Subgroup Discovery with Evolutionary Fuzzy Systems in R: the **SDEFSR** Package

Angel M. Garcia, Francisco Charte, Cristóbal J. Carmona, Pedro Gonzalez, Maria J. del Jesus

2020-01-28

Abstract

Subgroup discovery is a data mining task half-way between descriptive and predictive data mining. Nowadays it is very relevant for researches due to the fact that the knowledge extracted is simple and interesting. For this task, evolutionary fuzzy systems are well suited algorithms because they can find a good trade-off between multiple objectives in large search spaces. In fact, this paper presents the SDEFSR package, which contains all evolutionary fuzzy systems for subgroup discovery presented throughout the literature. It is a package without dependencies on other software, providing functions with recommended default parameters. In addition, it brings a graphical user interface to avoid the user having to know all the algorithm's parameters.

1 Introduction

Subgroup discovery (SD) is a data mining field that aims to describe data using supervised learning techniques. The goal is to find simple, general and interesting patterns with respect to a given variable of interest. Throughout the literature, SD has been applied with success to different real-world problems in areas such as marketing [del Jesus et al.(2007), Berlanga et al.(2006)], medicine [Carmona et al.(2015), Carmona et al.(2013), Stiglic and Kokol(2012), Gamberger et al.(2003)] and e-learning [Poitras et al.(2016), Lemmerich et al.(2011), Carmona et al.(2010b)], among others [Atzmueller et al.(2016), Jin et al.(2014), Rodriguez et al.(2013), Carmona et al.(2012)].

SD is a rule learning process within a complex search space. Therefore, the search strategy used becomes a key factor in the efficiency of the method. Different strategies can be found in the literature such as beam search in the algorithm CN2-SD [Lavrač et al.(2004b)] and Apriori-SD [Kavšek and Lavrač(2006)], exhaustive algorithms such as SDMap [Atzmueller and Puppe(2006)] or Evolutionary Algorithms (EAs), for example.

EAs are stochastic algorithms for optimizing and searching tasks, based on the natural evolution process [John(1992)]. There are different paradigms within EAs: genetic algorithms [John(1992), Goldberg(1989)], evolution strategies [Schwefel(1995)], evolutionary programming [Fogel(2006)] and genetic programming [Koza(1992)]. With these methods the inclusion of rules as knowledge representation is known as evolutionary rule-based systems [Freitas(2003)] and has the advantage of allowing the inclusion of domain knowledge and returning better rules. The use of EAs is very well suited to SD because these algorithms perform a global search in the space in a convenient way, stimulating the obtaining of simple, interesting and precise subgroups.

Nowadays, there are several frameworks that allow the realization of data mining tasks, but only a few of them have implementations of SD algorithms. The most well-know frameworks with SD algorithms are KEEL [Alcalá-Fdez et al.(2011)] and VIKAMINE [Atzmueller and Lemmerich(2012)], but ORANGE [Demšar et al.(2013)], and CORTANA [Meeng and Knobbe(2011)] provide some implementations too. In fact, VIKAMINE also provides an R package called **rsubgroup** [Atzmueller(2014)] which is an interface for R to VIKAMINE Java algorithms.

In this contribution the **SDEFSR** package is introduced. It provides the user with the most important evolutionary fuzzy rule-based methods for SD documented in the literature, being a major contribution to the R community. In addition, it also brings the capability of reading datasets in the KEEL data format. This file format is not supported by R. Similarly, this package provides a Graphical User Interface (GUI) to make this task easier for the user, especially the novel one.

This contribution is organized according to the following structure: The first section presents SD concepts, main properties and features. In the second section, the structure of the SDEFSR package and its operations are described. In the third section, a complete example of use of the package is presented. Finally, the GUI of SDEFSR is shown.

2 Subgroup Discovery

SD was defined by [Wrobel(2001)] as:

> In subgroup discovery, we assume we are given a so-called population of individuals (objects, customer, . . .) and a property of those individuals we are interested in. The task of subgroup discovery is then to discover the subgroups of the population that are statistically "most interesting", i.e. are as large as possible and have the most unusual statistical (distributional) characteristics with respect to the property of interest.

SD tries to find relations between different properties of a set with respect to one interesting or target variable. Such relations must be statistically interesting, so it is not necessary to find complete relations, partial relations could be interesting too.

Usually these relations are represented by means of rules, one per relation. These rules are defined as [Lavrač et al.(2004a), Gamberger and Lavrac(2002)]:

$$R : Cond \rightarrow Target_{value} \tag{1}$$

where $Target_{value}$ is the value for the variable of interest (target variable) for the SD task and $Cond$ is normally a conjunction of attribute-value pairs which describe the characteristics of the induced subgroup.\ SD is halfway between description and classification, because it has a target variable but its objective is not to predict it but rather to describe it. The use of a target variable is not possible in description, because description simply finds relationships between unlabeled object.\ A key point to fully understanding the goal of SD is how it differentiates from the classification objective. Classification is a predictive task that tries to split the entire search space, usually in a complex way, aiming to find the correct value for the variable in new incoming instances. On the other hand, SD aims to find interesting relations among these instances regarding the variable of interest.\ For instance, assuming there is a group of patients, the variable of interest is whether they have heart disease or not. The predictive data mining objective is to predict if new patients will have heart disease. However, SD tries to find which subgroup of patients are more likely to have heart disease according to certain characteristics. This is relevant for developing treatment against those characteristics, for instance.

2.1 Main elements of subgroup discovery algorithms

Below, the most relevant aspects of SD algorithms are presented [Atzmueller et al.(2004)]:

- *Type of target variable:* This is the kind of information the interest variable can hold. The target variable could be binary (two possible values), categorical (n posible values) or numerical (a real value within a range). Nevertheless, the majority of SD algorithms can only deal with binary or categorical target variables.
- *Description language:* Knowledge representation is a key factor in SD due to its descriptive nature. In this way, rules must be as simple as possible. Rules are usually represented by conjunctions of attribute-value pairs or in disjunctive normal form. Fuzzy logic could also be included in the rules in order to improve the interpretability of the knowledge [Zadeh(1975), Hüllermeier(2005)].
- *Quality measures:* This is the most important aspect in the design of SD algorithms. The quality measures must guide the learning process and must show the quality of the extracted knowledge. They will be briefly described below.
- *Search strategy:* The search space grows exponentially with the number of variables.The use of a search strategy able to find a good solution, or the optimal one, by searching efficiently through the whole search space is very important.

2.2 Quality measures for subgroup discovery

A quality measure tries to measure the interestingness of a given rule or subgroup, but there is not a formal definition of what interestingness is. However, the interestingness could be defined as a concept that emphasizes conciseness, coverage, reliability, peculiarity, diversity, novelty, surprisingness, utility, and actionability [Geng and Hamilton(2006)]. For SD, the most used criteria to measure the interestingness of a rule are conciseness, generality or coverage, reliability, and novelty [Carmona et al.(2014)].

