S-FRULER: Scalable Fuzzy Rule Learning through Evolution for Regression

I. Rodríguez-Fdez, M. Mucientes*, A. Bugarín

Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela, Santiago de Compostela, Spain

Abstract

In genetic fuzzy systems (GFS) the size of the problem has a huge influence in the performance of the obtained models, since i) the fuzzy rule bases learned suffer from exponential rule explosion when the number of variables increases, and ii) the convergence time increments with the number of examples. In this paper we present S-FRULER, a scalable distributed version of FRULER which is a GFS that learns simple and linguistic TSK-1 knowledge bases for regression problems. S-FRULER obtains models with high accuracy and low complexity, whilst reducing the algorithm runtime. S-FRULER focuses on splitting the problem into smaller partitions and incorporates a feature selection process for reducing the number of variables used in each partition. Each partition is then solved independently using the FRULER algorithm. Afterwards, an aggregation function obtains the final linguistic TSK fuzzy rule base from the information generated in each partition. S-FRULER has been validated in terms of scalability, precision and complexity using 10 large-scale datasets and has been compared with three state of the art GFSs. Experimental results show that S-FRULER scales well while achieving simple models with a linguistic approach and a precision comparable with approximative models. Moreover, S-FRULER has been applied to a bioinformatics problem that was proposed as a benchmark for scalability of regression problems, obtaining good results in both accuracy and complexity.

Keywords: Genetic Fuzzy Systems, regression, instances selection, multi-granularity fuzzy discretization

1. Introduction

The vast majority of techniques in statistical learning and data mining were traditionally designed to work with a limited amount of data [22]. The adaptation of these approaches to large scale datasets has been a huge challenge addressed in the last years [21]. Usually, all the problems that emerge with the use of big amounts of data are labelled under the term Big Data, which covers distributed processing and storing of data. Statistical learning algorithms in a Big Data context suffer from the problem of scalability, that is, the capability of the algorithm to maintain competitive performance as the size of the problem increases. Performance of the algorithms refers in this context to the computational cost, the precision of the obtained models, and their complexity. Also, the size of the problems can grow in two different dimensions: the number of examples available in training and the number of variables considered.

Recently, the use of Fuzzy Systems in the Big Data paradigm has attracted attention [16]. The first approach in this field was to scale the fuzzy c-means algorithm [23]. More recently, fuzzy modeling was applied to medical big data problems, using a neuro-fuzzy classifier for dimensionality reduction [5]. In [28], the authors propose an algorithm for Big Data binary imbalanced classification problems using the MapReduce scheme. Finally, in [14] it is introduced the first approach of a fuzzy rule based associative classifier based on the MapReduce approach.

Particularly, in a genetic fuzzy system (GFS) approach, the size of the problem has a huge influence in the performance of the obtained models [12, 24]. The fuzzy rule bases learned suffer from exponential rule explosion when the

*Corresponding author.

Email addresses: `ismael.rodriguez@usc.es` (I. Rodríguez-Fdez), `manuel.mucientes@usc.es` (M. Mucientes), `alberto.bugarin.diz@usc.es` (A. Bugarín) number of variables increases. Thus, with huge search spaces, the convergence time towards precise and simple models rises. Moreover, evolutionary algorithms are computationally expensive by themselves due to the large number of evaluations needed to reach convergence. Furthermore, in many cases, the evaluation process to obtain the fitness may take a long time. There are different techniques to improve the scalability of GFS that can be classified into three categories [17]: i) algorithm oriented that adapts the structure of the evolutionary algorithm, ii) data oriented that modifies the training data to reduce the computational cost of the learning process, and iii) distributed approaches that take advantage of the availability of several machines to reduce the runtime.

To scale the learning process of a GFS in an algorithm-oriented manner, several papers in the literature focused in the control of the search space, by reducing the number of rules and/or the number of labels used in the rule base through a multi-objective approach [1, 2, 15, 4]. Specifically, in linguistic approaches where the partition of each variable into fuzzy labels is defined equally for all rules, rule explosion can be controlled by limiting the number of labels considered in the learning process [1, 36, 37]. Moreover, in recent years, the data oriented approach has received increasing attention. The use of instance selection techniques decreases the complexity of large scale problems and reduces overfitting. In [1] was introduced an estimation error mechanism which selects randomly a subset of examples to estimate the real error, and the complete training dataset was only used for the most promising individuals. Also, in [35], a new instance selection method for regression was applied to generate the rules with the selected examples and the error of the rule base was estimated with the whole training dataset.

From a Big Data point of view, the distributed computing approach is the most appropriate for scaling GFS. Among the most frequently used frameworks in Big Data analytics [33], the most popular ones are: i) MPI (Message Passing Interface) which efficiently exploits multi-core clusters architectures, and ii) Apache Spark [44], a recently developed platform that can be executed in traditional clusters such as Hadoop [43]. Spark was designed to perform distributed processing and other workloads like streaming, interactive queries, and machine learning focused algorithms. While MPI provides a solution mostly oriented to high performance computing, Spark also deals with failures and straggler nodes effectively but with an impact on speed. From the perspective of GFS, only a few works use Big Data frameworks to solve the scaling problem [17].

The extended use of Spark is closely linked to the success of Hadoop, which processes vast amounts of data in parallel on large clusters, usually implemented using the Hadoop Distributed File System. Hadoop popularized the approach of MapReduce, a distributed methodology based on the definition of two different functions: Map and Reduce [13]. On one hand, a Map function distributes a block of data to several Worker Nodes, and executes the same process — called Task — in each data partition. On the other hand, the Reduce function aggregates the results of the Map functions by means of a Key-Value representation of the results and performs some operations to obtain the final result. Spark adds to this framework the capability to use other data-flows with an improvement of in-memory computing and an easy-of-programming high-level functions that facilitate to build parallel applications.

Solving large scale regression problems with GFSs can be found in some of the most recent works in the field [1, 2, 36, 37]. However, the number of variables and/or number of examples in the datasets used in these works are still not high enough to be properly considered labelled as Big Data. Among the different approaches, FRULER [37] obtains Takagi-Sugeno-Kang 1-order (TSK-1) fuzzy rule bases with high accuracy and the lowest number of rules. Although the runtime of this approach is acceptable (between 1-23 minutes) for the most simple datasets, it does not scale properly when solving large scale problems (from 1 to 30 hours). Moreover, with larger problems it may not converge to a good solution in reasonable time.

In this paper we propose a scalable version of FRULER, called S-FRULER, which allows to obtain models with similar characteristics than those obtained by FRULER —accurate and simple—, but reducing the runtime of the algorithm to converge. The main contributions of this work are: i) a novel distributed GFS approach, ii) a random feature selection process to reduce the number of variables used in each dataset partition, and iii) an aggregation function to obtain linguistic TSK fuzzy rule bases from the rule bases obtained in each dataset partition.

This paper is structured as follows: Section 2 defines the TSK model used in this work and its implications for a Big Data approach. Section 3 describes the different stages of S-FRULER. Section 4 shows the results of the approach in 10 regressions problems and the application of S-FRULER to a large bioinformatics problem. Finally, Section 5 presents the conclusions.

2. TSK Fuzzy Systems and Big Data

In this section the factors to be taken into account when learning TSK fuzzy rule bases in a Big Data environment are described. Takagi, Sugeno, and Kang proposed in [40, 39] a fuzzy rule model in which the antecedents are comprised of linguistic variables, as in the case of Mamdani [29, 30], but the consequent is represented as a polynomial function of the input variables. This type of rules is called TSK fuzzy rules. The most common function for the consequent of a TSK rule is a linear combination of the input variables (TSK-1), and its structure is as follows:

$$\begin{aligned}&\text{If } X_1 \text{ is } A_1 \text{ and } X_2 \text{ is } A_2 \text{ and} \ldots \text{ and } X_p \text{ is } A_p \text{ then}\\ &Y = \beta_0 + X_1 \cdot \beta_1 + X_2 \cdot \beta_2 + \cdots + X_p \cdot \beta_p\end{aligned} \tag{1}$$

where X_j represents the j-th input variable, p the number of input variables, A_j is the linguistic fuzzy term for X_j, Y is the output variable, and β_j is the coefficient associated with X_j in the consequent part of the rule.

