FuAGGE: A Novel System to Automatically Generate Fuzzy Rule Based Learners

Romaissaa Mazouni and Abdellatif Rahmoun
Djilali Liabes University, Computer Science Departement, Algeria
E-mail: rom.mazouni@yahoo.com

Eric Hervet
Moncton University, Computer Science Departement, NB, Canada

Keywords: FuAGGE, grammatical evolution, fuzzy set covering, fuzzy rule-based learners, data mining

Received: January 6, 2016

Data in real world applications are in most cases linguistic information that are ambiguous and uncertain. Hence, such data should be handled by fuzzy set representation schemes to increase expressiveness and comprehensiveness. Moreover, mining these data requires ways to generate automatically useful information/knowledge through a set of fuzzy rules. This paper proposes a novel system called FuAGGE that stands for Fuzzy Automatic Generator Genetic Expression. The FuAGGE approach uses a grammar based evolutionary technique. The grammar is expressed in the Backus Naur Form (BNF) and represents a fuzzy set covering method. The grammar is mapped into programs that are themselves implementations of fuzzy rule-based learners. Binary strings are used as inputs to the mapper along with the BNF grammar. These binary strings represent possible potential solutions resulting from the initializer component and the building blocks from Weka, a workbench that contains a collection of visualization tools and algorithms for data analysis and predictive modeling. This operation facilitates the induction process and makes induced programs shorter. FuAGGE has been tested on a benchmark of well-known datasets and experimental results prove the efficiency of the proposed method. It is shown through comparison that our method outperforms most recent and similar, manual techniques. The system is able to generate rule-based learners specialized to specific domains, for example medical or biological data. The generated learners will be able to produces efficient rule models. The produced rule models will achieves more accurate classification for the specific used domain.

Povzetek: Razvit in testiran je algoritem FuAGGE za učenje mehkih pravil, ki omogoča učenje s slovnico, prilagojeno vsaki problemski domeni.

1 Introduction

Fuzzy systems have gained popularity due to their ability to express ambiguous and uncertain information in various real-world applications [1, 2, 3]. Hence, in order to take advantage of the well-established foundations of predicate logic-based expert systems, researchers focus on extracting fuzzy knowledge from available numerical data [4]. Yet, in [5], the authors propose a fuzzy extension of a rule learner called FILSMR, where they apply fuzzy set concepts [6] to the algorithm PRISM [7]. Later on, Wang and al proposed an algorithm called Fuzzy-AQR [8], where they introduced a seed that represents the highest membership to the positive set. It is used to generate an initial rule which should cover the seed. In [9], Van Zyl and al propose FuzzConRi. It is mainly based on the CN2 method [10]. FuzzConRi is composed of two layers: The upper layer uses a set covering approach, while the lower one is used to induce a single rule. Later on, in [11, 12], the same previous authors propose the Fuzzy-BEXA framework. This framework consists of three layers: The top layer implements a set covering approach, the middle layer uses heuristics to guide the search, and the lower layer is dedicated to refining conjunctions. Huhn and al introduced a new rule learner called FURIA [13]. This is a fuzzy extension of the RIPPER algorithm [14], except the fact that it learns fuzzy unordered rule sets instead of conventional rule lists. It also uses a rule stretching method to solve the problem of uncovered records. In 2014 Swathi et al [15] used fuzzy classification entropy to generate fuzzy decision tree (FDT) and later the parameters of FDT are tuned to further increase the accuracy of FDT. Nevertheless, all of existing methods in the current literature rely on designing fuzzy rule learners manually .In this work we present a grammar evolution based system that automatically generates such rules. We believe that automating the process of writing fuzzy rule based classifiers can highly improve the efficiency of such classifiers. Yet, an automatic generation of fuzzy rule based classifiers alleviates the burden of writing long source codes. The proposed system relies on a grammar that represents the overall structure for fuzzy set covering approaches.

The paper is composed of six sections in addition to the above introduction: Section 2 describes the Grammatical Evolution method. Section 3 describes the fuzzy rule-based classifiers and their features. Section 4 illustrates the automatic generation of fuzzy rule learners. Section 5 offers a description of how the system automatically generates rule-based learners. Section 6 presents the results obtained with the proposed system, and finally section 7 concludes the paper.

