Approximate Versus Linguistic Representation in Fuzzy-UCS

Albert Orriols-Puig[1], Jorge Casillas[2], and Ester Bernadó-Mansilla[1]

[1] Grup de Recerca en Sistemes Intel·ligents, Enginyeria i Arquitectura La Salle, Universitat Ramon Llull, 08022 Barcelona (Spain)
{aorriols,esterb}@salle.url.edu
[2] Dept. Computer Science and Artificial Intelligence University of Granada, 18071, Granada (Spain)
casillas@ugr.es

Abstract. This paper introduces an approximate fuzzy representation to Fuzzy-UCS, a Michigan-style Learning Fuzzy-Classifier System that evolves linguistic fuzzy rules, and studies whether the flexibility provided by the approximate representation results in a significant improvement of the accuracy of the models evolved by the system. We test Fuzzy-UCS with both approximate and linguistic representation on a large collection of real-life problems and compare the results in terms of training and test accuracy and interpretability of the evolved rule sets.

Keywords: Genetic algorithms, learning classifier systems, genetic fuzzy systems, supervised learning.

1 Introduction

Fuzzy-UCS [1] is a *Michigan-style learning fuzzy-classifier system* that evolves a fuzzy rule set with descriptive or linguistic representation. One of the main novelties of Fuzzy-UCS with respect to other *genetic fuzzy systems* is that it evolves the rule set on-line from a stream of examples. Fuzzy-UCS represents the knowledge with linguistic fuzzy rules, which are highly interpretable since they share a common semantic. However, as this representation implies the discretization of the feature space, a single rule may not have the granularity required to define the class boundary of a given domain accurately. Thus, Fuzzy-UCS creates a set of overlapping fuzzy-rules around the decision boundaries which match examples of different classes, and the output depends on how the reasoning mechanism combines the knowledge of all these overlapping rules. Fuzzy-UCS proposed three inference schemes which led to a tradeoff between the amount of information used for the inference process and the size of the rule set.

To achieve better accuracy rates in fuzzy modeling, several authors have introduced the so-called *approximate* rule representation (also known as non-grid-oriented fuzzy systems, prototype-based representation, or fuzzy graphs), which proposes that the variables of fuzzy rules define their own fuzzy sets instead of representing linguistic

E. Corchado, A. Abraham, and W. Pedrycz (Eds.): HAIS 2008, LNAI 5271, pp. 722–729, 2008.

variables [2]. In this way, approximate fuzzy rules are semantic free, being able to tune the fuzzy sets of any variable of each rule independently. However, this also results in a degradation of interpretability of the rule set, since the fuzzy variables no longer share a unique linguistic interpretation. In this paper, we analyze whether the flexibility provided by the approximate representation allows the system to achieve higher performance and how this affects the interpretability of the evolved rule sets. That is, the approximate representation is more powerful than the linguistic one since it enables fuzzy systems to evolve independent fuzzy sets for each attribute that fit the class boundaries of each particular problem accurately. Nonetheless, the search space also increases since the semantics is evolved together with the fuzzy rules, posing more difficulties to the learner. Moreover, this flexibility could also result in overfitting the training instances in complex, noisy environments. Therefore, the aim of the present work is to study the frontier in the accuracy-interpretability tradeoff, clearly identifying the advantages and disadvantages—in terms of accuracy and readability of the rule sets—of having a more flexible knowledge representation in the field of on-line learning. For this purpose, we include the approximate representation to Fuzzy-UCS and adapt several mechanisms to deal with it. This new algorithm is addressed as Fuzzy-UCS with Approximate representation, i.e., Fuzzy-UCSa. We compare the behavior of Fuzzy-UCS and Fuzzy-UCSa in a large collection of real-life problems.

The remainder of this paper is organized as follows. Section 2 describes Fuzzy-UCSa focusing on the new fuzzy representation. Section 3 explains the analysis methodology and presents the results. Finally, Section 4 concludes the work.

2 The Approximate Fuzzy-UCS Classifier System

Fuzzy-UCSa is a system that extends Fuzzy-UCS [1] by introducing an approximate fuzzy representation. Fuzzy-UCSa works in two different models: exploration or training and exploitation or test. As follows, we describe the system focusing on the changes introduced with respect to Fuzzy-UCS. For further details, the reader is referred to [1].