Quality measures that accomplish this criteria available in the **SDEFSR** package are:

- Measures for conciseness (or complexity). It measures the complexity of the induced rules. Rules with a few number of attribute-value pairs are easy to remember and to add to the expert's knowledge. The quality measures associated to this criterion are:

  - N_r: The number of rules generated. A rule set with a large number of rules is much more difficult to remember than other that has less rules. Additionally, the lower the number of rules, the easier for the expert to filter those rules that are interesting.
  - N_v: The number of variables that have the rules generated. Rules with less number of variables are easier to understand and to remember, also, it tends to have more generality. Thus, rules with low number of variables are interesting.
- Measures for generality (or coverage). It measures the capacity of a rule to match with a great number of examples in the dataset. Also, the capacity to generalize the rule to other instances that are not in the training dataset is greater. The quality measures associated to this criterion are:
  - *Support:* It measures the frequency of correctly classified examples covered by the rule:

$$Sup(R) = \frac{n(Cond \wedge Target_{value})}{n_s} \tag{2}$$

    where $n(Cond \wedge Target_{value})$ means the number of examples that satisfy the antecedent and consequent part of the rule and n_s is the number of examples in the dataset.
  - *Coverage:* It measures the percentage of examples covered by the rule related to the total number of examples:

$$Cov(R) = \frac{n(Cond)}{n_s} \tag{3}$$

    where $n(Cond)$ means the number of examples that satisfy the antecedent part of the rule.
- Measures for reliability. A rule is reliable when the relation described in the rule occurs in a high percentage of cases where the rule can be applied. The quality measures associated to this criterion are:
  - *Confidence:* It measure the percentage of examples correctly covered of the total of covered examples:

$$Conf(R) = \frac{n(Cond \wedge Target_{value})}{n(Cond)} \tag{4}$$

- Measures for novelty. A rule is novel if the knowledge obtained from this one is unknown by the user or it is unable to infer such knowledge from other rules. For this kind of criterion, the quality measures availables in the package are:
  - *Significance:* It reflects the novelty in the distribution of the examples covered by the rule regarding the whole dataset:

$$Sign(R) = 2 \cdot \sum_{k=1}^{n_c} n\left(Cond \wedge Target_{value_k}\right) \cdot log\left(\frac{n\left(Cond \wedge Target_{value_k}\right)}{n(Cond \wedge Target_{value}) \cdot p(Cond)}\right) \tag{5}$$

    where $p(Cond) = \frac{n(Cond)}{n_s}$, n_c is the number of possible values of the target variable and $Target_{value_k}$ is the value k-th value of the target variable.
- Hybrid measures. These measures try to maximize more than one criterion with a single expression that finds a good trade-off between the criteria used. The hybrid quality measures implemented are:
  - *Unusualness:* It is defined as the weigthed relative accuracy of a rule and tries to maximize generality and realiability:

$$WRAcc(R) = \frac{n(Cond)}{n_s}\left(\frac{n(Cond \wedge Target_{value})}{n(Cond)} - \frac{n(Target_{value})}{n_s}\right) \tag{6}$$

  - *True Positive Rate (TPR) or Sensitivity:* It measures the proportion of covered examples that has been correctly classified.

$$TPR(R) = \frac{n(Cond \wedge Target_{value})}{n(Target_{value})} \tag{7}$$

  - *False Positive Rate (FPR):* It measures the proportion of examples that are covered that do not belong to the target variable.

$$FPR(R) = \frac{n(Cond \wedge \overline{Target_{value}})}{n(\overline{Target_{value}})} \tag{8}$$

    where $\overline{Target_{value}}$ means the negation of $Target_{value}$, i.e., the examples that not belong to the target class.

A more extended classification and definition of quality measures for SD is available in [Herrera et al.(2011)] and [Atzmueller(2015)].

2.3 Evolutionary fuzzy systems

Evolutionary fuzzy systems (EFS) are the union of two powerful tools for approximate reasoning: evolutionary algorithms and fuzzy logic.

In one side, evolutionary algorithms [Eiben and Smith(2003)] are stochastic algorithms based on the natural evolution to solve complex optimization problems. It is based on a population of representations of possible solutions, called chromosomes. A basic evolutionary algorithm scheme is:

1. Generate the initial population
2. Evaluate the chromosomes of the population. This is the most important and expensive part of the GA. In the algorithms of this package, quality measures described above are used as evaluation function.
3. Select the chromosomes which the genetic operators will be applied.
4. Apply the genetic operators. The most used are:
   - Crossover operator. Which takes two chromosomes and generates two descendants as a combination of the elements of the parents.
   - Mutation operator. Which takes a chromosome and changes randomly a gene (a value of the possible solution).
5. Replace the population with the new generated chromosomes.
6. Go to step 2 until a stopping criteria is reached. Normally this criteria is a number of evaluations or generations.

These algorithms performs a global stochastic search through a huge search space efficiently. However, it is possible that these algorithms can not find an optimal solution (global optimum), but a good one (a local optimum) that can solve the problem too. Theys are well suited for SD because the problem of finding subgroups can be formulated from the optimization point of view coding rules as a parameters structure that optimize some measures. Additionally, different kinds of rules exists in SD (with inequality, with intervals, fuzzy rules...). This can change dramatically the size of the search space and genetic algorithms can adapt these structures easily without performance degradation. Likewise, the selection of the genetic operators can make GA a great candidate to introduce expert knowledge [Carmona et al.(2014)].

On the other side, fuzzy logic [Zadeh(1975)] is an extension of the traditional set theory. Its main objective is to model and deal with imprecise knowledge, which is nearer to the human reasoning. The main difference with traditional set theory is that belonging degree is not zero or one, but a real value in [0,1]. This possibility allows to define fuzzy limits and the chance of overlapping between fuzzy sets.

A fuzzy variable is a set of linguistic labels, e.g. low, medium and high, which are defined by overlapped fuzzy sets. This information is nearer to human reasoning and it is possible to calculate with precision the value of each belonging degree. This expressivity allow to obtain simpler rules because continuous variables are more understandable for humans. A rule can be a set of fuzzy variables. To determine if the rule cover an example it is neccessary to calculate the belonging degree of each variable in the rule with respect to the example. If all variables has a belonging degree grater than zero, the example is covered.

Evolutionary fuzzy systems is the union of both techniques, and has three ways of work [Herrera(2008)]:

- Evolutionary algorithms that evolve the fuzzy rules (changing the number of variables and their values) and uses fuzzy sets definitions defined by the user. This way of work is used by all the algorithms implemented in the SDEFSR package.
- Evolutionary algorithms that evolve the fuzzy sets, changing the number of fuzzy sets for each variable, its shapes, etc.
- Evolutionary algorithms that evolve both rules and fuzzy sets.

3 The SDEFSR package

SDEFSR is a package entirely written on R from scratch. This package includes the most important EFS for SD presented throughout the literature up to the moment. In addition, **SDEFSR** has the capacity to read data in different standard and well-known formats such as ARFF (Weka), KEEL, CSV and *data.frame* objects. Similarly, all functions included in the **SDEFSR** package have default parameters with values recommended by the authors. This allows the algorithms to be executed in an easy way, without the necessity of knowing the parameters for final users.

3.1 Algorithms of the package

Now, we describe the algorithms that are available in our package. This contains three algorithms: SDIGA [**?**], MESDIF [Berlanga et al.(2006)] and NMEEF-SD [Carmona et al.(2010a)]. In chapter 4 we describe

how to use the algorithms.

3.1.1 SDIGA (Subgroup Discovery Iterative Genetic Algorithm)

The algorithm SDIGA is a subgroup discovery algorithm that extract rules with two possible representations: one with conjunctions of attribute-value pairs (called canonical representation) or one with a disjunctive normal form (DNF) in the antecedent. It follows an iterative schema with a genetic algorithm in his core to extract those rules, this genetic algorithm only extract one rule, the best of the population, and after the extraction a local optimization could be performed in order to improve the generality of the rule. As the algorithm follows an iterative schema, the genetic algorithm is executed one time after another until a stopping criteria is reached: the rule obtained by the genetic algorithm and after the local improvement phase must cover at least one example not covered by precedent rules and this rule must have a minimum confidence (see Equation 4).
SDIGA can work with lost data (represented by the maximum value of the variable + 1), categorical variables and numerical ones using fuzzy logic with the latter to improve the interpretability of the results.

3.1.1.1 Components of genetic algorithm of SDIGA

As we mentioned above, a genetic algorithm is the core of SDIGA. Such genetic algorithm is the responsible of extract one rule per execution and this rule it is the best of the population at the final of the evolutive process.

3.1.1.1.1 Representation schema

Each chromosome in the population represents a possible rule but it only represents the antecedent part of the rule because the consequent is prefixed. SDIGA can handle two types of representation as we mentioned above, canonical and DNF.
Canonical representation is formed by a numerical vector of a fixed length equal to the number of variables with possibles values in a range in $[0, max]$ where max is the number of possible values for categorical variables or the number of fuzzy sets defined for numerical variables. This max value represents the no participation of that variable in the rule.
DNF representation is formed by a binary vector, also with fixed length equal to the sum of all number of values/fuzzy sets of the variables. Here, a variable does not participate in a rule when all of his values are equal to zero or one.