2 Grammatical evolution

Grammatical Evolution is a special case of grammar-based genetic programming that uses a mapping procedure between the genotypes and the phenotypes of individuals [16]. Grammatical evolution can generate complete programs in an arbitrary programming language using populations of variable-length binary strings. These computer programs are solutions to a given problem [17, 18]. When using Grammatical Evolution to generate solutions to a given problem, there is no need to pay attention to the genetic operators and how they are implemented: Grammatical Evolution ensures the validity of the generated programs for free. As described in [19], Grammatical Evolution applies concepts from molecular biology to the representational power of formal grammars [20]. The genotype-phenotype mapping in Grammatical Evolution has the advantage, over solution trees used in traditional Genetic Programming, to allow operators to act on genotypes. By analogy to biological DNA, a string of integers that represents a genotype is called a chromosome, and the integers that compose a chromosome are called codons. A Backus Naur Form grammar definition must be introduced prior to using Grammatical Evolution to solve a given problem. This grammar describes the output language produced by the system in [21, 22], and is used along with the chromosomes in the mapping process, which consists of mapping non-terminals to terminals, and completed through converting the binary string data structure into a string of integers, which is finally passed from the genetic algorithm on to the Grammatical Evolution system. The string of integers then goes through a translation process, where rules of the BNF grammar are selected. The production rules, which can be considered equivalent to amino acids in genetics, are combined to produce terminals, which are the components making up the final program. One problem that can occur when mapping binary strings is the production of short genes, meaning we run out of genes, but there are still some non-terminals to map. A solution to this issue is to wrap out the individuals, and to reuse the genes. A gene can be used several times in the mapping process. It is also possible to declare some individuals as invalid by penalizing them with a suitable harsh fitness value. The rules selection is performed by using the mod ulo operator, and every time a codon (an integer) is read, it is divided by the number of the rule's choices. The remainder of this division is the number of the rule to be selected. Grammatical Evolution can be used to automatically generate fuzzy classifiers by using a grammar that represents the overall structure of these fuzzy classifiers. The initial population which is a group of individuals (integer arrays) is used along with the grammar in the mapping process. The GE Mapper produces phenotypes (fuzzy classifiers) which are evaluated using the fitness function. Phenotypes go through an evolution process in the Search Engine until a stopping criterion is met and a best fit fuzzy classifier is found. The different modules of the whole Grammatical Evolution approach are illustrated in Fig. 1.

3 Fuzzy rule induction

One of the main and most studied tasks of data mining is classification, which aims at predicting to which class belongs a certain element according to any given classification model. The classification model can be a set of decision rules extracted, using a given dataset, and which represents local patterns of a model in this dataset. Decision rules can be extracted from other knowledge representations such as Decision Trees [23]. Moreover, they can be drawn out directly from the training set, or may also be induced by using evolutionary algorithms, more specifically, genetic algorithms or genetic programming. Fuzzy set theory lead researchers to look for fuzzy alternatives for data mining problems such as fuzzy induction learners, fuzzy decision trees, and fuzzy clustering. The present work focuses on the sequential covering rule induction paradigm. The fuzzy version of the sequential covering paradigm is called fuzzy set covering. The proposed idea is to train one rule at a time, remove the examples it covers, and repeat the process until all data is fully covered, as described in [24]. There are plenty of proposed algorithms that follow the fuzzy set covering paradigm. These algorithms usually differ in some components such as the search mechanisms [25]. There are three different search strategies: The bottom-up one starts with a random sample from the dataset, then generalizes it; The top-down strategy starts with an empty rule, then specializes it by adding preconditions to it; Finally the bidirectional strategy, which is the least common one, allows to either generalize or specialize the candidates. There are also two different categories of search methods. The most used ones are the greedy method (ex: FILSMR [7]) and the beam method (ex: FuzzConRi [9], FuzzyBEXA [12]). Covering algorithms have different ways to evaluate rules. Some of the existing methods are: The fuzzy entropy (ex: fuzzy ID3 [26]), the Laplace estimation (ex: Fuzzy- BEXA [12], FuzzConRi [9]), the Fuzzy Bexa framework (fuzzy purity, the fuzzy ls-content and fuzzy accuracy function), and the fuzzy info gain (ex: FILSMR [6], Fuzzy-AQR [8]). The final component that differentiates covering algorithms is the pruning method, which consists in handling over fitting and noisy data. There are two types of pruning: pre-pruning that deals