2.1 Knowledge Representation

Fuzzy-UCS evolves a population [P] of classifiers which consist of a fuzzy rule and a set of parameters. The fuzzy rule follows the structure

$$\textbf{IF } x_1 \text{ is } FS^k_1 \text{ and } \ldots \text{ and } x_n \text{ is } FS^k_n \textbf{ THEN } c^k \textbf{ WITH } w^k, \quad (1)$$

where each input variable x_i is represented by a fuzzy set FS_i. In our experiments, we used triangular fuzzy sets; so, each FS_i is defined by the left vertex *a*, the middle vertex *b*, and the right vertex *c* of the triangle, i.e., $FS_i = (a, b, c)$. The consequent of the rule indicates the class c^k which the rule predicts. w^k is a weight ($0 \leq w^k \leq 1$) that denotes the soundness with which the rule predicts class c^k. The matching degree $\mu_A{}^k(e)$ of an example e with a classifier k is computed as the T-norm (we use the product) of the membership degree of each input attribute e_i with the corresponding fuzzy set FS_i. We enable the system to deal with missing values by considering that $\mu_A{}^k(e) = 1$ if e_i is not known.

Each classifier has four main parameters: 1) the fitness *F*, which estimates the accuracy of the rule; 2) the correct set size *cs*, which averages the sizes of the correct sets in which the classifier has participated (see Sect. 2.2); 3) the experience *exp*, which computes the contributions of the rule to classify the input instances; and 4) the numerosity *n*, which counts the number of copies of the rule in the population.

2.2 Learning Interaction

At each learning iteration, the system receives an input example *e* that belongs to class *c*. Then, it creates the match set [M] with all the classifiers in [P] that have a matching degree $\mu_A^k(e)$ greater than zero. Next, in exploration mode, the classifiers in [M] that advocate class *c* form the correct set [C]. In exploitation mode, the system returns the class of the rule that maximizes $F^k \cdot \mu_A^k(e)$ and no further action is taken. If none of the classifiers in [C] match *e* with $\mu_A^k(e) > 0.5$, the covering operator is triggered, which creates the classifier that maximally matches the input example. The covering operator creates an independent triangular-shape fuzzy set for each input variable with the following supports

$$(\text{rand}(\min_i - (\max_i\text{-}\min_i)/2, e_i), e_i , \text{rand}(e_i, \max_i+(\max_i\text{-}\min_i)/2)) , \quad (2)$$

where $\min_i$ and $\max_i$ are the minimum and maximum value that the attribute i can take, e_i is the attribute i of the example *e* for which covering has been fired, and rand generates a random number between both arguments. The parameters *F*, *n*, and *exp* of the new classifiers are set to 1. The new classifier is inserted into the population, deleting another one if there is not room for it.

2.3 Parameters Update

In the end of each learning iteration, Fuzzy-UCSa updates the parameters of the rules in [M]. First, the experience of the rule is incremented according to the current matching degree: $exp^k_{t+1} = exp^k_t + \mu_A^k(e)$. Next, the fitness is updated. For this purpose, each classifier internally maintains a vector of classes $\{c_1, ..., c_m\}$ and a vector of associated weights $\{v^k_1, \ldots, v^k_m\}$. Each weight v^k_j indicates the soundness with which rule *k* predicts class *j* for an example that fully matches this rule. The class c^k advocated by the rule is the class with the maximum weight v^k_j . Thus, given that the weights may change due to successive updates, the class that a rule predicts may also vary.

To update the weights, we first compute the sum of correct matchings cm^k for each class j: $cm^k_{jt+1} = cm^k_{jt}+m(k, j)$, where $m(k, j) = \mu_A^k(e)$ if the class predicted by the classifier equals the class of the input example and zero otherwise. Then, cm^k_{jt+1} is used to calculate the weights v^k_{jt+1}: $v^k_{jt+1} = cm^k_{jt+1}/exp^k_{t+1}$. Note that the sum of all the weights is 1.

The fitness is computed from the weights with the aim of favoring classifiers that match examples of a single class. We use $F^k_{t+1}=v^k\ max_{t+1}-\Sigma_{jlj\neq max}\ v^k_{jt+1}$, where we subtract the values of the other weights from the weight with maximum value v^k max. The fitness F^k is the value used as the weight w^k of the rule (see Equation 1). Next, the correct set size of all the classifiers in [C] is calculated as the arithmetic average of the sizes of all the correct sets in which the classifier has participated.