The matching degree h between the antecedent of the rule r_k and the current inputs to the system $(x_1, x_2, \ldots, x_p)$ is calculated as:

$$h_k = T(A_1^k(x_1), A_2^k(x_2), \ldots, A_p^k(x_p)) \tag{2}$$

where A_j^k is the linguistic fuzzy term for the j-th input variable in the k-th rule and T is the t-norm conjunctive operator, usually the minimum function. The final output of a TSK fuzzy rule base system composed by m TSK fuzzy rules is computed as the average of the individual rule outputs Y_k weighted by the matching degree:

$$\hat{y} = \frac{\sum_{k=1}^{m} h_k \cdot Y_k}{\sum_{k=1}^{m} h_k} \tag{3}$$

In a regression analysis, the β coefficients are the relation that transforms the inputs into the desired output. The most fitted coefficients to the data can be found minimizing the least squares equation:

$$\hat{\beta} = \arg\min_{\beta} \|Y - X \cdot \beta\|_2^2 \tag{4}$$

where β is the coefficients vector $(\beta_0, \beta_1, \ldots, \beta_p)$, Y is the outputs vector $(y^1, \ldots, y^n)$, X is the inputs matrix with size $n \times p$ —rows represent examples while columns are the input variables. The coefficients associated with each rule consequent cannot be learned separately using Eq. 4 because the function that needs to be approximated is the aggregation of all rules (Eq. 3). Therefore, all the coefficients must be optimized at the same time, taking into account the degree of fulfillment of each rule (Eq. 2) for each input vector. Thus, the X matrix is modified as follows:

- The normalized degree of fulfillment for each rule r_k for each example e^i is calculated as:

$$z_k^i = \frac{h_k(x^i)}{\sum_{u=1}^{m} h_u(x^i)} \tag{5}$$

  where the denominator is the normalization term for each input vector x^i, i.e., the summation of the degree of fulfillment of all rules.

- Then, the X matrix is defined as:

$$X = \begin{pmatrix} z_1^1, & x_1^1 \cdot z_1^1, & \ldots, & x_p^1 \cdot z_1^1, & \ldots, & z_m^1, & x_1^1 \cdot z_m^1, & \ldots, & x_p^1 \cdot z_m^1 \\ \vdots & & & & & & & & \vdots \\ z_1^n, & x_1^n \cdot z_1^n, & \ldots, & x_p^n \cdot z_1^n, & \ldots, & z_m^n, & x_1^n \cdot z_m^n, & \ldots, & x_p^n \cdot z_m^n \end{pmatrix} \tag{6}$$

  where each row replicates the input vector $x^i = (1, x_1^i, x_2^i, \ldots, x_p^i)$ —where a 1 was added to take into account the independent term— as many times as the number of rules (m), weighting each rule r_k by z_k^i.

- Finally, the coefficient vector is the concatenation of the coefficients of all rules:

$$\beta = \begin{pmatrix}\beta_0^1, & \beta_1^1, & \ldots, & \beta_p^1, & \ldots, & \beta_0^m, & \beta_1^m, & \ldots, & \beta_p^m\end{pmatrix} \tag{7}$$

The computational cost for automatic learning of the coefficients of a TSK fuzzy rule base depends on the size of the X matrix (Eq. 6). The size of X not only depends on the number of examples (n) and the number of variables (p), but also on the number of rules (m). The number of rules increases exponentially with the number of variables, number of fuzzy labels per variable and the number of examples to be covered due to rule explosion. Thus, this process is the largest computational cost in a GFS that learns TSK Fuzzy rule bases and the most critical point to take into account for scalability. This can be partly solved through limiting the number of available labels — controlling the rule explosion and reducing the size of X —, using instance selection — reducing the number of examples n — or by feature selection — reducing the number of variables p.

Traditionally, an iterative Kalman filter was used to solve least squares to approximate the most fitted solution to the data [40]. However this approach has the drawbacks of overfitting and the high computational cost of the iterative Kalman filter. On one hand, the overfitting problem can be solved by shrinking (Ridge regularization) [34] or setting some coefficients to zero (Lasso regularization) to obtain simpler models. Moreover, a combination of both regularizations, called Elastic Net [45], can be used [37].

On the other hand, new scalable approaches for solving regression problems, regularized regression problems in particular, have been proposed in the recent years, such as coordinate descent [19] and stochastic gradient descent (SGD) [41, 10]. SGD is characterized by updating each coefficient separately using only one example at a time. This is particularly suited for sparse datasets, which is a common case when X is constructed using Eq. 6 —z_k^i is 0 when a rule does not cover an example. For example, in [37], a SGD approach was developed to solve Elastic-Net regularization to obtain the consequents of TSK-1 Fuzzy rule bases.

3. S-FRULER

This section presents S-FRULER (Scalable Fuzzy Rule Learning through Evolution for Regression), a distributed approach for applying FRULER [37] with scalability properties that allow its application to large scale problems. FRULER is a GFS that learns simple and linguistic TSK-1 knowledge bases for regression problems. It is composed of a two-stage preprocessing — formed by an instance selection and a multi-granularity fuzzy discretization—, and a genetic algorithm, which contains an ad-hoc TSK-1 rule generation module. Although the runtime of this approach is acceptable for medium size datasets, it does not scale properly when solving large scale problems as it may not converge. To solve that, S-FRULER divides the problem into a set of smaller problems that are more tractable using a distributed approach. Each of the divisions is then solved independently in the Map phase using FRULER. Then, the solutions obtained in each Map are combined in the Aggregation phase in order to obtain a final solution for the original data.

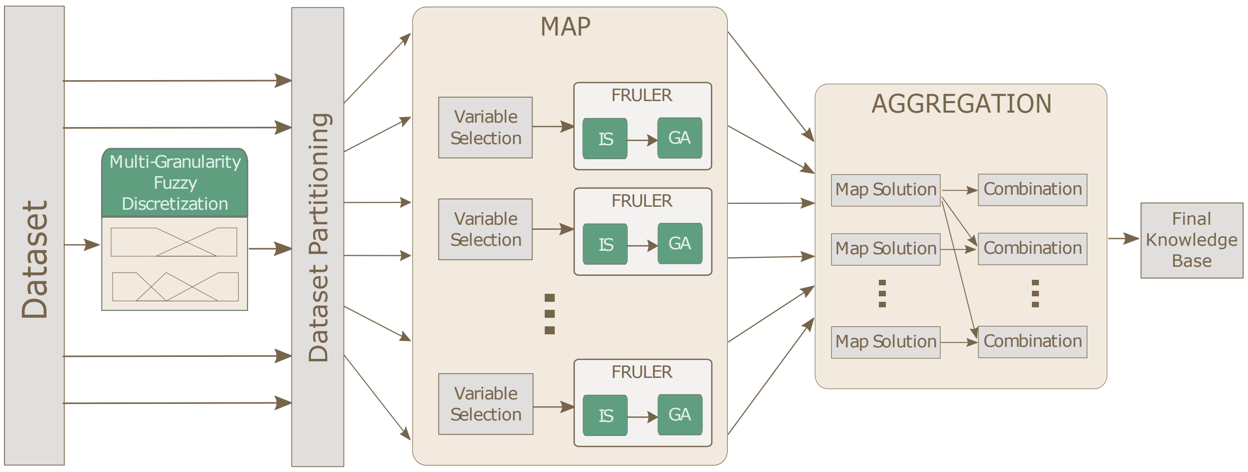

Figure 1: S-FRULER architecture showing the preprocessing, Map and Aggregation phases.

The algorithm structure is shown in figure 1. First, the multi-granularity fuzzy discretization process is performed using the whole training dataset. Then, the training dataset is splitted into n_{map} partitions, which correspond to the tasks to be distributed into the Working Nodes. Also, for each dataset partition, only a subset of randomly selected variables is taken into account. Each task is solved using FRULER —without the discretization phase—, as if it was an independent problem. Thus, only the instance selection (IS) and the genetic algorithm (GA) are executed. After obtaining the solutions for each task, these are combined completing the variables not used in one dataset partition with information of the others. The following subsections describe each of these phases in more detail.

3.1. Preprocessing before mapping

This stage comprises the multi-granularity fuzzy discretization of the input variables and the partition of the training dataset.

3.1.1. Multi-granularity Fuzzy Discretization

The first step of S-FRULER consists in the application of the Multi-granularity Fuzzy Discretization method used in FRULER [37] to discretize the input variables into fuzzy labels. This method obtains non-uniform fuzzy partitions with different degrees of granularity. A granularity g^i_{var} divides the variable *var* into *i* fuzzy labels, i.e., $g^i_{var} = \{A^{i,1}_{var}, \ldots, A^{i,i}_{var}\}$.

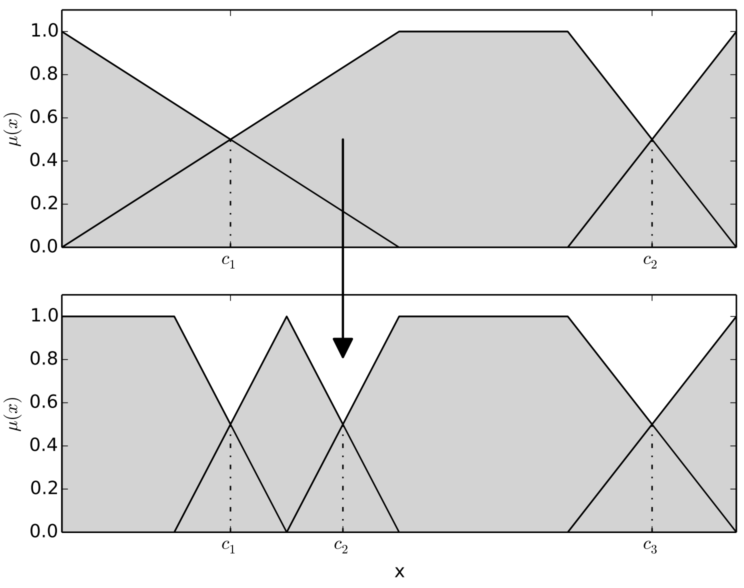

Figure 2: Top-down approach for the multi-granularity discretization. Only one label is divided into two new labels in order to obtain the next granularity.