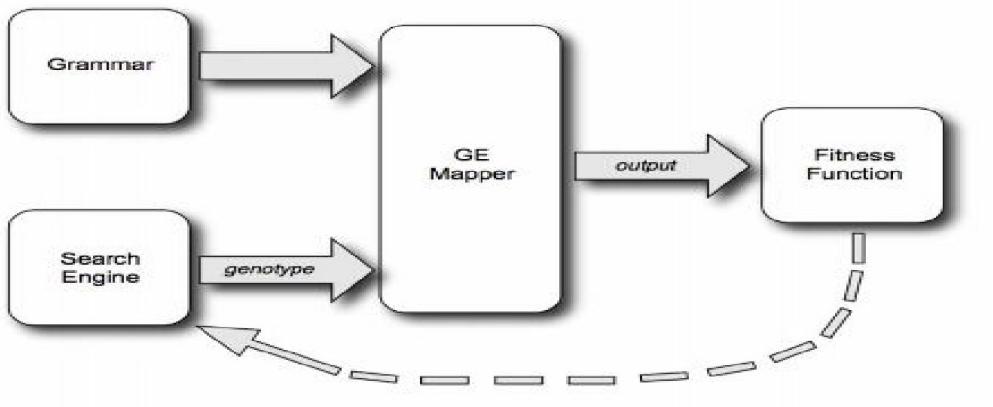

Figure 1: Grammatical Evolution Modules

with over fitting and noisy data, and post-pruning that deals with rejection and conflicts in order to find a complete consistent rule set. Pre-pruning offers the ability to obtain a high-speed model, while post-pruning helps getting simpler and more accurate models. The grammar we use is a context-free grammar in Backus Naur form that contains all elements necessary to build a basic fuzzy rule learner, following the fuzzy set covering paradigm. It contains 19 production rules, each one representing non-terminal symbols, and 40 terminal symbols describing the elements. The grammar produces fuzzy rule set without any specific order needed when applying them. This process provides different initialization, refinement, and evaluation techniques.

4 Automatic generation of fuzzy rule learners

To build a system that is able to generate fuzzy rule-based learners, we used a context-free grammar that represents the whole structure of the sequential fuzzy set covering paradigm (Fig. 2). This grammar was built after reviewing different fuzzy rule set inducers in the existing literature. It contains 19 rules and 40 terminals, where each terminal stands for a building block performing an action. Algorithm 1 illustrates the building block represented by the terminal Include1Selector in the rule number 11 in the grammar. It is worth mentioning that this method is the first one quoted in the literature that automatically generates a fuzzy rule induction algorithm. To do this, we put in place a grammar evolution scheme. The decision rules model has been selected on the basis of intuitiveness and comprehensiveness. In the following section, we propose a system that combines Grammatical Evolution with a context-free grammar, in order to generate code fragments possessing the ability to generate accurate, noise-tolerant, and compact decision fuzzy rule sets.

Algorithm 5 `Include1Selector` (rule R).

1: refinements = $\emptyset$
2: **for** i = 0 to i < numberAtt **do**
3: **for** for j = 0 to j < numberVal(Att_i) **do**
4: newAntecedant = R U (Att_i,Val_j)
5: refinements = refinements U newAntecedant
6: **end for**
7: **end for**
8: return refinements

4.1 Proposed system: FuAGGE

The suggested method includes five main components. To start, all what we need is a grammar that represents the overall structure of all manually designed fuzzy rule learners that obeys to the fuzzy set covering technique. We use some building blocks from Weka, which is a workbench that contains a set of visualization tools and a set of algorithms for data analysis and predictive modeling. This step will help reading the dataset files, and testing the newly generated rule based learners. It might be seen as "code reusing". We also need some machine learning datasets in order to train and test the newly generated learners. The datasets we used were taken from the UCI machine learning repository [27]. Indeed, we need several and various datasets, so that when fuzzy rule learners (candidate solutions) are trained, these are not tailored to a specific domain. Finally, we have to take care of the mapper of the Grammatical Evolution [17], modified in such a way that when it reads terminals, pieces of Java code representing these terminals actions are generated. At this stage, we used the GEVA framework to implement the whole system [18]. The most important component of our system is the mapper, which must have the ability to read the integer values from the chromosomes / candidate solutions that are called in this paper the FuAGGE classifiers. Then, we had to choose the appropriate corresponding rule to