3.1.1.1.2 Crossover operator

SDIGA follows a pure stationary schema which only crosses the two best chromosomes in an iteration of the evolutionary process. This crossover is performed by a two-point cross operator.

3.1.1.1.3 Mutation operator

Mutation operator is applied over all the population of parents and chromosomes generated by crossover. The mutation probability is applied at every gene. This operator can be applied in two ways (both with the same probability):

- Eliminate the variable: it puts the no participation value in that variable.
- Put a random value: it puts a random value in that variable. The no participation value is also included.

3.1.1.1.4 Fitness function

The function to define the quality of a chromosome in SDIGA is a weighted average of three of the quality measures described in section 2.2. So the functions to maximize is:

$$f(x) = \frac{Sop(x) \cdot w_1 + Conf(x) \cdot w_2 + Obj3(x) \cdot w_3}{\sum_{i=1}^{3} w_i} \tag{9}$$

where:

- $Sop(x)$ is the local support. This support is a modification of support (see Equation 2) and can be crisp or fuzzy:
  - Crisp Support:

$$Sop_n(x) = \frac{Ne^+(R_i)}{Ne_{NC}} \tag{10}$$

– Fuzzy Support:

$$Sop_d(x) = \frac{\sum_k APC(E^k, R_i)}{Ne_{NC}} \tag{11}$$

where $Ne^+(R_i)$ is the number of examples covered by the rule and it is not covered by previous rules. Ne_{NC} is the number of examples of the target variable that it is not covered by any rule yet and APC is the *Antecedent Part Compatibility* equation calculted only with new covered examples (see Equation 13).

- $Conf(x)$ is the confidence defined as in Equation 4 for the crisp case, but for fuzzy case is defined as the ratio of the sum of APC expression of the correctly covered examples and the sum of APC all examples covered by the rule:

$$Conf_d(x) = \frac{\sum_{E^{CC}} APC(E^k, R_i)}{\sum_{E^C} APC(E^k, R_i)} \tag{12}$$

where E^{CC} are the correctly covered examples and E^C are the covered examples.

- $Obj3(x)$ is other quality measure of section 2.2
- w_i is the weight of objective i

As this rules uses fuzzy logic, we need an expresion to determine when an example is covered or not by a rule and also determine the belonging degree of that example to the rule. This function is determined by the expression APC (*Antecedent Part Compatibility*) and it is calculate by the expression:

$$APC(e, R_i) = T(TC(\mu_1^1(e_1), ..., \mu_n^1(e_i)), ..., TC(\mu_1^i(e_1), ..., \mu_n^i(e_i))) \tag{13}$$

where:

- e_i is the value of the example for the variable i
- μ_n^i is the belonging degree to the set n of the variable i
- TC is the fuzzy t-conorm. In this case is the maximum t-conorm.
- T is the fuzzy t-norm. In this case is the minimum t-norm.

μ_n^i will be one or zero if the variable is categorical and its value is the same of the rule or not. In case of numerical variable, μ_n^i will be computed following the triangular belonging degree function.

$$\mu_{a,b,c} = \begin{cases} 0 & x \leq a \\ \frac{x-a}{b-a} & a \leq x \leq b \\ \frac{c-x}{c-b} & b \leq x \leq c \\ 0 & c \leq x \end{cases} \tag{14}$$

3.1.1.1.5 Replace operator

To get next population for the next iteration a replace operator must be performed. This operator sort the population by fitness value and keep only the n best chromosomes, being n the size of the population.

3.1.1.2 Local optimization

After the genetic algorithm returns a rule, this rule could be improved by means of a local optimization based on a *hill climbing first best* local search. The algorithm eliminate one by one a variable and if the rule has a support and confidence greater than or equal the actual, the process starts again with that variable eliminated.

3.1.2 MESDIF (Multiobjetive Evolutionary Subgroup DIscovery Fuzzy rules)

MESDIF is a multi-objective genetic algorithm that extract fuzzy rules. The representation used could be canonical or DNF (see Section 3.1.1.1.1). This algorithm follows the SPEA2 [Zitzler et al.(2002)] approach where an elite population is used along the whole evolutive process. At the end of the process, the rules stored in the elite population where returned as result.
The multi-objective approach is based on a niches schema based on the dominance in the Pareto front. This schema allows to find non-dominated individuals to fill the elite population, that has a fixed size. In Figure 1 we see a basic schema of the algorithm.

3.1.2.1 Components of the genetic algorithm.

Now we describe briefly the components of the genetic algorithm to show how it works.

Step 1. Initialization:
Generate an initial population P_0 and create an empty elite population $P'_0 = \emptyset$. Set t = 0.
Repeat
Step 2. Fitness assignment: calculate fitness values of the individuals in P_t and P'_t.
Step 3. Environmental selection: copy all non-dominated individuals in Pt and P'_t to P'_{t+1}. As the size of P'_{t+1} must be exactly the number of individuals to store (N), we may have to use a truncation or a filling function.
Step 4. Mating selection: perform binary tournament selection with replacement on P'_{t+1} applying later crossover and mutation operators in order to fill the mating pool (obtaining P_{t+1}).
Step 5. Increment generation counter (t = t+1)
While stop condition is not verified.
Step 6. Return the non-dominated individuals in P'_{t+1}.

Figure 1: MESDIF operations schema

3.1.2.1.1 Initial Population generation

The initial population operator performed by MESDIF produce random chromosomes in two ways: one percentage of chromosomes are produced randomly and the rest are produced also randomly, but with the difference that only a maximum number of variables could participate in the rule. This produce more generality in the generated rules.

3.1.2.1.2 Fitness function

To obtain the fitness value of each chromosome, MESDIF follows this steps:

- First, we look for dominated and non-dominated individuals in both populations. We call *strength of an individual* the number of individuals that this domain.
- An initial fitness value A_i is calculated for each individual, such value is the sum of the strength of all dominators of individual i. So, we have to minimize this value and for non-dominated individuals is zero.
- Due to this system could fail if there are a lot of non-dominated individuals, additional information about density is included. This density is computed by the nearest neighbour approach and it is calculate by

$$D(i) = \frac{1}{\sigma^k + 2} \tag{15}$$

  where σ^k is the k-th nearest neighbour.
- The final adaptation value is the sum of initial fitness and density information

$$Fitness(i) = D(i) + A(i) \tag{16}$$

3.1.2.1.3 Truncation operator

As the elite population has a fixed size, we need a truncation operator to truncate the elite population if all non-dominated individuals can not fit in elite population. To make this truncation, the operator take all non-dominated individuals and calculate the distance among every one. Then, the two closest individuals are taken and eliminate the individual with his k-th nearest neighbour with a minor distance. This process is repeated until non-dominated indivuals fit in elite population.

3.1.2.1.4 Fill operator

If the number of non-dominated individuals are less than the size of the elite population, we need to fill elite population with dominated individuals. The operator sort the individuals by its fitness value and copy the n first individuals to elite population, where n is the size of the elite population.

3.1.2.1.5 Genetic operators

The genetic operators of MESDIF are:

- A two-point crossover operator (see Section 3.1.1.1.2)
- A biased mutation operator, the functionality is the same operator of SDIGA (see Section 3.1.1.1.3) but it is applied over a population of selected individuals.
- A selection operator based in a binary tournament selection. This selection is only applied over the individuals of elite population.
- A replacement of the selected population based on the direct replace of the k worse individuals of the population. k is the number of individuals returned after crosses and mutations.

3.1.3 NMEEF-SD (Non-dominated Multi-objective Evolutionary algorithm for Extracting Fuzzy rules in Subgroup Discovery)

NMEEF-SD is another multi-objective genetic algorithm based in the NSGA-II [Deb et al.(2000)] approach. This algorithm has a fast sorting algorithm and a reinitialisation based on coverage if the population does not evolve for a period.

This algorithm only works with a canonical representation (see Section 3.1.1.1.1). In [**?**] a study is presented where it reflects the low quality of rules obtained with a DNF representation.