2.4 Discovery Component

Fuzzy-UCSa uses a steady-state *genetic algorithm* (GA) [3] to discover new promising rules. The GA is triggered in [C] when the average time since its last application upon the classifiers in [C] exceeds a certain threshold θ_{GA}. It selects two parents p_1 and p_2 from [C] using proportionate selection [3], where the probability of selecting a classifier k is proportional to $(F^k)^\nu \cdot \mu_A{}^k(e)$, in which $\nu>0$ is a constant that fixes the pressure toward maximally accurate rules (in our experiments, we set $\nu=10$). Rules with negative fitness are not considered for selection. The two parents are copied into offspring ch_1 and ch_2, which undergo crossover and mutation with probabilities χ and μ respectively. The crossover operator generates the fuzzy sets for each variable of the offspring as

$$b_{ch1} = b_{p1}\,\alpha + b_{p2}\,(1-\alpha) \;\text{ and }\; b_{ch2} = b_{p1}\,(1-\alpha) + b_{p2}\,\alpha \tag{3}$$

where $0 \le \alpha \le 1$ is a configuration parameter. As we wanted to generate offspring whose middle vertex b was close to the middle vertex of one of his parents, we set $\alpha=0.005$ in our experiments. Next, for both offspring, the procedure to cross the most-left and most-right vertices is the following. First, the two most-left and two most-right vertices are chosen

$$\min_{left} = \min(a_{p1}, a_{p2}, b_{ch}) \text{ and } mid_{left} = middle(a_{p1}, a_{p2}, b_{ch})\,, \tag{4}$$

$$mid_{right} = middle(c_{p1}, c_{p2}, b_{ch}) \text{ and } \max_{right} = \max(c_{p1}, c_{p2}, b_{ch})\,, \tag{5}$$

And then, these two values are used to generate the vertices *a* and *c*.

$$a_{ch} = rand(\min_{left}, mid_{left}) \text{ and } c_{ch} = rand(mid_{right}, \max_{right})\,, \tag{6}$$

where the functions *min*, *middle*, and *max* return respectively the minimum, the middle, and the maximum value among their arguments.

The mutation operator decides randomly if each vertex of a variable has to be mutated. The central vertex is mutated as follows:

$$b = rand(b - (b - a)\cdot m_0,\; b + (c - b)\cdot m_0)\,, \tag{7}$$

where m_0 ($0<m_0\le1$) defines the strength of the mutation. The left-most vertex is mutated as

$$a = rand\,(a - m_0(b-a)/2,\; a) \qquad \text{if } F>F_0 \;\&\; \text{no crossover}, \tag{8}$$

$$a = rand\,(a - m_0(b-a)/2,\; a + m_0(b-a)/2) \qquad \text{otherwise}. \tag{9}$$

And the right-most vertex

$$c = rand(c,\; c + m_0(c-b)/2) \qquad \text{if } F>F_0 \;\&\; \text{no crossover}, \tag{10}$$

$$c = rand(c - m_0(c-b)/2,\; c + m_0(c-b)/2) \qquad \text{otherwise}. \tag{11}$$

That is, if the rule is accurate enough ($F>F_0$) and has not been generated through crossover, mutation forces to generalize it. Otherwise, it can be either generalized or specified. In this way we increase the pressure toward maximum general and accurate rule sets.

The new offspring are introduced into the population. First, we check whether there exists a classifier in [C] that subsumes the new offspring. If it exists, the numerosity of the subsumer is increased. Otherwise, the new classifier is inserted into the population. We consider that a classifier *k1*, which is experienced ($exp^{k1} > \theta_{sub}$) and accurate enough ($F^{k1} > F_0$), can subsume another classifier k2 if for each variable i, $a^i_{k1} \leq a^i_{k2}$, $c^i_{k1} \geq c^i_{k2}$, and $b^i_{k1} - (b^i_{k1} - a^i_{k1})\delta \leq b^i_{k2} \leq b^i_{k1} + (c^i_{k1} - b^i_{k1})\delta$, where δ is a discount parameter (in our experiments we set δ=0.001). Thus, a rule condition subsumes another if the supports of the subsumed rule are enclosed in the supports of the subsumer rule and the middle vertices of their triangular-shaped fuzzy sets are close in the feature space.

If the population is full, excess classifiers are deleted from [P] with probability proportional to their correct set size estimate cs_k and their fitness F^k [1].

3 Experiments

In this section, we analyze if the approximate representation a) permits to fit the training instances more accurately, b) whether this improvement is also present in the prediction of previously unseen instances, and c) the impact on the interpretability of the evolved rule set. As follows we explain the methodology and present the obtained results.

3.1 Methodology

We compare Fuzzy-UCSa as defined in the previous section with Fuzzy-UCS with the three types of reasoning schemes defined in [1], that is, weighted average (wavg), in which all matching rules emit a vote for the class they predict; action winner (awin), in which the class of the rule that maximizes $Fk \cdot \mu_A{}^k(e)$ is chosen as output; and most numerous and fit rules (nfit), in which only the most numerous and fit rules are kept in the final population and all the matching rules emit a vote for the class they predict. Moreover, we also included C4.5 in the comparison to analyze how Fuzzy-UCS performs with respect to one of the most influential learners. We employed the same collection of twenty real-life problems used in [1] for the analysis.