a given non-terminal, import some of the already coded Weka building blocks, and insert them along with the terminals corresponding to Java code into the Java class that builds the new FuAGGE classifier. Fig. 2 represents the whole system and its modules. Individuals in this system are represented as integer arrays that are to be mapped to the fuzzy rule learners. Every array is read one integer at a time [28], and every integer is divided by the number of choices of the current rule. The number of the rule to be chosen and applied next is the remainder of the division (Eq. 1), where N is the currently read integer, Nc is the number of choices of the next rule to apply, and Idrule is the identifier of the rule selected by the mapper. This rule will be mapped into Java code (Fig. 3).

$$N \bmod N_c = Id_{\text{rule}} \tag{1}$$

When using Grammatical Evolution to solve a given problem, we need a measure that favors the selection of the best individuals among a population of possible solutions. The metric, also called the fitness function, used in this work to evaluate the FuAGGE classifiers generated during the evolution process, is the accuracy method. After the initialization of the first population, individuals go through the mapping process and are integrated into Java programs (FuAGGE classifiers), which are the actual classifiers that need to be compiled and executed (trained and tested) using different datasets. Each fuzzy rule learner has a set of different accuracies accuracy per dataset). The average of all these accuracies is used as the classifiers overall accuracy, so we are able to compare the different FuAGGE classifiers over a population. The following equation defines the overall accuracy of a FuAGGE classifier in a given population. acci;j represents the accuracy of the FuAGGE classifier i using the dataset j, and h is the number of datasets:

$$f(i) = \frac{\sum_{j=0}^{h} acc_{ij}}{h} \tag{2}$$

When using Grammatical Evolution, we do not need to check the off-springs generated after a mutation or a crossover operation, because the genetic operators are applied on genotypes: Since these are represented by integer arrays, a crossover or a mutation operation over them generates the same type of arrays, which are then mapped using the grammar. This might not be always the case if we were using the Context Free Grammar Genetic Programming method, because the genetic operators are applied on the phenotypes (syntax trees), and this may generate unsuitable off-springs.

5 Experimental results and discussion

Three components are needed so that the system can start the evolution process: The grammar introduced in section. 4, the meta datasets, and finally the Grammatical Evolution parameters. The parameters have been set as follows: the number of generation has been set to 60, the population size to 150, the mutation probability to 1%, the crossover probability to 80%, and the selection method is the tournament selection with generational replacement. We should note here that all these parameters have been fixed empirically after that we analyzed a certain number of trials and experimentations.

In order to evaluate the newly generated fuzzy learners, we computed the accuracies of two manually designed fuzzy rule learners using all ten datasets. The last column of Table 1 reports accuracies of the new generated fuzzy learners (FuAGGE classifier), while the remaining columns shows the accuracies of the two manually designed classifiers (Furia, Fuzzy CN2). The upper six rows reports the accuracies of the fuzzy rules sets generated by the FuAGGE-classifier, and the two baseline ones show the accuracies using only the meta-training set (each row represents the test accuracy of a single set from the meta training set). These accuracies are reported here to show the success of the training phase, while the lower four rows of the table show the predictive accuracies of the FuAGGE-classifier for sets that were part neither of the training, nor of the validation phase.