3.1.3.1 Evolutionary process

The evolutionary process follows this steps

- An initial biased population P_t is generated (see Section 3.1.2.1.1).
- The genetic operators are applied over P_t obtaining Q_t with the same size as P_t.
- P_t and Q_t are joined obtaining R_t and the fast sorting algorithm is applied over R_t. The individuals are sorted forming different fronts in the following way: "The first front (F_1) is composed of non-dominated individuals, the second front (F_2) is composed by individuals dominated by one individual; the third front (F_3) is composed by individuals dominated by two, and so on."
- After that, the algorithm generates the population of the next generation (P_{t+1}). First, the algorithm checks if the Pareto front covers new examples as it can be show in Figure 2. If this condition is not satisfied during a period of the evolutionary process, a reinitialisation based on coverage is performed. Otherwise, the algorithm gets the next population (P_{t+1}) introducing, in order, the first complete fronts of R_t. If the last front does not fit completely in P_{t+1} then, the front is sorted by the *crowding distance* and first individuals are copied until P_{t+1} is filled.
- At the final of the evolutionary process the individuals in the Pareto front are returned.

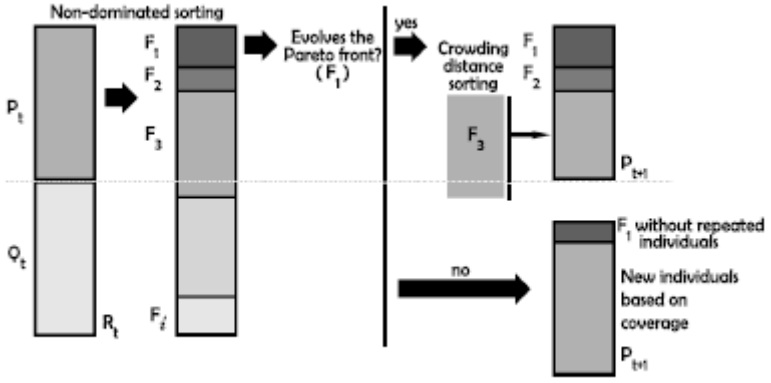


Figure 2: Operation schema of NMEEF-SD

3.1.3.2 Genetic operators

The genetic operators of NMEEF-SD are a two-point crossover operator (see Section 3.1.1.1.2) and a biased mutation operator (see Section 3.1.1.1.3).

3.1.3.3 Reinitialisation operator

This operator is applied if the Pareto front does not cover any new example during a 5% of the total number of evaluations. Then, the algorithm copy the non duplicated individuals in the Pareto front to P_{t+1} and the rest of individuals are generated by means of trying to cover one example of the target class with a maximum number of variables.

3.1.4 FuGePSD

FuGePSD [Carmona et al.(2015)] is another genetic algorithm that finds interesting subgroups of a given class. It uses a programming genetic approach, where individuals are represented as trees with variable length instead of vectors. Also, the consecuent of the rule is represented. This schema has the advantage of get rules for all possible values of a given target variable in one execution. Furthermore, FuGePSD has a variable population length, which can change over the evolutive process based on an competitive-cooperative approach, the Token Competition.

3.1.4.1 Evolutionary process

The evolutionary process of FuGePSD works as follow:

1. Create a random population of a fixed length.
2. Create a new population called Offspring Population, which is generated via genetic operators.
3. Join original population and offspring and execute the token competition procedure
4. Get global fitness and replace best population if necessary.
5. return to 2 until number of generations is reached.
6. Apply to the best population a Screening function and return the resulting rules.

The key function on this scheme is the token competition procedure, which promove diversity on the population, and some of the genetic operator will bring specificity to the rules.

3.1.4.2 Genetic operators

The genetic operators used by FuGePSD are:

- Crossover: it takes to parents and to random subtrees of each one. The crossover will cross the subtrees if the grammar are correct.
- Mutation: Change randomly a variable of an individual with a random value.
- Insertion: Inserts a new node on an individual, This node is a variable with a random value.
- Dropping: Remove a subtree of an individual.

This genetics operators will be applied with a probability given by the user.

3.1.4.3 Token Competition procedure

The token competition is the key procedure of FuGePSD, this brings diversity on the population keeping the best individuals. The algorithm works as follows: let a token be an example of the datasets, each individual can catch a token if the rule can cover it. If its occur, a flag is set at that token and this token cannot be catched by other rules.

So, the token competition works as follows:

1. Sort the population in descending order by their individual fitness value.
2. In order, each individual takes as much token as it can. This action is storied in a value for each individual called penalized fitness:

$$PenalizedFitness(R_i) = unusualness(R_i) * \frac{count(R_i)}{ideal(R_i)} \tag{17}$$

   Where $count(R_i)$ is the number of tokens the rule really seized and $ideal(R_i)$ is the maximum number of tokens the rule can seize.
3. Remove individuals with $PenalizedFitness = 0$.

3.1.4.4 Screening Function

At the end of the evolutive process, the screening function is launched to get the users quality rules only. This rules must reach a minimum level of confidence and sensitivity. The function has an external parameter (ALL_CLASS) which if true, the algorithm will return, at least, one rule per value of the target variable, or at least the best rule of the population if false.

3.2 Package architecture

The main advantage of the **SDEFSR** package is that it provides all evolutionary fuzzy systems for SD that exists in the bibliography. Algorithms included are an important kind of SD methods that are not available in R at the moment. Therefore, this package provides to the R community a brand new possibility for data mining and data analysis.

The base of the package is defined by two S3 classes. This classes are:

- *SDEFSR_Dataset*. This object defines a dataset and contains information about it. Such information are stored in the following fields:
  - *relation*. Defines the name of the relation that this dataset belongs to, e.g. "german credit".
  - *attributeNames*. Store the names of the attributes.
  - *attributeTypes*. A character that defines the data type of the attribute, i.e. 'r' for real values, 'e' for integers and 'c' for categorical variables.
  - *min*. A vector with the minimum value for numerical variables, for categorical variables, this value is zero.
  - *max*. A vector with the maximum value for numerical variables or the number of different categorical values that has a categorical variable.
  - *nVars*. The number of variables in the dataset (excluding the target class).
  - *data*. A list that contains each example of the dataset. The categorical variables in data are codified. This means that a categorical value is represented as a value in [0, max -1] that represents the position of the value. For example, in the *german-credit* dataset, the variable *ForeignWorker* has two values: "A201" and "A202", these values are represented in the *data* field of an *SDEFSR_Dataset* class as 0 or 1, respectively.
  - *class_names*. A vector with the categorical values of the target class. By default, the target variable is the last one. If it is not categorical, the methods that reads datasets fail. The user can select other target class when executing the algorithms.
  - *examplesPerClass*. A list with the number of examples that belongs to each class.
  - *lostData*. A logical that indicate the presence of lost data.
  - *covered*. A logical vector with length the number of examples that indicates which examples ar covered by the generated rules. This value by default is *NA*, but it is used in some algorithms like SDIGA.
  - *fuzzySets*. A list that indicates the fuzzy sets definitions for each variable. This value is *NA* by default and it is filled for each algorithm.
  - *crispSets*. A list that indicates the crisp sets obtained from the fuzzy sets. As this value is inferred from *fuzzySets*, this is NA too.
  - *sets*. A vector that defines the number of fuzzy sets that each variable has or the number of categorical values.
  - *categoricalValues*. A list that contains a vector of names of each categorical variable or *NA* if the variable is numerical.
  - *Ns*. The number of examples in the dataset.

  This class also exports the well-known S3 methods *print()* and *summary()* that show the data structure without codification and a summary with basic information about the dataset respectively.
- *SDEFSR_Rules*. This class is a list that contains the rules generated by an algorithm. To know the number of rules generated, it is possible to use *length(SDEFSR_RulesObject)*. Each rule has the following fields:
  - *rule*. The string that represent the rule description.
  - *qualityMeasures*. A list that contains the quality measures of the rule. Such measures are the same as described in the quality measures section.

  This object must be returned for all the SD algorithms of this package, and it is neccessary to make an analysis of the generated rules. This object is necessary to the exported functions *plot()* that plots an FPR vs TPR graph that allows the visualization of rules, and *orderRules()* that return other *SDEFSR_Rules* object with the rules sorted by a given quality measure in descendant order. Likewise, this object overloads the subset operator (*'['*) to allow filtering operations easily.