We used the accuracy, i.e., the proportion of correct predictions, and the number of rules in the population to compare the performance and interpretability of the different approaches. To obtain reliable estimates of these metrics, we employed a ten-fold cross validation procedure. The results were statistically analyzed following the recommendations pointed out in [4]. We applied the multiple-comparison test of Friedman to contrast the null hypothesis that all the learning algorithms performed equivalently on average. If Friedman's test rejected the null hypothesis, we used the non-parametric Nemenyi test to compare all learners with each other. We complemented the statistical analysis by comparing the performance of each pair of learners by means of the non-parametric Wilcoxon signed-ranks test. For further information about the statistical tests see [4].

We configured both systems as (see [1] for notation details): N=6,400, F_0 = 0.99, ν = 10, $\{\theta_{GA}, \theta_{del}, \theta_{sub}\}$ = 50, χ = 0.8, μ = 0.04, δ=0.1 and P # = 0.6. Moreover, for Fuzzy-UCS, we set the number of linguistic terms to 5.

3.2 Results

Our first concern was to compare the precision in fitting the training instances of Fuzzy-UCSa with respect to Fuzzy-UCS and show how both systems perform with respect to C4.5. Thus, we computed the training accuracy obtained with the five approaches. Table 1 summarizes the average rank of each algorithm (the detailed results are not included due to space limitations). As a case study, Fig. 1(a) shows the domain of one of the tested problems, tao, Figs. 1(b) and 1(c) plot the decision boundaries learned by Fuzzy-UCS awin with 5 and 15 linguistic terms per variable—the grid in the two figures indicates the partitions in the feature space made by the cross-points of the triangular membership functions associated to the different fuzzy sets—, and Fig. 1(d) shows the decision boundaries learned by Fuzzy-UCSa. The results clearly show that the flexibility provided by the approximate representation enabled Fuzzy-UCSa to fit the training instances more accurately.

The multi-comparison test permitted to reject the null hypothesis that all the learners were equally accurate at α=0.001. The post-hoc Nemenyi test, at α=0.1, indicated that Fuzzy-UCSa achieved significantly better training performance than Fuzzy-UCS with any inference type and equivalent results to C4.5. Moreover, Fuzzy-UCS awin significantly degraded the training performance achieved with Fuzzy-UCS nfit. The pairwise comparisons by means of the non-parametric Wilcoxon signed-ranks test at α=0.05 confirmed the conclusions extracted by the Nemenyi test.

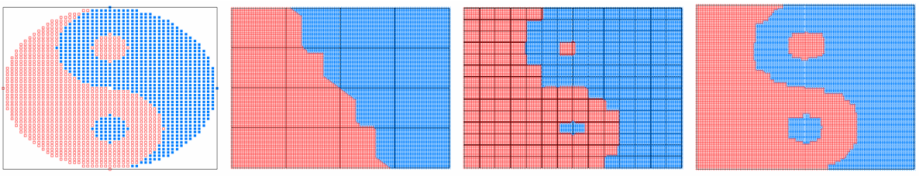

Fig. 1. Tao domain (a) and decision boundaries obtained by Fuzzy-UCS awin with 5 (b) and 15 (c) linguistic terms per variable and Fuzzy-UCSa (d). The training accuracy achieved in each case is 83.24%, 94.74%, and 96.94% respectively.

As expected, the approximate representation enabled Fuzzy-UCSa to fit the training examples more accurately; since there was no semantic shared among all variables, each variable could define its own fuzzy sets. Next, we analyzed if this improvement was also present in the test performance. Table 1 shows the average rank of test performance. The multi-comparison test rejected the hypothesis that all the learners performed the same on average at α = 0.001. The Nemenyi procedure, at α=0.1, identified two groups of techniques that performed equivalently. The first group included Fuzzy-UCS wavg, Fuzzy-UCSa, and C4.5. The second group comprised Fuzzy-UCSa, Fuzzy-UCS awin, and Fuzzy-UCS nfit. The same significant differences were found by the pairwise comparisons.

Further analysis pointed out that Fuzzy-UCSa was overfitting the training data in some of the domains. To contrast this hypothesis, we monitored the evolution of the training and test performance of the problems in which Fuzzy-UCSa degraded the

Table 1. Comparison of the average rank of train accuracy, test accuracy, and rule set size of linguistic Fuzzy-UCS with weighted average (wavg), action winner (awin), and most numerous and fitted rules inference (nfit), and Fuzzy-UCSa on a set of twenty real-world problems. The training and test accuracy of C4.5 is also included.