After the grammar has been created, the data prepared, and the implementation and testing phases launched, the system proved its ability to produce rule learners that are competitive with manually designed learners. We can clearly see the performance of these formers in Table 1. We should clarify that these accuracies were calculated by averaging the accuracies of the fuzzy rule model generated by the FuAGGE-classifier for each test set over the 10 iterations of the 10-fold-cross-validation method used during experiments. This also applies to the rest of the benchmark rule based learners used for purpose of comparison. It is worth mentioning that in Table 1, the new generated fuzzy learners has almost the same results as the other methods, and if we compare only the baseline methods with each other's we can clearly notice that the Fuzzy CN2 records 5 wins over 1 for Furia even though Furia is more sophisticated: It uses a growing, pruning and optimization phase just like the crisp version RIPPER and a fuzzy rule stretching method. Now if we look at the FuAGGE-classifier accuracies, we can notice how close these accuracies are to the baseline algorithms accuracies, which is very interesting due to the fact that the FuAGGE-classifier is automatically generated, and this removes a great deal of human time coding tasks. Human designers can easily go wrong when parameterizing an algorithm during the design process, contrariwise the chance of having wrong parameters when using automatic evolution of algorithms is very low. The last four rows show that the FuAGGE- classifier records 1 win against the baseline classifiers (Puba), and an equality with Fuzzy CN2 (Haberman). For the Ion dataset, the FuAGGE results are very close to the best accuracy (90.38% versus 91.17%). These results prove that the pro-

```
1-<Start>              --><FuzzyRuleSet>  [<PostPrune>]
2-<Fuzzy RuleSet>      --> For-each-concept  <WhileLoop> endFor.
3-<WhileLoop>          -->While <Cond> <GenerateAnt> endWhile
4-<Cond>               -->(Pos  ≥  <α-cover> ) ≠Ø  | ((Pos  ≥  <α-cover> ) ≠Ø  && (newAntExist))
5-<GenerateAnt>        --><InitializeAnt> <SecWhile> [<PrePrune>]
6-<InitializeAnt>      -->True | bestSEED | randomSEED
7-<SecWhile>           -->while <SecCond> <FindAnt> endWhile
8-<SecCond>            -->SelectorSetNotempty |  SelectorSetNotempty && SelectorNeg  <α-cover>
9-<FindAnt>            --><RefineAnt> <EvaluateAnt>  <SelectAnt> |<RefineAnt> <Bayes> |
                          if numCovExp ( >1 <)(90%| 95%| 99%) then <RefineAnt> else <RefineAnt> |
                          if AntSizeSmaller (2| 3| 5| 7) then <RefineAnt> else <RefineAnt>
10-<RefineAnt>         --><IncludeSelectors>|<ExcludeSelectors>
11-<IncludeSelectors>  -->Include1selector | Include 2selectors
12-<ExludeSelectors>   -->Exlude1selector | Exlude2selectors
13-<EvaluateAnt>       -->LaplaceEstimate |  FuzzyLaplace|  Fuzzyls-content| FuzzyAccFunction|  FuzzyPurity|
                          FuuzyInfGain| FuzzyEntropy
14-<SelectAnt>         -->1beam |2beam| 3beam| 4beam |5beam
15-<α-cover>           -->0.5 | 0.6| 0.7| 0.8| 0.9
16-<Bayes>             --> if  ( bayesSelector  ≥ <β-belonging>) formRule
17-<β-belonging>       -->0.7| 0.8| 0.9| 0.95
18-<PrePrune>          -->(Exlude1selector | Exlude2selector) <EvaluateAnt>
19-<PostPrune>         -->ExludeRule  modelAcc | <ExludeSelectors> modelAcc
```

Figure 2: The Fuzzy Set Covering Grammar.