The algorithm works as follows[1]:

- First, each variable is discretized to obtain a set of split points C^g for each granularity g.
  - The split points are searched iteratively, i.e., only a new split point is added at each granularity, starting from the most general granularity. Therefore, the approach proposed in this work aims to preserve interpretability between contiguous granularities: adding a new label to the previous granularity and modifying the adjacent labels (Fig. 2).
  - The split points that minimize the error when a linear model is applied to each of the resulting intervals are selected.

[1]For the sake of completeness, the stages of the algorithm are sketched in this section. A detailed description can be found in [37]

  - The method keeps adding split points until a bayesian information criterion (BIC) worsens. To calculate the BIC, the error is measured as the summation of the mean squared error of a least squares fitted model for each interval of the discretization, while the complexity is determined by the number of inner splits and the parameters fitted by each regression applied in each interval.

- Then, the method proposed in [26] is applied to each C^g —set of split points for the granularity g— in order to get the multi-granularity fuzzy partitions. This method uses a parameter that assesses the fuzziness of the linguistic labels. Fuzziness 0 indicates crisp intervals, while fuzziness 1 indicates the selection of a fuzzy set with the smallest kernel —set of points with membership equal to 1.

3.1.2. Dataset Partitioning

After the discretization of the input variables, the dataset is partitioned and distributed along the Workers Nodes of the computation cluster. The training set is divided randomly into n_{map} partitions with equal size $\frac{n}{n_{map}}$, where n is the total number of examples in the training data. In scalability terms, the higher the number of partitions n_{map} that can be used, the better. However, when the number of partitions surpasses a certain problem-dependent threshold, the obtained results worsen. This means that the problem has been divided too much, and each Map has not enough information to get a valid result. Thus, the automatic definition of the most adequate number of partitions for each problem can be useful when no information about the underlying characteristics of the problem is known. In S-FRULER, the number of partitions n_{map} is defined heuristically as:

$$n_{map} = log_2(p^2 * l * n) \tag{8}$$

where p is the number of input variables, n is the number of examples and l is the maximum granularity over all the input variables. Thus, n_{map} depends on the maximum size of X (Eq. 6) allowed in the initialization of FRULER, which is the critical part in scaling the algorithm (See Sec. 2).

3.2. Map function

The Map function is applied independently and distributively to each training dataset partition. For each training dataset partition, only a subset of selected variables is used. Thus, all the process done in each task uses the corresponding subset of examples and subset of variables, which is going to be referred as the dataset partition. For each task, FRULER is applied without the discretization step, since it was already performed before the partition of the data. It is composed by an instance selection of the most representative examples and a genetic algorithm, which contains an ad-hoc TSK-1 rule generation module (Fig. 3). The evolutionary learning process obtains a definition of the data base. Then an ad-hoc TSK-1 rule generation module obtains the antecedents and consequents of each possible rule using only the representative examples.

3.2.1. Variable Selection

To simplify the learning process in each task, only a subset of randomly selected variables is used for each dataset partition. The probability of selecting a particular input variable X_j in a dataset partition is:

$$P(X_j \in X_s^i) = \frac{p_m}{p}, \tag{9}$$

where X_s^i is the selected subset of input variables in the dataset partition i, p_m is the subset size of selected input variables and p is the total number of input variables. Thus, the probability that a particular input variable is not selected for all the dataset partitions is:

$$P(X_j \notin X_s^i, \forall i = 1, \ldots, n) = \left(\frac{p - p_m}{p}\right)^{n_{map}}. \tag{10}$$

where n_{map} is the total number of dataset partitions. Therefore, to define a subset size of selected input variables p_m that assures that a particular feature X_j is selected in at least one dataset partition with probability α_{p_m}, the following equation can be used:

$$p_m \geq -p \cdot ((1 - \alpha_{p_m})^{1/n_{map}} - 1) \tag{11}$$

Therefore, each task uses a dataset formed by $\frac{n}{n_{map}}$ examples from the original training dataset and p_m randomly selected input variables.

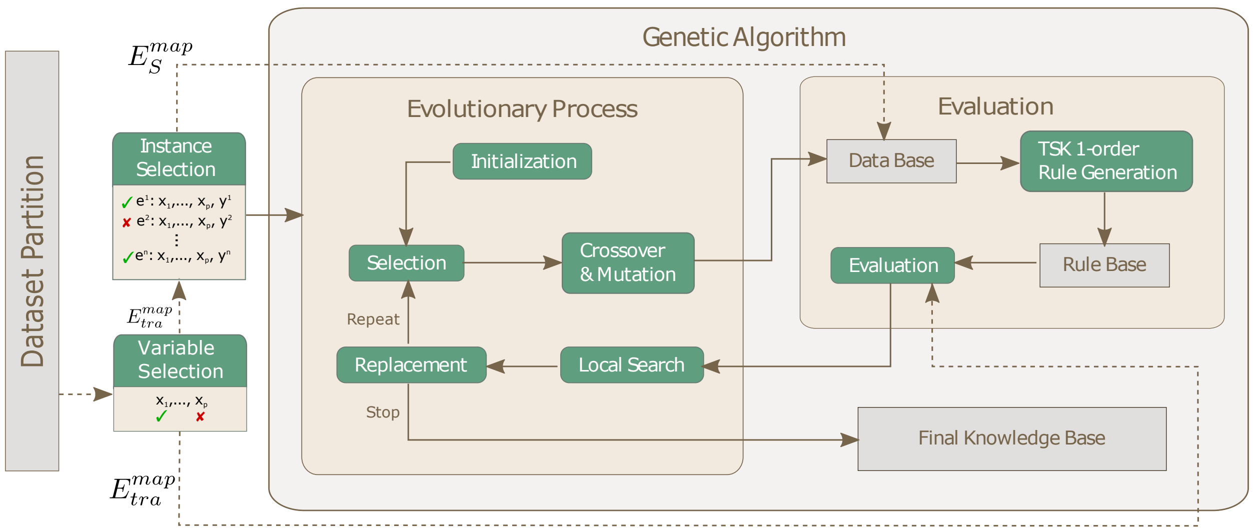

Figure 3: FRULER [37] architecture showing each of the two stages. Dashed lines indicate flow of data sets and solid lines represent process flow.

3.2.2. Instance Selection for Regression

The instance selection method for regression used in FRULER [37] is an improvement of the CCISR (Class Conditional Instance Selection for Regression) algorithm [35]. The method is based on a relation called class conditional nearest neighbor which is used to obtain the nearest example with the same class and the nearest example of a different class for each instance in the dataset partition. With this information, two graphs can be built: one where each example points to the nearest example with the same class and another where each example points to the nearest example with a different class. Using these graphs, the method obtains the in-degree for same-class and different-class for each example, and then an information measure based on the K-divergence is calculated. This measure is then used to sort the examples by its ability to represent its own class and to differentiate contiguous classes.

Since the class conditional relation uses classes, firstly it is necessary to discretize the output variable. For that, the Kernel Density Estimation (KDE) with a Gaussian Kernel is used to estimate the probability density function of the output. Then, the local minima of this function are used as the split points between classes, and, therefore, to determine which class corresponds to each example.

Then the instance selection method obtains the subset of selected examples for a *map* (E_S^{map}) from the full training dataset partition (E_{tra}^{map}). It performs the following three stages[2]:

- First, the k_0 first examples —sorted by the K-divergence score— are picked as the initial set of selected examples E_S^{map}, where k_0 is defined as:

$$k_0 = max\left(c, \left\lceil \frac{\epsilon^{E_{tra}^{map}} \cdot |E_{tra}^{map}|}{max(y) - min(y)} \right\rceil\right) \tag{12}$$

  where c is the number of classes obtained from KDE, E_{tra}^{map} is the complete training dataset partition and $\epsilon^{E_{tra}^{map}}$ is the 1-nearest neighbor error for regression using E_{tra}^{map}.

- Then, the second stage adds examples in order, until the error worsens for more than $\sqrt{|E_{tra}^{map}|/|E_S^{map}|}$ iterations.

- Finally, the last stage removes points that are not close to the decision boundary of the 1-nearest neighbor rule, that is, examples with zero in-degree in the different-class graph — there is no other instance that points to the example.

