ed, Springer-Verlag, 2002.
[19] Nasraoui O. and Krishnapuram R., "A New Evolutionary Approach to Web Usage and Context Sensitive Associations Mining," International Journal on Computational Intelligence and Applications - Special Issue on Internet Intelligent Systems, in press (2002).