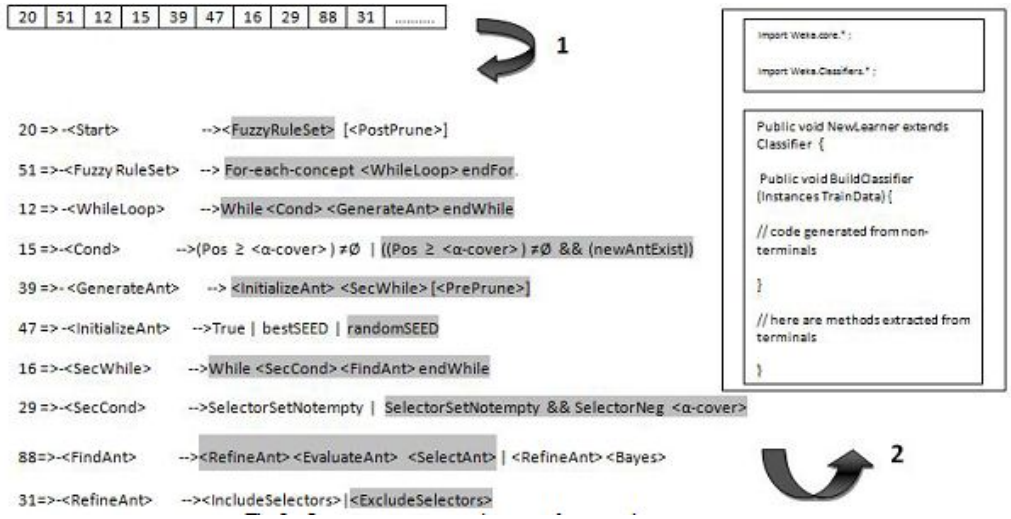

Figure 3: Integer Arrays to a Java Code Mapping Process.

posed approach is very promising. However, the FuAGGE system is time consuming while evolving FuAGGE classifiers and requires high computational power. Actually, if run on an ordinary computer, the evolution process can take up to one week of continuous calculation. We should also note that this version of the system does not handle missing values. The system eliminates instances with missing value before using the datasets. We should also note that this version uses only numeric attributes. And finally, the data has been fuzzified manually which is really time consuming . This should be tackeled in the next version of the system by giving it the ability to handle missing values either by replacing missing values with the mean or median of the current class, or by the most common attribute values. We could also give the system the ability to automatically fuzzify data, this can be done simply by coding the steps required to fuzzify the data, or by following a method proposed by Swathi et al. [29, 30] which converts numerical attributes into fuzzy membership functions using fuzzy clustering. Swathi et al. [30] presented two heuristic algorithms for the estimation of parameterized family of membership functions, namely, triangular and trapezoidal.

Table 1: Accuracy rates (%) using both meta-sets.

	Furia	Fuzzy-CN2	FuAGGE Classifier
Iris	92.06	95.20	95.13
Pima	75.65	79.10	60.20
Glass	69.63	65.90	70.43
Wine	65.82	97.90	59.81
Vehicle	70.57	73.30	73.58
Wbc	95.28	98.11	96.26
Ecoli	83.63	83.90	79.82
Ion	91.17	89.50	90.38
Puba	67.83	57.44	70.10
Haberman	72.55	72.92	72.91

Table 2 presents the accuracies of rule-based learners generated using our proposed system AGGE [31] in the first column, and those generated using the fuzzy exten-

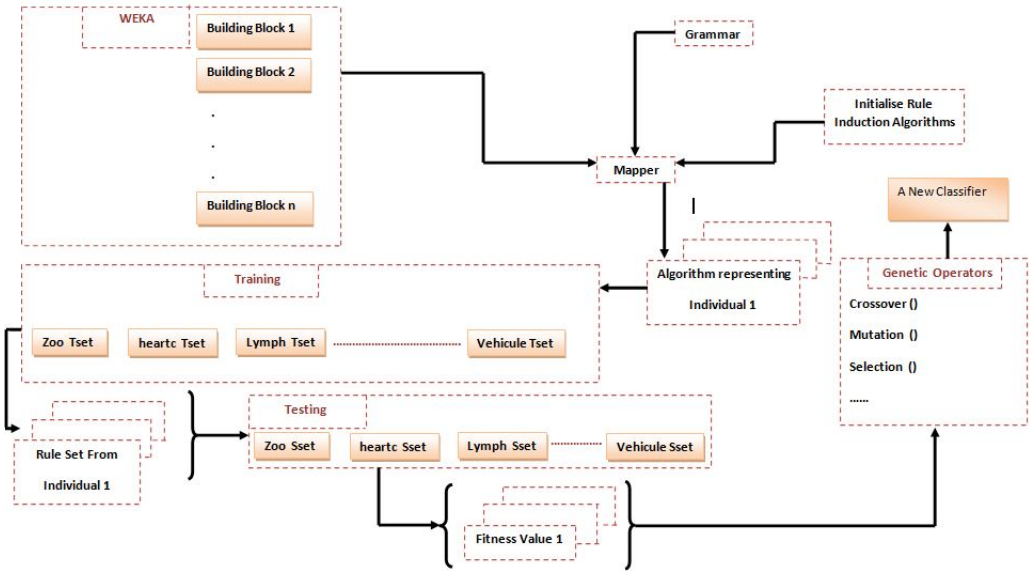

Figure 4: Modules of the proposed system.