[2]For the sake of completeness, the stages of the algorithm are sketched in this section. A detailed description can be found in [37]

3.2.3. Genetic Algorithm

The evolutionary algorithm is visually described in the Evolutionary Process box in Fig. 3. Its objective is to obtain the best data base configuration using the obtained fuzzy partitions (Fig. 1). To evaluate each individual, the algorithm generates the entire linguistic TSK-1 fuzzy rule base using the selected fuzzy partitions and the selected examples E_S^{map} (sec. 3.2.2), and then calculates the Mean Squared Error (MSE) using the whole training dataset partition E_{tra}^{map}. In the next sections each step of the evolutionary algorithm are described.

Codification

The chromosome codification represents the parameters needed to create the data base using a double coding scheme ($C = C_1 + C_2$):

- C_1 represents the granularity used in each input variable. It is coded as a vector of p_m integers:

$$C_1 = (g_1, g_2, \ldots, g_{p_m}) \tag{13}$$

  where g_i represents the granularity for input variable i. When the granularity of a variable is equal to 1, then it is not used in the antecedent part. However, this variable can still be used in the consequent, since it could be relevant for calculating the output.

- C_2 represents the lateral displacements of the split points of the input variables fuzzy partitions. Thus, the length of C_2 depends on the granularity for each input variable: $|C_2| = \sum_{j=1}^{p}(|g_j| - 1), \forall g_j \in C_1$:

$$C_2 = (\alpha_1^1, \ldots, \alpha_1^{g_1-1}, \ldots, \alpha_p^1, \ldots, \alpha_p^{g_p-1}) \tag{14}$$

  where α_i^j represents the lateral displacement of the j-th split point of variable i. Each lateral displacement are allowed to vary in the $(0.5, 0.5)$ interval which represents half of the distance between each split point (Fig. 4).

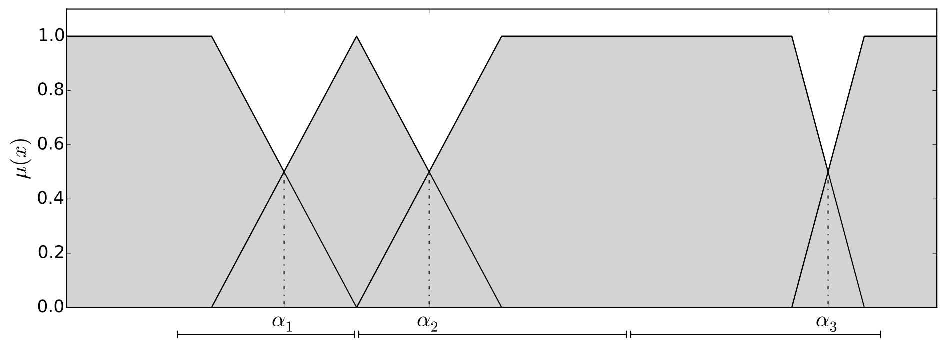

Figure 4: An example of lateral displacement intervals for limits equal to $(0.5, 0.5)$. The split points are allowed to move a maximum of half of the distance to the next split point.

Initialization

The initial pool of individuals is generated by a combination of two initialization procedures. A half of the individuals are generated with the same random granularity for each variable, while the other half is created with a different random granularity for each variable. The lateral displacements are initialized to 0 in all cases. After that, when the product of the granularities indicated in C_1 (i.e., the maximum number of rules that can be obtained) is greater than the number of input variables times the highest maximum granularity of the variables, then a variable is randomly selected to be removed from the antecedent part —its granularity is set to 1— until the previous condition is satisfied. This is done in order to avoid too complex solutions in the initialization stage —during the evolutionary learning this upper bound to the number of rules does not apply.

Crossover

The iterative part of the evolutionary algorithm starts with a binary tournament selection process. Then, two crossover operations can be applied: one-point crossover for exchanging the C_1 parts (it also exchanges the corresponding C_2 genes) and, when the C_1 parts are equal, the parent-centric BLX (PCBLX) [25] is used to crossover the C_2 part. In order to prevent the crossover of too similar individuals, an incest prevention was implemented. When the Euclidean distance of the lateral displacements is less than a particular threshold L, the individuals are not crossed.

Mutation

After the crossover, the mutation is applied to each offspring with probability p_{mut}. It can apply two possible operations with equal probability to a randomly selected gene of the C_1 part: i) decreasing the granularity by 1 or ii) increasing the granularity to a more specific granularity —all the granularities having the same chance. In order to calculate the new lateral displacements in the corresponding C_2 part, the algorithm uses the displacements of the two nearest split points of the previous granularity (before mutation) weighted by the distance between the split points.

Local Search

The offsprings are evaluated using the mechanism described in Sec. 3.2.4. Then, a local search is performed for each offspring, generating n_{ls} new C_1 parts with equal or less granularity —with equal probability— for each variable and the C_2 is generated randomly. The new chromosomes are decoded and evaluated and, if there is a solution that obtains better fitness, then it replaces the original individual. After that, the previous and current populations are merged, and the N best individuals are selected as the new population.

Incest Prevention and Restart Mechanism

The algorithm incorporates a restart mechanism that uses the incest prevention threshold L as a trigger. First, L is initialized as the maximum length of the C_2 part, i.e. the product of the number of input variables times the largest maximum granularity of the variables, divided by 4. This implies that the incest prevention allows crossovers between individuals that have a distance higher than a quarter of the maximum euclidean distance. Then, for each iteration, L is decreased always by 0.4, and by 0.2 if there are no new individuals or the best individual does not change, in order to accelerate convergence. When L reaches 0, the population is restarted, and L is reinitialized. Only the best individual so far is kept, and the local search process is executed with the best individual in order to generate new individuals until the population is complete. When the restart criterion is fulfilled twice, the algorithm stops, i.e., one single restart is executed.

3.2.4. Evaluation

The evaluation process is described in the Evaluation box in Fig. 3. First, the real data base is obtained applying the displacements in C_2 to the fuzzy partitions selected in C_1. Then, the Wang & Mendel algorithm [42] is used to create the antecedent part of the rule base for each individual. The consequent part of the rules is learned using the Elastic Net method [45] in order to obtain the coefficients of the degree 1 polynomial for each rule. In order to solve the minimization problem of Elastic Net, an Stochastic Gradient Descent optimization technique was used [10, 41]. Only those examples in E_S^{map} are used to obtain the rule base from the codified chromosome. In this manner, those examples that are not representative are discarded for the rule generation.

Then, the resulting TSK-1 fuzzy rule base is evaluated using the following equation:

$$fitness = MSE(E_{tra}^{map}) = \frac{1}{2 \cdot |E_{tra}^{map}|} \sum_{i=1}^{|E_{tra}^{map}|} (F(x^i) - y^i)^2, \tag{15}$$

where E_{tra}^{map} is the full training dataset partition and $F(x^i)$ is the output obtained by the knowledge base for the input x^i. Using all the examples for evaluation can be seen, in some way, as a validation process, as the rule base was constructed with a subset of them (E_S^{map}).

3.3. *Aggregation function*

After the execution of FRULER for each dataset partition the reduction phase is applied in order to get the final solution. Each of the FRULER executions obtains a TSK-1 Knowledge Base, composed by a linguistic partition of the input variables — data base — and the TSK fuzzy rule set — rule base. These Knowledge Bases may be combined using an ensemble technique, that takes into account the degree of fulfillment of the input data to each knowledge base, to calculate a weighted average of the outputs. However, this approach increases remarkably the complexity of the model, as the number of rules of the final solution increases with the number of partitions of the data.

Thus, to combine the solutions generated in the Map phase without increasing significantly the complexity, the properties of each of the obtained Knowledge Bases must be taken into account. On one hand, the rule base strongly depends on the label partitions of the data base. A change of granularity in one input variable partition can lead to a substantial change on the rule consequents. On the other hand, the granularity for each input variable can be easily combined, in a similar way to a crossover operation for integer valued individuals. This combination of granularities can take into account the variables not used in each dataset partition. After combining the granularities, the rule base can be generated using the ad-hoc TSK rule base Generation process of FRULER.

```
1: function AGGREGATION($S = \{s_1, s_2, \dots, s_{n_{map}}\}, E_S = E_S^1 \cup E_S^2 \cup \cdots \cup E_S^{n_{map}}$)
2:     for $i = 1, \dots, n_{map}$ do
3:         for $k = 1, \dots, n_{map}$ do
4:             $r_{i,k} = \bigcup_{j=1,\dots,p} \left\{ a_{i,k,j} \, : \, a_{i,k,j} = \begin{cases} s_{i,j} & \text{if } s_{i,j} \neq \text{NULL} \\ s_{k,j} & \text{if } s_{i,j} = \text{NULL and } s_{k,j} \neq \text{NULL} \\ 1 & \text{otherwise} \end{cases} \right\}$
5:         end for
6:     end for
7:     $R = \cup\, r_{i,k},\ i = 1, \dots, n_{map},\ k = 1, \dots, n_{map}$
8:     $error_{min} = \infty$
9:     for each $r \in R$ do
10:        $rb$ = Generate rule base from $r$ using $E_S$
11:        if $MSE(rb, E_{tra}) < error_{min}$ then
12:            $error_{min} = MSE(rb, E_{tra})$
13:            $Best = rb$
14:        end if
15:    end for
16:    return $Best$
```

Figure 5: Pseudocode of the Aggregation function.