Additionally, the package has a general function that reads datasets in ARFF, KEEL or CSV format called *read.dataset()* and *SDEFSR_DatasetFromDataFrame()* to transform a *data.frame* into a *SDEFSR_Dataset*.

4 Use of SDEFSR package

In this section we are going to explain how to use this package. This package tries to use in a really simple way subgroup discovery algorithms and also without any dependencies.

4.1 Installing and load the package.

The package SDEFSR is now available at CRAN servers, so it can be installed as any other package by simply typing:

```
install.packages("SDEFSR")
```

Also, the develop version is available into GitHub at http://github.com/aklxao2/SDEFSR, feel free to clone and contribute if you wish. If you wish to use the development version you need to install the package `devtools` using the commando `install_github`:

```
devtools::install_github('aklxao2/SDEFSR')
```

SDEFSR depends only on the package **shiny** [**?**]. This package is neccessary to use the user interface in a local way and if the package is not installed, it asked to the user to be install when running the function `SDEFSRR_GUI()`. Once installed the package has to be loaded before use it. The package can be loaded through `library()` or `require()`. After loading the package there are six datasets available as *SDEFSR_Dataset*: `carTra`, `carTst`, `germanTra`, `germanTst`, `habermanTra` and `habermanTst` that corresponds to `car`, `german` and `haberman` training and test datasets. Also, rules generated by the SDIGA algorithm with defaults parameters over the `haberman` dataset are loaded as `habermanRules` as an *SDEFSR_Rules* object.

4.2 Loading a dataset

In order to use SD algorithms available in the **SDEFSR** package, a `SDEFSR_Dataset` object is neccessary. This object can be generated using the *read.dataset()* function. This function reads ".dat" files with the KEEL data mining tool format, ARFF files (".arff") from WEKA or even CSV files (".csv"). Assuming the files `iris.dat`, `iris.arff` and `iris.csv` corresponding to the classical iris dataset in KEEL, ARFF and CSV formats respectively in the working directory, the loading of these files will be as follows:

```
irisFromKEEL <- read.dataset("iris.dat")
irisFromARFF <- read.dataset("iris.arff")
irisFromCSV <- read.dataset("iris.csv")
```

Note that the function detects the type of data by the extension. To read csv files, the function has optional parameters that defines the separator used between fields, the decimal separator, the quote indicator and the `NA` identifier as parameters. By default, this options and values are `sep = ",", quote = "\"", dec = "."` and `na.strings = "?"` respectively. It is important to remark that sparse ARFF data is not supported.

If the dataset is not available in this formats, it is possible to obtain a *SDEFSR_Dataset* object from a `data.frame`. This `data.frame` could be loaded by `read.table()` or similar functions. Eventually, the resulting `data.frame` has to be given to the *SDEFSR_DatasetFromDataFrame()* function. As we can see, this function allows the creation of datasets on the fly, as in this example:

```
library(SDEFSR)
df <- data.frame(matrix(data = runif(1000), ncol = 10))
#Add class attribute (It must be the last attribute and it must be categorical)
df[,11] <- c("Class 0", "Class 1", "Class 2", "Class 3")
SDEFSR_DatasetObject <- SDEFSR_DatasetFromDataFrame(df, relation = "random")
#Load from iris dataset
irisFromDataFrame <- SDEFSR_DatasetFromDataFrame(iris, "iris")
```

This will assign to *SDEFSR_DatasetObject* a new dataset created randomly with 100 examples and 11 attributes. Note that the target variable must be categorical, because the SD algorithms can only deal with categorical target variables.\

The `SDEFSR\_DatasetFromDataFrame()` function has three additional parameters: `names`, `types`, and `classNames`. These allow the manual assignment of attribute names, their types, and a vector with values

of target variable, respectively. Leaving the default values (`NA`), the function automatically retrieves these values through the information found on the dataset. However, if the information in the dataset is not accurate, it could cause unexpected results for the SD algorithms.

4.3 Obtaining information from a loaded dataset

Once the dataset is loaded, it is possible to view a simple summary of its content by using the usual `summary()` function:

```
summary(irisFromDataFrame)
```

```
## Summary of the SDEFSR_Dataset object: 'irisFromDataFrame'
##  - relation: iris
##  - nVars: 4
##  - Ns: 150
##  - attributeNames: Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, Species
##  - class_names: setosa, versicolor, virginica
##  - examplesPerClass: 50, 50, 50
```

Any of these values can be obtained individually using the `$` operator on the `SDEFSR_Dataset` object:

```
irisFromDataFrame$nVars
```

```
## [1] 4
```

```
irisFromDataFrame$attributeNames
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
```

Also, it is possible to print the dataset as a `data.frame` with the `print()` function.

4.4 Executing Subgroup Discovery algorithms

Once our datasets are ready to be used, it is time to execute one subgroup discovery algorithm. For example we want to execute the algorithm MESDIF. For the rest of the algorithm this steps are equal and only a few parameters are different, if you need help with the parameters, refer to the help of the function using `help(function)` or `?function`.
It is possible to execute an SD algorithm in two ways: Through a parameters file, specifying as argument the path to such file. Or by entering all the parameter names and values at the command line. You can find the structure of the parameters file, among other useful information, on the help pages of each function.

Assuming the `SDEFSR_Dataset` object `irisFromDataFrame` that has been loaded before, and that the `params.txt` parameters file is stored in the working directory, the easiest way to run the `MESDIF()` algorithm, for example, will be: When we execute the algorithm, the results are shown in the console. This results are divided in three fields:

```
> ruleSet <- MESDIF(paramFile = "param.txt")
```

```
ruleSet <- MESDIF(paramFile = NULL,
                  training = irisFromDataFrame,
                  test = NULL,
                  output = c("optionsFile.txt", "rulesFile.txt", "testQM.txt"),
                  seed = 0,
                  nLabels = 3,
                  nEval = 300,
                  popLength = 100,
                  eliteLength = 2,
                  crossProb = 0.6,
                  mutProb = 0.01,
                  RulesRep = "can",
                  Obj1 = "CSUP",
```

```
                  Obj2 = "CCNF",
                  Obj3 = "null",
                  Obj4 = "null",
                  targetVariable = "Species",
                  targetClass = "virginica")
```

```
## --------------------------------
## Algorithm: MESDIF
## Relation: iris
## Training dataset: training
## Test dataset: test
## Rules Representation:  CAN
## Number of evaluations: 300
## Number of fuzzy partitions: 3
## Population Length: 100
## Elite Population Length: 2
## Crossover Probability: 0.6
## Mutation Probability: 0.01
## Obj1: CSUP   (Weigth:  )
## Obj2: CCNF   (Weigth:  )
## Obj3: null   (Weigth:  )
## Obj4: null
## Number of examples in training: 150
## Number of examples in test: 150
## Target Variable: Species
## Target Class: virginica
## --------------------------------
##
##
## Searching rules for only one value of the target class...
##
## GENERATED RULE 1
## Variable Sepal.Width = Label 1 ( 2 , 3.2 , 4.4 )
##  THEN virginica
##
## GENERATED RULE 2
## Variable Petal.Width = Label 2 ( 1.3 , 2.5 , 3.7 )
##  THEN virginica
##
##
## Testing rules...
## Rule 1 :
##   - N_vars: 2
##   - Coverage: 0.8
##   - Significance: 0.602743
##   - Unusualness: 0.02
##   - Accuracy: 0.357724
##   - CSupport: 0.286667
##   - FSupport: 0.245
##   - CConfidence: 0.358333
##   - FConfidence: 0.3528
##   - True Positive Rate: 0.86
##   - False Positive Rate: 0.77
```

```
## Rule 2 :
##   - N_vars: 2
##   - Coverage: 0.193333
##   - Significance: 27.673033
##   - Unusualness: 0.128889
##   - Accuracy: 0.9375
##   - CSupport: 0.193333
##   - FSupport: 0.201667
##   - CConfidence: 1
##   - FConfidence: 0.889706
##   - True Positive Rate: 0.58
##   - False Positive Rate: 0
## Global:TRUE
##   - N_rules: 2
##   - N_vars: 2
##   - Coverage: 0.496666
##   - Significance: 14.137888
##   - Unusualness: 0.074444
##   - Accuracy: 0.647612
##   - CSupport: 0.3
##   - FSupport: 0.223334
##   - FConfidence: 0.621253
##   - CConfidence: 0.679166
##   - True Positive Rate: 0.72
##   - False Positive Rate: 0.385
```