AGGE Classifier	FuAGGE Classifier	
Iris	95.09	95.13
Ion	88.14	90.38
Pima	75.34	60.20
Wine	92.03	59.81
Vehicle	90.03	73.58
Glass	62.69	70.43
WBC	94.93	96.26
Ecoli	73.24	79.82
Haberman	67.45	72.81
Puba	76.44	70.10

sion presented in this paper FuAGGE in the second column. It is worth noting that the parameters were set as follows: generations=60, population size=150, crossover=0.8, mutation=0.1, for both AGGE systems. The upper 6 rows represent the accuracies using the meta training set, and the lower rows the accuracies using the meta testing sets. We notice that even though the grammar used for the FuAGGE system is very straightforward, the system's classifiers recorded 7 wins versus 3 wins for the crisp AGGE classifiers. This is due to the efficiency of fuzzy systems versus crisp ones in the classification domain. FuAGGE proves to be more interesting than AGGE in terms of the efficiency of its generated classifiers. However, if the number of generations or the mutation and crossover rates are too high, we might lose the efficiency of the system because of the destructive nature of its operators. Accordingly, setting AGGE and FuAGGE parameters is highly sensitive.

6 Conclusion

In this paper we present a new approach able to automatically produce fuzzy rule-based learners. The system is mainly based on a BNF context free grammar representing the overall structure of the baseline fuzzy rule learners. It also applies the concept of Grammatical Evolution to produce the best fit rule learner. Experiments on a commonly used data benchmark show that it is possible to automatically produce rule learners that can compete with manually designed ones, even with a basic grammar. This is of great importance, because the proposed method reduces the burden of writing manually thousands of lines of source code, and offers a better parameterizing of programs. Our method can be easily extended to a wide range of applications, provided that sufficient related data is available. This may open interesting opportunities and new trends in data mining and computational intelligence. For instance, our system could produce rule based learners dedicated to medical data, biological data, or physics and engineering data. Algorithms would be parameterized in a way to better fit a specific domain and as a result, would achieve more accurate results in this particular field. Another way of extending the system is to change the grammar to make the system capable to generate other data mining algorithms, such as clustering or fuzzy clustering algorithms.

References

[1] X. Wang, D. Nauck, M. Spott, R. Kruse. (2007) Intelligent data analysis with fuzzy decision trees, *Soft Computing*, Springer, pp. 439–457.

[2] H. Shen, J. Yang, S. Wang, X. Liu, (2006) Attribute weighted mercer kernel based fuzzy clustering algo-

rithm for general non-spherical datasets, *Soft Computing*, Springer, 10 (11) pp. 1061–1073.

[3] R. Ab Ghani, A. Salwani, Y. Razali (2015) Comparisons between artificial neural networks and fuzzy logic models in forecasting general examinations results, *2nd International Conference on Computer, Communications and Control Technology*, Malaysia, pp. 253–257.

[4] J. Zyl (2007) *Fuzzy set covering as a new paradigm for the induction of fuzzy classification rules*, University of Mannheim.

[5] C. Wang, J. Liu, T. Hong, S. Tseng (1997) FILSMR: a fuzzy inductive learning strategy for modular rules, *Proceedings of the Sixth IEEE International Conference on Fuzzy Systems*, pp. 1289–1294.

[6] L. A. Zadeh (1965) Fuzzy sets, *Information and Control*, pp. 333–353.

[7] D. Cendrowska (1987) PRISM: An Algorithm for Inducing Modular Rules, *In International Journal of Machine Learning Studies*, pp.349–370.

[8] C. Wang, C. Tsai, T. Hong, S. Tseng, (2003) Fuzzy Inductive Learning Strategies, *Applied Intelligence*, Publisher, 18 (2) pp. 179–19.

[9] J. Zyl, I. Cloete, (2004) FuzzConRI- A Fuzzy Conjunctive Rule Inducer, *Proceedings of the ECML/PKDD04 Workshop on Advances in Inductive Rule Learning*, pp. 194–203.