Fig. 5 shows the pseudocode of the Aggregation function. The Aggregation function uses two parameters: S contains the solutions generated by FRULER in each dataset partition as a list of granularities, and E_S contains the union of all the selected instances in each dataset partition. Thus, E_S can be seen as the selected instances of the entire training dataset if a stratification approach is used to perform the instance selection method taking each dataset partition as a stratum. The solution obtained in each dataset partition contains the displacements in addition to the granularity for each input variable [3].

The process is as follows: for each partition solution (line 2) the Aggregation function completes the input variables not previously selected in the dataset partition with information of the other solutions (line 3). Thus the Aggregation function generates a maximum of n_{map}^2 final solutions. This combination is done, for each input variable j, as follows (line 4):

- The granularity $s_{i,j}$ (and the displacements associated with it) is selected if it exists.

[3]This information was omitted in the pseudocode for the sake of clarity

- If the value is not defined in s_i ($s_{i,j}$ = NULL) but it is in s_k, then $s_{k,j}$ is used.
- Finally, if the value is not defined neither s_i nor s_k, this variable is not considered (granularity 1).

Note that the inner loop (line 4) takes also into account s_i, thus the original solution is also kept. Then, R (line 7) is the set of the final solutions that combine the information of the different results obtained in each dataset partition.

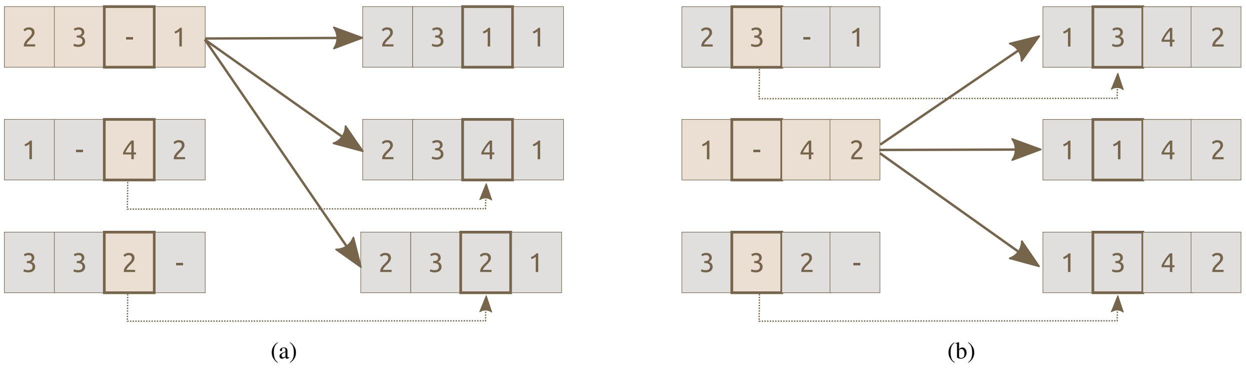

Figure 6: Visual illustration of the Aggregation function for three different solutions obtained in the Map phase.

Fig. 6 illustrates how the solutions obtained by three partitions are combined with this Aggregation function. In the (a) case, the third input variable was removed in the feature selection process and this information is completed with two choices: i) adding a granularity 1 to the variable (the variable is not used) and ii) the granularity indicated in the other solutions. In the (b) scenario, we show the particular case where the other solutions indicate the same granularity for the incomplete information of the selected solution. However, two solutions are created because the lateral displacement information may not be the same.

Once all the possible combinations have been obtained, the Aggregation function generates the associated rule bases (line 10) using the ad-hoc TSK rule base generation process of FRULER. This process uses the combination of selected instances for all dataset partitions E_S. Then, the error is measured using the whole training dataset E_{tra} (line 11), so a validation is performed due to the use of examples not seen in the TSK rule base generation process. Finally, the solution with the lowest error (lines 8-15) is selected as the best solution and returned by the Aggregation function (line 16).

4. Results

In order to analyze the performance of S-FRULER, two types of validation have been done: a) a comparison with other GFSs using 10 regression problems (Sec. 4.2) from the KEEL project repository [3]; and b) an application to a bioinformatics problem (Sec. 4.3), which contains more than 250,000 examples and the number of input variables range from 60 to 180 — available in the Interdisciplinary Computing and Complex BioSystems (ICOS) research group webpage [8]. Table 1 shows the characteristics of the regression datasets, with the number of instances ranging from 7,129 to 40,768 examples, and the number of input variables from 5 to 40. The datasets are sorted in incremental order of the number of variables.

4.1. Experimental Setup

In terms of parameters, S-FRULER only adds to FRULER the parameter of the probability that a particular feature is selected in at least one dataset partition (α_{p_m}). For the experiments performed in this section $\alpha_{p_m} = 0.9$. Moreover, FRULER [37], which is embedded in S-FRULER, was designed to keep the number of parameters as low as possible. In the multi-granularity fuzzy discretization, the fuzziness parameter used for the generation of the fuzzy intervals was 1, i.e., the highest fuzziness value. For the instance selection technique, no parameters are needed. For the evolutionary algorithm, the values of the parameters were: population size = 61, maximum number of evaluations = 100,000, $p_{cross} = 1.0$, $p_{mut} = 0.2$, and the number of neighbours generated in the local search was $n_{ls} = 5$. For the

Problem	Abbr.	# Variables	# Cases
Delta Ailerons	DELAIL	5	7,129
Delta Elevators	DELELV	6	9,517
California Housing	CAL	8	20,640
MV Artificial Domain	MV	10	40,768
House-16H	HOU	16	22,784
Elevators	ELV	18	16,559
Computer Activity	CA	21	8,192
Pole Telecommunications	POLE	26	14,998
Pumadyn	PUM	32	8,192
Ailerons	AIL	40	13,750

Table 1: The 10 datasets of the experimental study.

generation of the TSK fuzzy rule bases, the weight of the tradeoff between ℓ_1 and ℓ_2 regularizations on the Elastic Net was $\alpha = 0.95$, and the regularization parameter λ was obtained from a grid search in the interval $[1, 1E-10]$. η^0 was obtained halving the initial value (0.1) until the result worsens.

S-FRULER[4] was developed entirely using Java version 8. The data partitioning (Sec. 3.1.2), Map Function (Sec. 3.2) and Aggregation function (Sec. 3.3) were implemented using Spark with standard functions.

For each dataset, we performed 10 trials (with different seeds for the random number generation) of S-FRULER. For each trial, the dataset was divided randomly into training (80%) and test (20%). The results shown in the next section are the mean values over all the runs. The runtimes have been obtained in two different ways:

- Using the Spark Standalone mode, where the workers are simulated as threads, executed in an HP Proliant composed by four processors AMD Opteron 6262 HE with a total of 64 cores and 128 GB of memory.
- Using the Spark Cluster mode executed in an Amazon Elastic MapReduce (EMR) 4.0.0 that deploys an Apache Hadoop NextGen MapReduce (YARN) system and uses m3.xlarge machines (Intel Xeon E5-2670 v2 with 4 cores and 15 GB of memory). For each dataset, a different configuration on the number of workers was used depending on the number of dataset partitions, assuring that there is at least one core for each dataset partition.

4.2. Statistical Analysis

In order to evaluate the models learned by S-FRULER, we compared them with three of the most accurate genetic GFSs for regression in the literature[5]:

- $\mathrm{FS_{MOGFS}}^{e}$+TUN^{e} [2]: a multi-objective evolutionary algorithm that learns Mamdani fuzzy rule bases. This algorithm learns the granularities from uniform multi-granularity fuzzy partitions (up to granularity 7) and the lateral displacement of the labels. It includes a post-processing algorithm for tuning the parameters of the membership functions and for rule selection.
- L-METSK-HD^{e} [20]: a multi-objective evolutionary algorithm that learns linguistic TSK-0 fuzzy rule bases. The algorithm learns the granularities from uniform multi-granularity fuzzy partitions (up to granularity 7).
- A-METSK-HD^{e} [20]: a multi-objective evolutionary algorithm that learns approximative TSK-1 fuzzy rule bases. The algorithm starts with the solution obtained on the first stage and applies a tuning of the membership functions, rule selection and a Kalman-based calculation of the consequents of the rules.

Tables 2 and 3 show the results obtained by S-FRULER and the other three GFSs for the datasets in Table 1 for precision — the mean test error over the executions — and complexity — the number of rules (# Rules) of the

[4]The software is available at `http://tec.citius.usc.es/fruler/`

[5]Although FRULER [37] has a better accuracy than these GFSs, we excluded it from the comparison, since S-FRULER is based on FRULER.