		Fuzzy-UCS			Fuzzy-UCSa	C4.5
train	*Rank*	3.35	4.20	3.10	1.75	2.60
	Pos	*4*	*5*	*3*	*1*	*2*
test	*Rank*	2.05	3.25	3.80	2.95	2.95
	Pos	*1*	*4*	*5*	*2.5*	*2.5*
size	*Rank*	3.95	2.05	1.00	3.00	-
	Pos	*4*	*2*	*1*	*3*	-

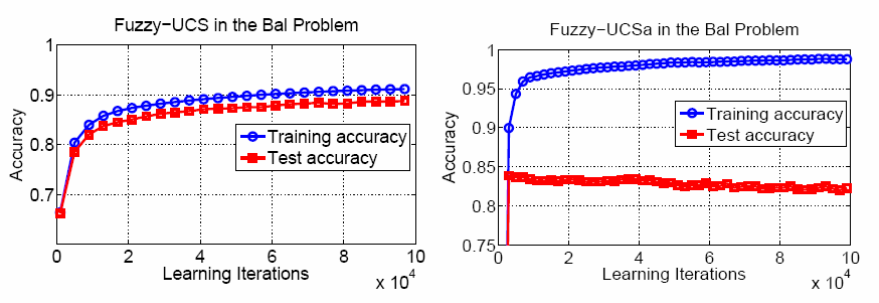

Fig. 2. Evolution of the training and test accuracies obtained with Fuzzy-UCS wavg and Fuzzy-UCSa in the *bal* problem

results obtained by Fuzzy-UCS with any inference type. Figure 2 plots the evolution of the training and test performance in the *bal* problem for Fuzzy-UCS wavg and Fuzzy-UCSa. During the first 5,000 learning iterations, both training and test performances of Fuzzy-UCSa rapidly increased, achieving about 90% and 84% accuracy rate respectively. After that, the training performance continued increasing while the test performance slightly decreased. After 100,000 iterations, the training performance reached 98%; nonetheless, the test performance decreased to 82%. Thus, at a certain point of the learning, the flexibility of the approximate representation led Fuzzy-UCSa to overfit the training instances in order to create more accurate classifiers, which went in detriment of the test performance. On the other hand, the training and test performance of Fuzzy-UCS wavg continuously increased, showing no signs of overfiting.

Finally, Table 1 also shows the average rank of the number of rules evolved for Fuzzy-UCS and Fuzzy-UCSa. The Friedman test rejected the hypothesis that the population sizes were equivalent on average at α=0.001. The post-hoc Nemenyi test supported the hypothesis that the four learners evolved populations with significantly different sizes. Fuzzy-UCS wavg created the biggest populations, closely followed by Fuzzy-UCSa. Nonetheless, Fuzzy-UCSa uses an approximate representation, in which

rules do not share the same semantic, thus, impairing the readability of the rule sets. Fuzzy-UCS awin and, specially, Fuzzy-UCS nfit resulted in the smallest populations.

4 Conclusions

This paper analyzed the advantages and disadvantages provided by the flexibility of the approximate representation in detail. We showed that the approximate representation enabled Fuzzy-UCSa to fit the training data more accurately. Nonetheless, there was no statistical evidence of this improvement in the test performance and it was identified that Fuzzy-UCSa may overfit the training instances in complex domains; furthermore, the approximate representation degraded the readability of the final rule sets. Therefore, the analysis served to identify that the flexibility provided by the approximate representation does not produce any relevant improvement to Fuzzy-UCS, strengthening the use of a linguistic, more readable representation.

Acknowledgments

The authors thank the support of *Ministerio de Educación y Ciencia* under projects TIN2005-08386-C05-01 and TIN2005-08386-C05-04 and *Generalitat de Catalunya* under grants 2005FI-00252 and 2005SGR-00302.

References

1. Orriols-Puig, A., Casillas, J., Bernadó-Mansilla, E.: Fuzzy-UCS: a Michigan-style learning fuzzy-classifier system for supervised learning. IEEE Transactions on Evolutionary Computation (in press)
2. Alcalá, R., Casillas, J., Cordón, O., Herrera, F.: Building fuzzy graphs: features and taxonomy of learning for non-grid-oriented fuzzy rule-based systems. Journal of Intelligent and Fuzzy Systems 11(3-4), 99–119 (2001)
3. Goldberg, D.E.: Genetic algorithms in search, optimization & machine learning, 1st edn. Addison-Wesley, Reading (1989)
4. Demsar, J.: Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research 7, 1–30 (2006)