[10] P. Clark, R. Boswell,(1991) Rule induction with CN2: Some recent improvements, *Proceedings of the Sixth European Working Session on Learning*,pp. 151–163.

[11] H. Theron, I. Cloete, (1996) BEXA: A covering algorithm for learning propositional concept descriptions, *Machine Learning*, Publisher, pp. 5–40.

[12] J. Zyl, I. Cloete, (2005) Fuzzy rule induction in a set covering framework, *IEEE Trans. Fuzzy Syst*, pp. 93–110.

[13] J. Huhn, E. Hullermeier, (2009) FURIA: an algorithm for unordered fuzzy rule induction, *Data Mining and Knowledge Discovery*, pp. 293–319.

[14] J. Furnkranz, (1994) Pruning Methods for Rule Learning Algorithms, *Proceedings of the 4th International Workshop on Inductive Logic Programming*,USA, pp. 231–336.

[15] J. N. Swathi, B.B. Rajen, P. Ilango , M. Khalid , B. K. Tripathy, (2014) Induction of fuzzy decision trees and its refinement using gradient projected-neuro-fuzzy decision tree, *International Journal of Advanced Intelligence Paradigms*,pp. 346–369.

[16] J. Montana (1994) Strongly Typed Genetic Programming, *Evolutionary Computation Journal*, pp. 199–230.

[17] A. Nohejl, (2011) *Grammar Based Genetic Programming*, Charles University of Prague.

[18] H. Li, L. Wong (2014) Knowledge Discovering in Corporate Securities Fraud by Using Grammar Based Genetic Programming, *Journal of Computer and Communications* pp. 148–156.

[19] I. Dempsey, M. O'Neill, A. Brabazon (2009) *Foundations in Grammatical Evolution for Dynamic Environments*, Springer.

[20] M. O'Neill, E. Hemberg, C. Gilligan, E. Bartley, J. McDermott, A. Brabazon (2008) GEVA: Grammatical Evolution. in Java, *SIGEVOlution ACM*, (3) pp.17-23.

[21] A. De Silva, F. Noorian, R. Davis, P. Leong (2013) A Hybrid Feature Selection and Generation Algorithm for Electricity Load Prediction using Grammatical Evolution, *12th International Conference on Machine Learning and Applications*, Miami USA, pp. 211-217.

[22] P. Gisele, A. Freita, (2010) *Automating the Design of Data Mining Algorithms*, Springer Berlin Heidelberg.

[23] J. R. Quinlan, (1986) Induction of Decision Trees, *Machine Learning* , Kluwer Academic Publishers, pp. 81–106.

[24] A. Freitas,(2002) *Data mining and Knowledge Discovery with evolutionary algorithms*, Springer Verlag.

[25] M. Corn, M. A. Kunc, (2015) Designing model and control system using evolutionary algorithm, *8th Vienna International Conferenceon Mathematical Modelling — MATHMOD 2015*, Vienna, pp. 536–531.

[26] M. Dong, R. Kothari (2014) Classifiability Based Pruning of Decision Trees, *Neural Networks. Proceedings IJCNN '14, International Joint Conference on Neural Networks (IJCNN)*, Washington, pp. 1793–1947.

[27] UCI Machine Learning Repository, http://archive.ics.uci.edu/ml/

[28] A. Nohejl, (2009) *Grammatical Evolution*, Charles University of Prague.

[29] J. N. Swathi, I. Paramasivam, B. B. Rajen, M. Khalid (2015) A Study on the Approximation of Clustered Data to Parameterized Family of Fuzzy Membership Functions for the Induction of Fuzzy Decision Trees, *Cybernetics and Information Technologies*,pp. 75–96.

[30] B. B. Rajen, J. N. Swathi, P. Ilango, and M. Khalid,(2012) Approximating fuzzy membership functions from clustered raw data, *In Proceedings of India Conference (INDICON) Annual IEEE*, India, pp. 487–492.

[31] R. Mazouni, A. Rahmoun, (2015) AGGE: A Novel Method to Automatically Generate Rule Induction Classifiers Using Grammatical Evolution, *Studies in Computational Intelligence*, Springer, Madrid, pp. 270–288.