Algorithms	S-FRULER	A-METSK-HD^e	FS$_{\text{MOGFS}}$^{e}+TUN^e	L-METSK-HD^e
DELAIL	1.44	**1.40**	1.53	1.62
DELELV	1.12	**1.03**	1.09	1.12
CAL	2.18	**1.71**	2.95	2.64
MV	**0.05**	0.06	0.16	0.25
HOU	**8.2**	8.5	9.4	10.4
ELV	**3.2**	7.0	9.0	8.9
CA	**4.6**	5.0	5.2	5.9
POLE	124	**61**	103	151
PUM	0.349	**0.287**	0.292	0.594
AIL	**1.4**	1.5	2.0	1.8
Friedman Ranking	1.8	1.5	3	3.7
Holm p-value		0.603	0.075	0.003

Table 2: Average test errors for the different algorithms. The errors in this table should be multiplied by $10^{-8}, 10^{-6}, 10^{9}, 10^{8}, 10^{-6}, 10^{-4}, 10^{-8}$ in the case of DELAIL, DELELV, CAL, HOU, ELV, PUM, AIL respectively. The best results are marked in bold face. It also shows the Friedman Ranking for each approach and the Holm adjusted p-value comparing S-FRULER with the other algorithms.

Algorithms	S-FRULER	A-METSK-HD^e	FS$_{\text{MOGFS}}$^{e}+TUN^e	L-METSK-HD^e
DELAIL	**3**	37	6	98
DELELV	**2**	39.1	7.9	91
CAL	**7**	56	8	100
MV	**4**	56	14	76
HOU	**11**	30	12	69
ELV	**5**	35	8	76
CA	**11**	32	14	71
POLE	20	46	**13**	100
PUM	**4**	63	18	88
AIL	**11**	48	15	99
Friedman Ranking	1.1	3	1.9	4
Holm p-value		0.002	0.166	< 1E-3

Table 3: Average number of rules for the different algorithms. The best results are marked in bold face. It also shows the Friedman Ranking for each approach and the Holm adjusted p-value comparing S-FRULER with the other algorithms.

final solution. Moreover, in order to analyze statistically the difference between the different approaches, a Friedman ranking test followed by a Holm post-hoc test were performed for both measures. The Friedman ranking test calculates a rank for each approach based on its performance, where the lower the rank, the better. On the other hand, the Holm post-hoc method tests if the difference between two rankings is significant. We applied the Holm post-hoc test using the S-FRULER approach as a control method, thus calculating the p-value for each comparison of S-FRULER with each of the other approaches.

In the case of precision (Table 2), both S-FRULER and A-METSK-HD^e achieve the best result in 5 of the datasets. A-METSK-HD^e obtains the lowest rank (1.5) followed by S-FRULER (1.8), however the difference between them is rather low, since the Holm p-value is high (0.603). FS$_{\text{MOGFS}}$^{e}+TUN^e and L-METSK-HD^e obtain worse errors than S-FRULER (ranks 3 and 3.7 respectively) and the differences are statistically significant (Holm p-values below 0.1).

In terms of complexity (Table 3), S-FRULER obtains the best result in 9 of 10 datasets, while FS$_{\text{MOGFS}}$^{e}+TUN^e obtains the best result in the remaining dataset (POLE). The Friedman ranking test shows these results, where S-FRULER obtains the lowest rank (1.1) followed by FS$_{\text{MOGFS}}$^{e}+TUN^e (1.9), while A-METSK-HD^e and L-METSK-HD^e obtain the worst ranking (3 and 4 respectively). Comparing S-FRULER with FS$_{\text{MOGFS}}$^{e}+TUN^e, the Holm p-value

Algorithms	FRULER	S-FRULER	Standalone		Cluster		A-METSK-HD^e
	Time	n_{map}	Time	Speedup	Time	Speedup	Time
DELAIL	0:09:58	21	0:01:18	8	0:00:48	12	2:30:39
DELELV	0:25:01	22	0:01:38	15	0:01:13	21	1:29:30
CAL	1:57:03	23	0:04:20	27	0:03:22	35	5:13:28
MV	1:17:02	23	0:09:27	8	0:05:49	13	3:17:54
HOU	4:15:17	25	0:04:06	62	0:03:09	81	5:07:58
ELV	3:01:30	26	0:03:10	57	0:03:14	56	3:06:58
CA	0:38:12	25	0:03:46	10	0:01:48	21	3:37:49
POLE	1:53:15	27	0:10:20	11	0:05:14	22	4:40:22
PUM	31:14:27	24	0:01:58	956	0:01:39	1,139	2:22:25
AIL	12:50:38	28	0:07:13	107	0:03:32	218	5:26:30

Table 4: Runtime comparison between FRULER, S-FRULER and A-METSK-HD^e (the single GFS which is not statistically different from S-FRULER —Table 2). All times are formatted in hours:minutes:seconds. Runtimes of A-METSK-HD^e were obtained using a different computer (Intel Core 2 Quad Q9550 2.83GHz, 8GB RAM).

is rather low (0.166) showing a noticeable difference, while the difference with A-METSK-HD^e and L-METSK-HD^e is remarkable (p-value below 0.005).

Finally, we compare S-FRULER with FRULER [37] in terms of runtime of the full learning process in order to highlight the obtained speedup. Table 4 shows the runtimes of FRULER and S-FRULER and also shows the speedup obtained in each execution mode of S-FRULER (Standalone using threads or in a Cluster) when compared with FRULER and the number of dataset partitions n_{map} obtained for each dataset. We also show the runtime with A-METSK-HD^e, the single GFS which is not statistically different from S-FRULER (Table 2).

It can be seen that the number of dataset partitions depends on the problem, but, in general, as the number of variables increases so does the number of dataset partitions used. S-FRULER, in both Standalone (threads) and Cluster, obtains lower runtimes than FRULER, as expected. The speedup depends on the problem, as datasets with a similar number of dataset partitions (e.g., DELAIL, DELELV, CAL and MV with 21-23 partitions) have a very different speedup (8, 15, 27 and 8 respectively in Standalone or 12, 21, 35 and 13 in Cluster mode). It is worth emphasizing that for the problems with the worst runtime in FRULER (HOU, ELV, PUM and AIL), S-FRULER achieves speedups far above the parallelization level given by the number of dataset partitions. This is because the scalability obtained by S-FRULER is not only given by the distributed approach, but also due to the faster convergence of FRULER in each dataset partition as the partitioned problem is more simple.

These results show that S-FRULER not only reduces FRULER runtimes, but also is able to learn the simplest models in terms of number of rules, using a linguistic approach, while maintaining a high precision comparable to an approximative approach.

4.3. Application to Bioinformatics

As stated in the introduction, the large scale regression problems solved with GFSs in the literature are still far from those considered Big Data problems in terms of volume and velocity. A high computational cost when the X matrix (Eq. 6) is huge, due to number of examples and/or number of variables, can be solved with parallelization approaches, e.g. solving several X matrix at the same time. However, if the storage demand of various X cannot be stored in memory of a single processor, it is necessary the use of Big Data approaches, such as distributed computing. In order to demonstrate the capability of S-FRULER in a challenging domain, we have applied our approach to a real bioinformatics problem with much higher computational requirements than the datasets of the previous section.

The problem is focused on one of the main open problems in computational biology which is the prediction of the 3D structure of protein chains [9]. One approach to this problem is to predict some attributes of a protein, such as the secondary structure, the solvent accessibility or the coordination number (CN), and then integrate this knowledge into a full 3D structure predictor. Particularly, the CN problem is defined as the prediction, for a given residue, of the number of residues of the same protein that are in contact with it in the native state. Two residues are said to be in contact when the distance between them is below a certain threshold. Figure 7 shows a graphical representation of the CN — in this case the CN is 4— for a particular residue, discarding trivial contacts.

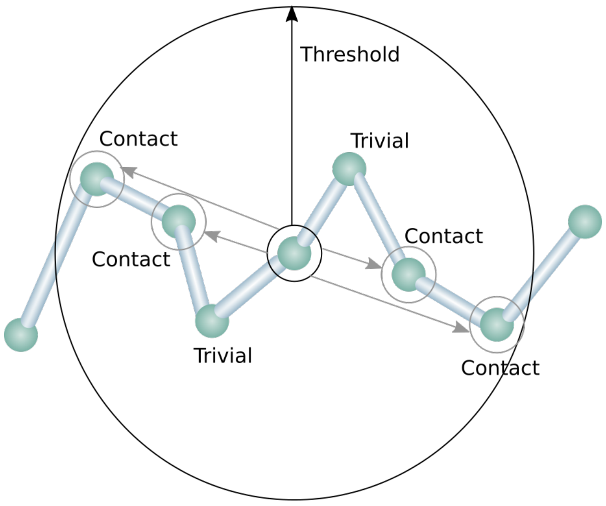

Figure 7: Graphical representation of the CN of a residue [9].