The output has three clearly defined sections:

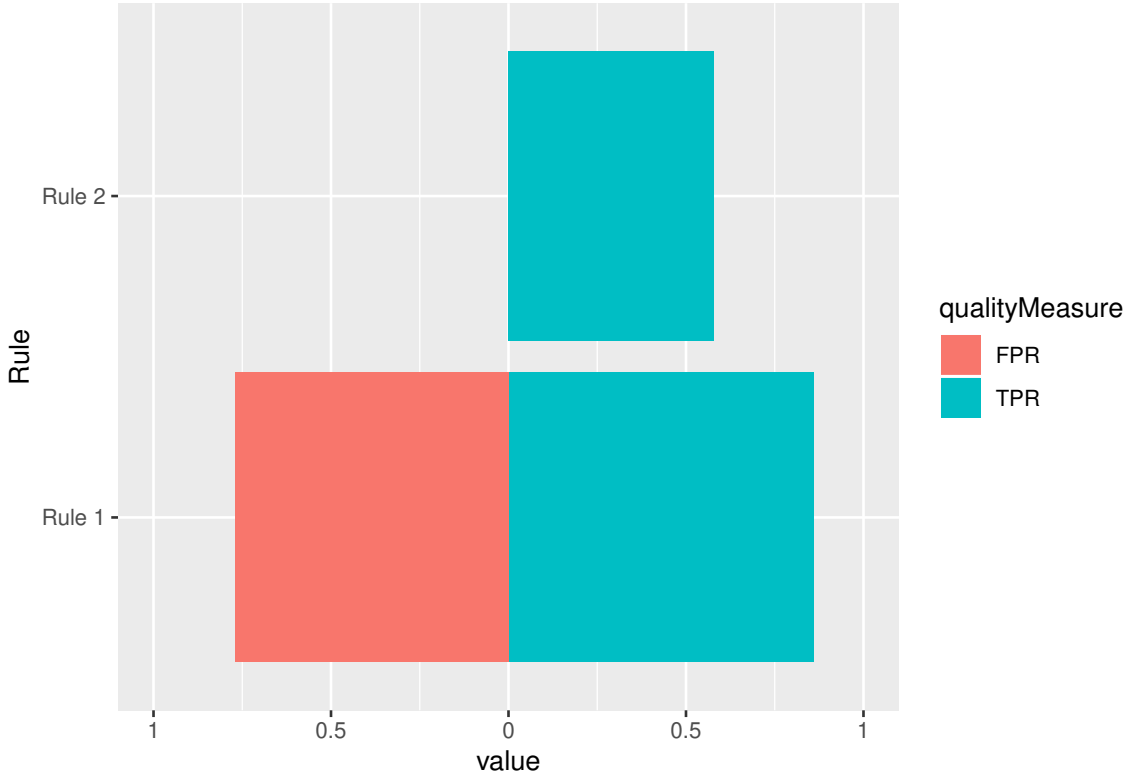
- The first one provides the user with information about the current execution, i.e., the values given to the parameters.
- After that, the rules obtained are shown one by one. These rules are numbered, starting at 1.
- Finally, the quality measures applied over the test (or training if 'test = NULL') dataset for each rule are shown. At the end of the results, the "Global" section shows quality measures as a summary, providing the mean of the quality measures of every rule.

As this output could be extremely large, the function also saves it to three files, one for each of the above sections. The name of these files by default are *optionsFile.txt*, *rulesFile.txt* and *testQM.txt* and being saved into the working directory, overwriting existing files. The format of these files is identical to the output generated by the algorithm, but divided into the sections described above.

4.5 Analyzing the rules obtained

After the execution of a SD algorithm, it returns a `SDEFSR_Rules` object that contains the rules obtained. Following the example, with the `ruleSet` object obtained we can plot a TPR vs FPR plot to view the reliability, and generality of the rule. Reliable rules has low values of FPR and great TPR values, and too much general variables has great values of both TPR and FPR. To plot the rules, we can use the function `plot()`:

```
library(ggplot2)
plot(ruleSet)
```



Additionally, we can directly order the rule set by a quality measure with the function `orderRules` which returns another `SDEFSR_Rules` object with the sorted rules. By default, the ordering is done by confidence.

```
    rulesOrderedBySignificance <- orderRules(ruleSet, by = "Significance")
```

Filter rules by number of attribute-value pairs or keep those rules with a quality measure greater than a given threshold is an interesting functionality to keep only high-quality rules. Such filtering operations are quite simple to apply in **SDEFSR**. Using the subset operator (`'[]'`) and introducing the filtering conditions will generate a new `SDEFSR_Rules` object with the result:

```
#Apply filter by unusualness
filteredRules <- ruleSet[Unusualness > 0.05]
#We check only if the number of rules decrease.
#In this case, this value must be 1.
length(filteredRules)
```

```
## [1] 1
```

```
#Also, you can make the filter as complex as you can
#Filter by Unusualness, TPr and number of variables:
filteredRules <- ruleSet[Unusualness > 0.05 & TPr > 0.9 & nVars == 3]
#In this case, any of the rules match the conditions. Therefore, the
#number of rules must be 0.
length(filteredRules)
```

```
## [1] 0
```

5 The user interface

As we mentioned at the begin of this paper, the **SDEFSR** package provide to the user an interface to use the subgroup discovery task in a more easy way and also do some basic exploratory analysis tasks. We can use the user interface by two ways: one of them is using the interface via web at: http://sdrinterface.shinyapps.io/shiny . This has the advantage of use the algorithm without having R installed

in our system and also avoid expending process time in our machine. The other way is to use the interface is in a local way, having our local server. This could be possible simply using:

```
SDR_GUI()
```

This function launch the user interface in our predetermined web browser. As we can see in Figure 3. The page are organized into an options panel at the left and a tabbed panel at the right. Here is where our results are shown when we execute the algorithms.
The first we have to do is to load a KEEL dataset. If we want to execute a subgroup discovery algorithm we must load a training and a test file **having the same `'@relation'` field in the KEEL dataset file.**
Once we select the dataset, automatically it shows a graph with information about the last variable defined in the dataset. The graph shows the distribution of examples having some values of the variable. At the rigth of this graphic we can see a table with some basic information, more extended if this variable is numerical.

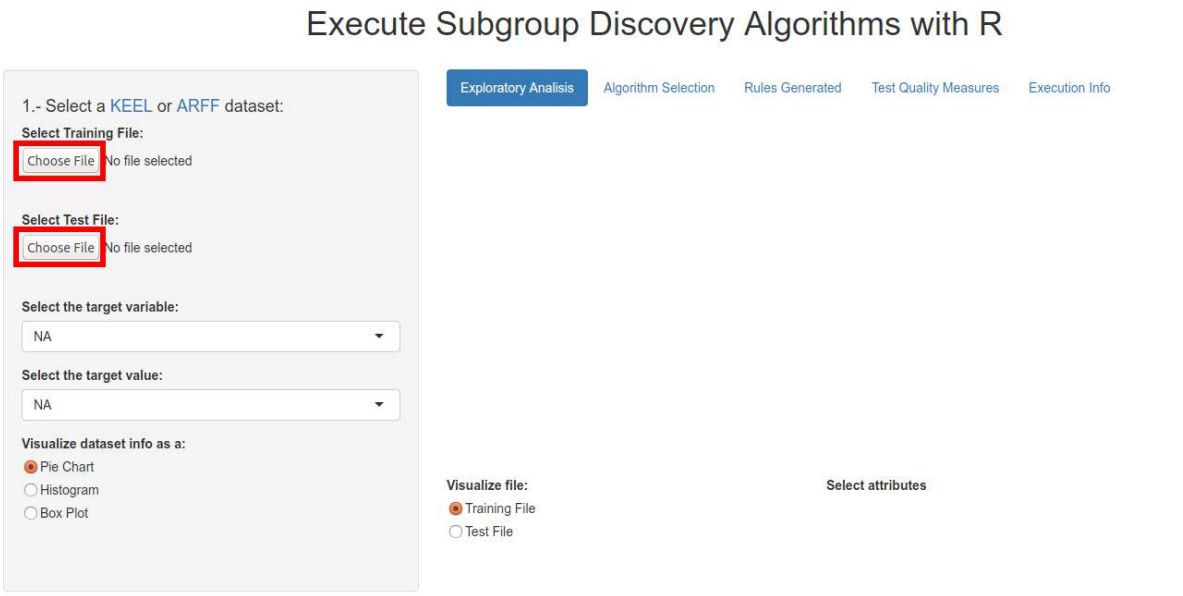


Figure 3: Initial screen of the SDEFSR user interface.