In [27] was proposed a real-valued definition of contact, and linear regression is used to predict the CN based on the amino acid type of the protein primary sequence and global information about the protein. The amino acid information can be extracted from the primary sequence, where one amino acid is defined as one nominal variable with 20 possible values. However, a Position-Specific Scoring Matrices (PSSM) representation derived from the primary sequence can be used to represent the amino acid information. The PSSM representation is an statistical profile of the primary sequence that takes into account how this sequence may have evolved. Thus, each amino acid is defined as 20 continuous variables.

The amino acids used to predict the CN can be extracted from a local context (a windows of amino acids) of the target in the chain. Therefore, the number of variables used to solve the problem can increase in blocks of 40 features as the window expands. For example, with the first neighbors two amino acids are added (before and after the target), each with 20 new features. Given these characteristics, in [38] this problem is proposed as a benchmark for testing the scalability of regression methods. In this manner, different datasets can be constructed adjusting the number of examples and/or variables used. The available datasets can be downloaded from the PSP Benchmark project web site [8].

S-FRULER was used to solve four versions of this dataset with different number of variables, depending on the size of the window of amino acids considered (Table 5). We have tested FRULER on these datasets, but it fails to converge —in less than three days— in the three most complex problems. As the runtime of FRULER is usually lower than the GFSs used for the comparisons in the previous section [20], it can be assumed that these GFSs would also fail on these datasets. We compared S-FRULER with the regression methods implemented in the Spark scalable machine learning library MLlib [31]. The methods available in this resource include Ridge Regression (ℓ_2 regularization) and Lasso Regression (ℓ_1 regularization), both solved using distributed mini-batch SGD. The regularization parameter λ was obtained in the same way as S-FRULER (see [37] for more details). The MLlib results presented in Table 5 correspond to 1,500 iterations. Nevertheless these methods were run for $3,000$ iterations with no further improvement.

Table 5 shows the characteristics of these four datasets, with the mean squared test error obtained by S-FRULER, Ridge and Lasso, and their runtime using an HP Proliant composed of four processors AMD Opteron 6262 HE with a total of 64 cores and 128 GB of memory. The test error in S-FRULER is lower than Ridge and Lasso approaches for all the datasets. Furthermore, the precision obtained with the less informative dataset (w1) is better than the result obtained by Ridge and Lasso with the dataset with more information (w4). Since the error of S-FRULER is always better than the other approaches —11-21% better than Ridge and 53-56% better than Lasso—, a statistical test is not necessary to confirm that S-FRULER performs significantly the best. The highest test error using S-FRULER occurs for w1, as it is the dataset with less information. On the other hand, when the number of input variables is really high

			S-FRULER			Ridge SGD		Lasso SGD	
Dataset	# Cases	# Vars	# n_{map}	Test Error	Time	Test Error	Time	Test Error	Time
w1	257,560	60	33	**12.45**	04:42:08	15.09	**00:42:56**	19.01	00:45:05
w2	257,560	100	34	**12.15**	05:32:33	14.20	**01:00:44**	18.91	01:01:25
w3	257,560	140	35	**12.23**	05:48:42	14.18	**01:04:36**	18.89	01:04:43
w4	257,560	180	36	**12.30**	12:17:42	13.62	01:06:42	18.93	**01:05:46**

Table 5: Datasets characteristics for the four CN problems, Mean Squared Error (MSE), and runtime (hours:minutes:seconds) for S-FRULER, Ridge and Lasso MLlib implementations.

(w4), the test error worsens very slightly when compared to w3. In the case of Ridge, the error improves even with the largest dataset (w4), since it uses all the variables in some degree. However, this has the drawback of increasing the complexity of the obtained model, while a low complexity model based on rules can be used by the expert. Regarding the runtimes, S-FRULER always takes longer times to get its best model. We have executed Ridge and Lasso for longer runtimes (comparable to those of S-FRULER) but they did not further improve.

We have also compared S-FRULER with four approaches presented in [11], where the regression problem was transformed into a classification problem with two classes using an uniform-frequency discretization. The algorithms in the comparison [11] are:

- GAssist[6]: is a Pittsburgh-style learning classifier system (LCS). It uses a standard genetic algorithm to evolve a population of individuals, each of them being a complete and variable-length rule set.
- BioHEL[7]: is an evolutionary learning system that follows the iterative rule learning approach and is designed to handle large-scale bioinformatic datasets.
- C4.5[32]: builds decision trees from a set of training data using the concept of information entropy. At each node of the tree, C4.5 chooses the attribute of the data that most effectively splits its set of samples into subsets that maximize the presence of one class.
- PART[18]: iteratively builds a partial C4.5 decision tree in each iteration and transforms the most promising leaf into a rule.

Algorithm	S-FRULER	BioHEL	GAssist	PART	C4.5
w1	**23.1**	24.2	25.2	29.1	31.4
w2	**22.7**	24.0	25.3	29.1	31.4
w3	**22.3**	23.6	25.4	30.1	31.9
w4	**22.6**	23.5	25.2	24	31.8
Friedman Ranking	1	2	3.25	3.75	5
Holm Adj. p-value		0.371	0.088	0.042	0.001

Table 6: Comparison of error (in percentage) for the four CN problems discretized into 2 classes. The Friedman ranking and the p-value were obtained by comparing each approach with S-FRULER.

In order to compare the regression models generated by S-FRULER with the classification models [11], the output of the regression model is discretized using the same uniform-frequency discretization that transformed the regression problem into a classification one. Table 6 shows the error (in percentage) obtained by S-FRULER compared to the other four approaches. It also displays the Friedman ranking test followed by a Holm post-hoc test comparing S-FRULER with the other approaches. In this particular case, the statistical tests are less reliable due to the low ratio between the number of datasets and the number of algorithms. In spite of this, we show on Tables 6 and 7 the p-values for the sake of completeness. S-FRULER obtains the best result in all the datasets and gets the best ranking in the Friedman test, followed by the other genetic approaches BioHEL and GAssisst.

Algorithm	S-FRULER	BioHEL	GAssist	PART	C4.5
w1	**31.6**	110.0	40.2	6,271.9	22,662.1
w2	**16.2**	110.9	36.5	7,889.5	22,144.3
w3	**7.6**	113.0	35.4	7,450.8	21,342.7
w4	52.7	113.6	**36.5**	7,006.0	20,671.1
Friedman Ranking	1.25	3	1.75	4	5
Holm Adj. p-value		0.235	0.655	0.042	0.003

Table 7: Comparison of complexity (number of rules/leaf nodes) for the four CN problems discretized into 2 classes. The Friedman ranking and the p-value were obtained by comparing each approach with S-FRULER.

In order to compare the complexity of the models, the number of rules of the resulting models are shown in table 7. In the case of C4.5, the complexity is measured as the number of leaf nodes. S-FRULER obtains the best results in three datasets (w1, w2 and w3) and the lowest ranking (1.25) while GAssist obtains the best result for w4 and the second lowest ranking (1.75).

The results presented in this section show that S-FRULER can cope with large datasets in both number of examples and number of attributes. It was also demonstrated that the models learned by S-FRULER obtain the best results in both accuracy and complexity when compared with other state-of-the-art techniques

5. Conclusions

In this paper, we have presented an scalable version of FRULER [37] —a GFS that learns simple and linguistic TSK-1 knowledge bases for regression problems—, called S-FRULER. S-FRULER obtains simple models with high precision but reducing the runtime of FRULER. The approach is based on a distributed computing methodology and was implemented using the Spark software, thus providing both standalone and cluster modes. S-FRULER partitions the dataset into small sets, which are more tractable, and executes FRULER for each of them. Also, for each dataset partition, a random feature selection process to reduce the number of variables is used. After the Map phase, the method implements an aggregation function to obtain linguistic TSK-1 fuzzy rule bases from the rule bases generated in each dataset partition. The use of Spark facilitates the building of parallel applications that can run on different cluster solutions, e.g. Hadoop YARN, providing some Big Data desired properties like fault tolerance. Nevertheless, further improvements may be applied to S-FRULER in order to take advantage of this type of architecture, such as adopting a scheme more similar to the MapReduce approach.

S-FRULER has speedups usually larger than the number of dataset partitions used, showing an scalability higher than linear in both standalone and cluster mode. It was compared with three GFSs in terms of complexity of the models (number of rules) and precision (mean squared error), showing good results with less complex models. S-FRULER was also applied to a large-scale problem in computational biology: the prediction of the coordinate number of a residue in the context of prediction of the 3D structure of protein chains. Results demonstrate the capability of S-FRULER to obtain precise and simple models in large scale problems.

Acknowledgments

This research was supported by the Spanish Ministry of Economy and Competitiveness (grant TIN2014-56633-C3-1-R) and the Galician Ministry of Education (grants EM2014/012, CN2012/151 and GRC2014/030). All grants were co-funded by the European Regional Development Fund (FEDER program). I. Rodriguez-Fdez was supported by the Spanish Ministry of Education, under the FPU national plan (AP2010-0627).