When loaded a dataset we can do a lot of things, in Figure 4 we can see all the possibilities we can do:

1. As we mentioned above, we can load a second dataset as a test (or training) file.
2. This lists have a double function. In one hand we could select the variable to visualize in the graph and in the other hand is to select the target variable for executing a subgroup discovery algorithm. Below the variable selection we can choose a target value of the variable to find subgroups about this value or search for all possible values of the target variable.
3. Here we can choose how the information will be visualized, for categorical variables we can choose show the information as a pie chart or as a histogram. For numerical variables, a histogram and a boxplot are available.
4. Here we can choose the subgroup discovery algorithm and his parameters, that it is shown below. Below the parameters, we have the button to execute the subgroup discovery algorithm.
5. This section allows the selection, for categorical variables, the values of that variable has that we can see in the graph. It is important to remark that it only "hide" the values in the graph, so it does not eliminate any example of the dataset.
6. It allows to visualize information about the training or test file.

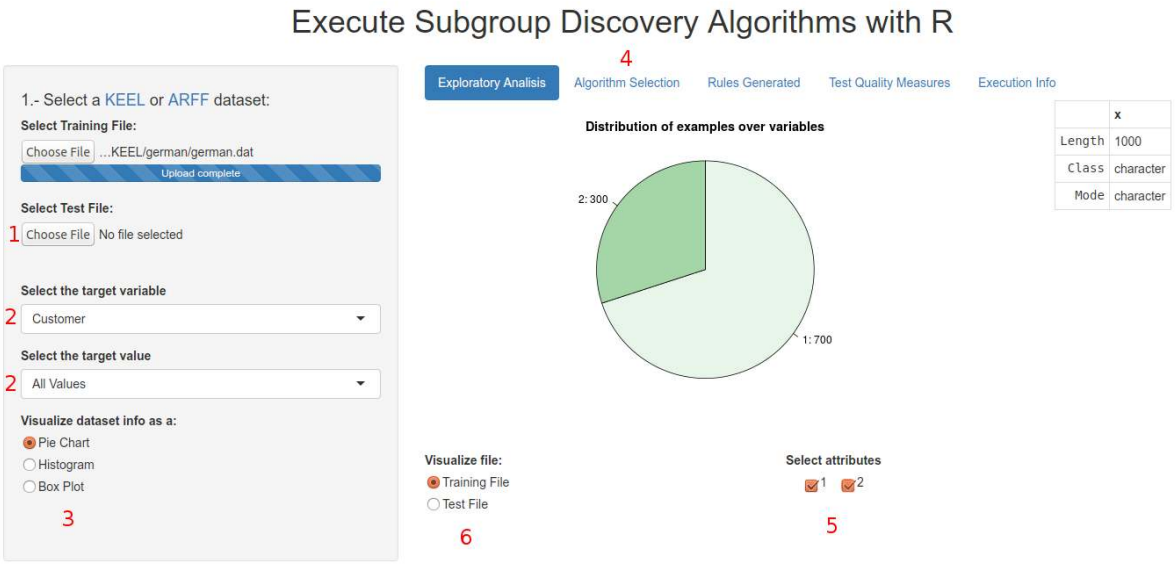


Figure 4: Screen of the user interface once loaded a dataset.

After the execution of a subgroup discovery algorithm, we go automatically to the tab `'Rules generated'`, this tab contains a table with all subgroups generated. If we want we could filter rules by variable, for example, typing into the `'Search'` field.
The tab `'Execution Info'` shows an echo of the parameters used for launch the algorithm. This echo is equal than the one we can see in R console.
The tab `'Test Quality Measures'` shows a table with quality measures of every rule applied to test dataset. The last row its a summary of results and shows the number of rules we have and the average results of every quality measure.

6 Summary

In this paper the **SDEFSR** package has been introduced. This package can use four subgroup discovery algorithms without any other dependencies to others tools/packages. Also, the possibility of read and load datasets in the KEEL dataset format is provided, but it can read dataset from ARFF format or load a data.frame object. Finally, a web-based interface is developed for make the work easier, even if we do not have R installed in our system.
The development of the package will continue in the future, including more functionality to work with

KEEL datasets, adding new subgroup discovery algorithms and also improve the web interface. As we can see we have a great job ahead, so we encourage other developers to participate adding tools or algorithms into the package, as we will do in futures releases.

References

[Alcalá-Fdez et al.(2011)] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic and Soft Computing*, 2-3(17): 255–287, 2011.

[Atzmueller(2014)] M. Atzmueller. *rsubgroup: Subgroup Discovery and Analytics*, 2014. URL http://CRAN.R-project.org/package=rsubgroup. R package version 0.6.

[Atzmueller(2015)] M. Atzmueller. Subgroup discovery. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 5(1):35–49, 2015.

[Atzmueller and Lemmerich(2012)] M. Atzmueller and F. Lemmerich. Vikamine–open-source subgroup discovery, pattern mining, and analytics. In *Machine Learning and Knowledge Discovery in Databases*, pages 842–845. Springer, 2012.

[Atzmueller and Puppe(2006)] M. Atzmueller and F. Puppe. Sd-map–a fast algorithm for exhaustive subgroup discovery. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 6–17. Springer, 2006.

[Atzmueller et al.(2004)] M. Atzmueller, F. Puppe, and H.-P. Buscher. Towards knowledge-intensive subgroup discovery. In *Proceedings of the Lernen - Wissensentdeckung - Adaptivität - Fachgruppe Maschinelles Lernen*, pages 111–117, 2004.

[Atzmueller et al.(2016)] M. Atzmueller, S. Doerfel, and F. Mitzlaff. Description-oriented community detection using exhaustive subgroup discovery. *Information Sciences*, 329:965–984, 2016.

[Back et al.(1997)] T. Back, D. B. Fogel, and Z. Michalewicz. *Handbook of evolutionary computation*. IOP Publishing Ltd., 1997.

[Berlanga et al.(2006)] F. Berlanga, M. J. Del Jesus, P. González, F. Herrera, and M. Mesonero. Multiobjective evolutionary induction of subgroup discovery fuzzy rules: a case study in marketing. *Advances in Data Mining. Applications in Medicine, Web Mining, Marketing, Image and Signal Mining*, pages 337–349, 2006.

[Carmona and Luengo(2015)] C. J. Carmona and J. Luengo. A first approach in the class noise filtering approaches for fuzzy subgroup discovery. *Advances in Intelligent Systems and Computing*, 368: 387–399, 2015.

[Carmona et al.(2009)] C. J. Carmona, P. González, M. J. del Jesus, and F. Herrera. An analysis of evolutionary algorithms with different types of fuzzy rules in subgroup discovery. In *Fuzzy Systems, 2009. FUZZ-IEEE 2009. IEEE International Conference on*, pages 1706–1711. IEEE, 2009.

[Carmona et al.(2010a)] C. J. Carmona, P. González, M. J. del Jesus, and F. Herrera. Nmeef-sd: non-dominated multiobjective evolutionary algorithm for extracting fuzzy rules in subgroup discovery. *IEEE Transactions on Fuzzy Systems*, 18(5):958–970, 2010a.

[Carmona et al.(2010b)] C. J. Carmona, P. González, M. J. del Jesus, C. Romero, and S. Ventura. Evolutionary algorithms for subgroup discovery applied to e-learning data. In *Proceedings of Education Engineering Conference (EDUCON)*, pages 983–990. IEEE, 2010b.

[Carmona et al.(2012)] C. J. Carmona, S. Ramirez-Gallego, F. Torres, E. Bernal, M. J. del Jesus, and S. Garcia. Web usage mining to improve the design of an e-commerce website: Orolivesur.com. *Expert Systems with Applications*, pages 11243–11249, 2012.

[Carmona et al.(2013)] C. J. Carmona, C. Chrysostomou, H. Seker, and M. J. del Jesus. Fuzzy rules for describing subgroups from influenza a virus using a multi-objective evolutionary algorithm. *Applied Soft Computing*, 13(8):3439–3448, 2013.

[Carmona et al.(2014)] C. J. Carmona, P. González, M. J. del Jesus, and F. Herrera. Overview on evolutionary subgroup discovery: analysis of the suitability and potential of the search performed by evolutionary algorithms. *WIREs Data Mining and Knowledge Discovery*, 4(2):87–103, 2014.

[Carmona et al.(2015)] C. J. Carmona, V. Ruiz-Rodado, M. J. del Jesus, A. Weber, M. Grootveld, P. González, and D. Elizondo. A fuzzy genetic programming-based algorithm for subgroup discovery and the application to one problem of pathogenesis of acute sore throat conditions in humans. *Information Sciences*, 298:180–197, 2015.

[Deb(2001)] K. Deb. Multi-objective optimization using evolutionary algorithms wiley. *Chichester, UK*, 2001.

[Deb et al.(2000)] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. *Lecture notes in computer science*, 1917: 849–858, 2000.

[del Jesus et al.(2007)] M. J. del Jesus, P. González, F. Herrera, and M. Mesonero. Evolutionary fuzzy rule induction process for subgroup discovery: a case study in marketing. *IEEE Transactions on Fuzzy Systems*, 15(4):578–592, 2007.

[Demšar et al.(2013)] J. Demšar, T. Curk, A. Erjavec, Č. Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič, et al. Orange: data mining toolbox in python. *The Journal of Machine Learning Research*, 14(1):2349–2353, 2013.

[Eiben and Smith(2003)] A. Eiben and J. Smith. Introduction to evolutionary algorithms, 2003.

[Fogel(2006)] D. B. Fogel. *Evolutionary computation: toward a new philosophy of machine intelligence*, volume 1. John Wiley & Sons, 2006.

[Freitas(2003)] A. A. Freitas. A survey of evolutionary algorithms for data mining and knowledge discovery. In *Advances in evolutionary computing*, pages 819–845. Springer, 2003.

[Gamberger and Lavrac(2002)] D. Gamberger and N. Lavrac. Expert-guided subgroup discovery: Methodology and application. *Journal of Artificial Intelligence Research*, pages 501–527, 2002.

[Gamberger et al.(2003)] D. Gamberger, N. Lavrač, and G. Krstačić. Active subgroup mining: A case study in coronary heart disease risk group detection. *Artificial Intelligence in Medicine*, 28(1):27–57, 2003. doi: 10.1016/S0933-3657(03)00034-4. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-0038500689&partnerID=40&md5=c145be9994977d3b34110e35c227cf31. cited By 46.

[Geng and Hamilton(2006)] L. Geng and H. J. Hamilton. Interestingness measures for data mining: A survey. *ACM Computing Surveys (CSUR)*, 38(3):9, 2006.

[Goldberg(1989)] D. Goldberg. Genetic algorithm in search, optimization and machine learing. *MA: Addison-Wesley Longman Publishing Co., Inc*, 1989.

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

[Herrera et al.(2011)] F. Herrera, C. J. Carmona, P. González, and M. J. del Jesus. An overview on subgroup discovery: foundations and applications. *Knowledge and information systems*, 29(3): 495–525, 2011.

[Hüllermeier(2005)] E. Hüllermeier. Fuzzy methods in machine learning and data mining: Status and prospects. *Fuzzy sets and Systems*, 156(3):387–406, 2005.

[Jin et al.(2014)] N. Jin, P. Flach, T. Wilcox, R. Sellman, J. Thumim, and A. Knobbe. Subgroup discovery in smart electricity meter data. *IEEE Transactions on Industrial Informatics*, 10(2):1327–1336, 2014.

[John(1992)] H. John. Adaptation in natural and artificial systems, 1992.

[Kavšek and Lavrač(2006)] B. Kavšek and N. Lavrač. Apriori-sd: Adapting association rule learning to subgroup discovery. *Applied Artificial Intelligence*, 20(7):543–583, 2006.

[Koza(1992)] J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[Lavrač et al.(2004a)] N. Lavrač, B. Cestnik, D. Gamberger, and P. Flach. Decision support through subgroup discovery: three case studies and the lessons learned. *Machine Learning*, 57(1-2):115–143, 2004a.

[Lavrač et al.(2004b)] N. Lavrač, B. Kavšek, P. Flach, and L. Todorovski. Subgroup discovery with cn2-sd. *The Journal of Machine Learning Research*, 5:153–188, 2004b.

[Lemmerich et al.(2011)] F. Lemmerich, M. Ifland, and F. Puppe. Identifying and presenting influence factors on student drop-outs by subgroup discovery. pages 129–132, 2011. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84874005031&partnerID=40&md5=35482d236cd80aad991d24fe681c313e. cited By 0.

[Leung et al.(1992)] K. S. Leung, Y. Leung, L. So, and K. F. Yam. Rule learning in expert systems using genetic algorithm: 1, concepts. In *Proceedings of the 2nd International Conference on Fuzzy Logic and Neural Networks (IIZUKA 92)*, volume 1, pages 201–204, 1992.

[Lowerre(1976)] B. T. Lowerre. *The Harpy speech recognition system*. PhD thesis, Carnegie-Mellon Univ., Pittsburgh, PA., 1976.

[Meeng and Knobbe(2011)] M. Meeng and A. Knobbe. Flexible enrichment with cortana–software demo. In *Proceedings of BeneLearn*, pages 117–119, 2011.

[Poitras et al.(2016)] E. Poitras, S. Lajoie, T. Doleck, and A. Jarrell. Subgroup discovery with user interaction data: An empirically guided approach to improving intelligent tutoring systems. *Educational Technology and Society*, 19(2):204–214, 2016. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84964491291&partnerID=40&md5=2c40f186d28100293654be5afba45ab4. cited By 1.

[Rodriguez et al.(2013)] D. Rodriguez, R. Ruiz, J. C. Riquelme, and R. Harrison. A study of subgroup discovery approaches for defect prediction. *Information and Software Technology*, 55(10):1810 – 1822, 2013.

[Schwefel(1995)] H.-P. Schwefel. Sixth-generation computer technology series, 1995.

[Stiglic and Kokol(2012)] G. Stiglic and P. Kokol. Discovering subgroups using descriptive models of adverse outcomes in medical care. *Methods of Information in Medicine*, 51(4):348–352, 2012. doi: 10.3414/ME11-02-0040. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84864857832&partnerID=40&md5=610901ec672dfe2b5c628c00a0476857. cited By 3.

[Wickham and Chang(2015)] H. Wickham and W. Chang. *devtools: Tools to Make Developing R Packages Easier*, 2015. URL http://CRAN.R-project.org/package=devtools. R package version 1.8.0.

[Wrobel(2001)] S. Wrobel. Inductive logic programming for knowledge discovery in databases. In *Relational data mining*, pages 74–101. Springer, 2001.

[Zadeh(1975)] L. A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning—i. *Information sciences*, 8(3):199–249, 1975.

[Zitzler et al.(2002)] E. Zitzler, M. Laumanns, and L. Thiele. Spea2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In *International Congress on Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, pages 95–100, 2002.