The authors are extremely grateful and sincerely acknowledge the help during this work provided by Dr. Jaume Bacardit and the Interdisciplinary Computing and Complex BioSystems (ICOS) research group of the Newcastle University.

References

[1] Alcalá, R., Alcalá-Fdez, J., Herrera, F., Otero, J., 2007. Genetic learning of accurate and compact fuzzy rule based systems based on the 2-tuples linguistic representation. International Journal of Approximate Reasoning 44 (1), 45–64.

[2] Alcalá, R., Gacto, M. J., Herrera, F., 2011. A fast and scalable multiobjective genetic fuzzy system for linguistic fuzzy modeling in high-dimensional regression problems. IEEE Transactions on Fuzzy Systems 19 (4), 666–681.

[3] Alcala-Fdez, J., Sanchez, L., Garcia, S., del Jesus, M. J., Ventura, S., Garrell, J., Otero, J., Romero, C., Bacardit, J., Rivas, V. M., et al., 2009. Keel: a software tool to assess evolutionary algorithms for data mining problems. Soft Computing 13 (3), 307–318.

[4] Antonelli, M., Ducange, P., Marcelloni, F., 2013. An efficient multi-objective evolutionary fuzzy system for regression problems. International Journal of Approximate Reasoning 54 (9), 1434–1451.

[5] Azar, A. T., Hassanien, A. E., 2015. Dimensionality reduction of medical big data using neural-fuzzy classifier. Soft computing 19 (4), 1115–1127.

[6] Bacardit, J., 2004. Pittsburgh genetics-based machine learning in the data mining era: Representations, generalization, and run-time. Ph.D. thesis, Ramon Llull University, Barcelona, Spain.

[7] Bacardit, J., Burke, E. K., Krasnogor, N., Mar. 2009. Improving the scalability of rule-based evolutionary learning. Memetic Computing 1 (1), 55–67.

[8] Bacardit, J., Krasnogor, N., 2008. The ICOS PSP benchmarks repository. `http://ico2s.org/datasets/psp_benchmark.html`.

[9] Bacardit, J., Stout, M., Krasnogor, N., Hirst, J. D., Blazewicz, J., 2006. Coordination number prediction using learning classifier systems. In: Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO '06. Seattle, Washington, USA, p. 247.

[10] Bottou, L., 2010. Large-scale machine learning with stochastic gradient descent. In: Proceedings of the 19th International Conference on Computational Statistics. Springer, pp. 177–186.

[11] Calian, D. A., Bacardit, J., 2013. Integrating memetic search into the BioHEL evolutionary learning system for large-scale datasets. Memetic Computing 5 (2), 95–130.

[12] Cordón, O., Herrera, F., Hoffmann, F., Magdalena, L., Cordon, O., Herrera, F., Hoffmann, F., 2001. Genetic fuzzy systems. World Scientific Publishing Company Singapore.

[13] Dean, J., Ghemawat, S., 2008. MapReduce: simplified data processing on large clusters. Communications of the ACM 51 (1), 107–113.

[14] Ducange, P., Marcelloni, F., Segatori, A., 2015. A mapreduce-based fuzzy associative classifier for big data. In: 2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). IEEE, pp. 1–8.

[15] Fazzolari, M., Alcalá, R., Nojima, Y., Ishibuchi, H., Herrera, F., 2013. A review of the application of multiobjective evolutionary fuzzy systems: Current status and further directions. IEEE Transactions on Fuzzy Systems 21 (1), 45–65.

[16] Fernández, A., Carmona, C. J., del Jesus, M. J., Herrera, F., 2016. A view on fuzzy systems for big data: Progress and opportunities. International Journal of Computational Intelligence Systems 9 (sup1), 69–80.

[17] Fernández, A., López, V., del Jesus, M. J., Herrera, F., 2015. Revisiting evolutionary fuzzy systems: Taxonomy, applications, new trends and challenges. Knowledge-Based Systems 80, 109–121.

[18] Frank, E., Witten, I. H., 1998. Generating accurate rule sets without global optimization. In: Shavlik, J. (Ed.), Fifteenth International Conference on Machine Learning. Morgan Kaufmann, pp. 144–151.

[19] Friedman, J., Hastie, T., Tibshirani, R., 2010. Regularization paths for generalized linear models via coordinate descent. Journal of statistical software 33 (1), 1.

[20] Gacto, M. J., Galende, M., Alcalá, R., Herrera, F., 2014. Metsk-hd e: A multiobjective evolutionary algorithm to learn accurate tsk-fuzzy systems in high-dimensional and large-scale regression problems. Information Sciences 276, 63–79.

[21] García-Pedrajas, N., de Haro-García, A., 2012. Scaling up data mining algorithms: review and taxonomy. Progress in Artificial Intelligence 1 (1), 71–87.

[22] Hastie, T., Tibshirani, R., Friedman, J., Hastie, T., Friedman, J., Tibshirani, R., 2009. The elements of statistical learning. Springer.

[23] Havens, T. C., Bezdek, J. C., Leckie, C., Hall, L. O., Palaniswami, M., 2012. Fuzzy c-means algorithms for very large data. IEEE Transactions on Fuzzy Systems 20 (6), 1130–1146.

[24] Herrera, F., 2008. Genetic fuzzy systems: taxonomy, current research trends and prospects. Evolutionary Intelligence 1 (1), 27–46.

[25] Herrera, F., Lozano, M., Sánchez, A. M., 2003. A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. International Journal of Intelligent Systems 18 (3), 309–338.

[26] Ishibuchi, H., Yamamoto, T., 2002. Performance evaluation of fuzzy partitions with different fuzzification grades. In: Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). Vol. 2. IEEE, pp. 1198–1203.

[27] Kinjo, A. R., Horimoto, K., Nishikawa, K., 2005. Predicting absolute contact numbers of native protein structure from amino acid sequence. Proteins: Structure, Function, and Bioinformatics 58 (1), 158–165.

[28] López, V., del Río, S., Benítez, J. M., Herrera, F., 2015. Cost-sensitive linguistic fuzzy rule based classification systems under the mapreduce framework for imbalanced big data. Fuzzy Sets and Systems 258, 5–38.

[29] Mamdani, E. H., 1974. Application of fuzzy algorithms for control of simple dynamic plant. In: Proceedings of the Institution of Electrical Engineers. Vol. 121. IET, pp. 1585–1588.

[30] Mamdani, E. H., Assilian, S., 1975. An experiment in linguistic synthesis with a fuzzy logic controller. International journal of man-machine studies 7 (1), 1–13.

[31] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al., 2015. MLlib: Machine learning in apache spark. arXiv preprint arXiv:1505.06807.

[32] Quinlan, R., 1993. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, CA.

[33] Reyes-Ortiz, J. L., Oneto, L., Anguita, D., 2015. Big data analytics in the cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf. Procedia Computer Science 53, 121–130.

[34] Rodríguez-Fdez, I., Mucientes, M., Bugarín, A., 2012. Photons detection in positron emission tomography through iterative rule learning

of TSK rules. In: Actas del VIII Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB). Albacete (Spain), pp. 251–258.

[35] Rodríguez-Fdez, I., Mucientes, M., Bugarín, A., 2013. An instance selection algorithm for regression and its application in variance reduction. In: Proceedings of the 2013 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). pp. 1–8.

[36] Rodríguez-Fdez, I., Mucientes, M., Bugarín, A., 2015. Reducing the complexity in genetic learning of accurate regression TSK rule-based systems. In: Proceedings of the 2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). IEEE, pp. 1–8.

[37] Rodríguez-Fdez, I., Mucientes, M., Bugarín, A., 2016. FRULER: Fuzzy rule learning through evolution for regression. Information Sciences, doi:10.1016/j.ins.2016.03.012.

[38] Stout, M., Bacardit, J., Hirst, J. D., Krasnogor, N., Apr. 2008. Prediction of recursive convex hull class assignments for protein residues. Bioinformatics 24 (7), 916–923.

[39] Sugeno, M., Kang, G., 1988. Structure identification of fuzzy model. Fuzzy sets and systems 28 (1), 15–33.

[40] Takagi, T., Sugeno, M., 1985. Fuzzy identification of systems and its applications to modeling and control. IEEE Transactions on Systems, Man and Cybernetics (1), 116–132.

[41] Tsuruoka, Y., Tsujii, J., Ananiadou, S., 2009. Stochastic gradient descent training for L1-regularized log-linear models with cumulative penalty. In: Proceedings of the Joint conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing. Association for Computational Linguistics, pp. 477–485.

[42] Wang, L.-X., Mendel, J. M., 1992. Generating fuzzy rules by learning from examples. IEEE Transactions on Systems, Man and Cybernetics 22 (6), 1414–1427.

[43] White, T., 2012. Hadoop: The definitive guide. O'Reilly Media, Inc.

[44] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., Stoica, I., 2010. Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. Vol. 10. p. 10.

[45] Zou, H., Hastie, T., 2005. Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society 67 (2), 301–320.