Using Subgroup Discovery algorithms in R: The **SDR** package

Angel M. Garcia

2016-01-15

Abstract

Subgroup Discovery is a data mining task between classification and description, this task is nowadays a relevant task for researchers due to it success in many fields, especially the medical field. The SDR package provide the posibility of use some of the algorithms that exists in the specialized bibliography by means of using datasets provided in the format specified by data mining tool KEEL without dependencies to other tools/packages in the R console. Also, the package provide a graphical interface to use the algorithms easily and to do basic exploratory analysis with datasets in KEEL format.

1 Introduction

Nowadays, a huge amount of information is generated everyday over the Internet. Such information contains implicit knowledge that is very important to many organizations because they give the possibility, for example, of improve their products or services.
In order to achieve this purpose, traditional techniques of knowledge extraction like *OLAP(On-Line Analitical Processing)* are not capable to extract knowledge as well as data mining do. Due to the success of data mining at extracting knowledge, a lot of techniques and methods have been designed. Such methods can be classified into two vast fields according to their final objective:

- Predictive data mining: which objective is to find a value of a predefined variable of interest in new instances that arrive at the system.
- Descriptive data mining: which objective is to find relationships between instances that are at the moment in the system.

This classification does not cover all the algorithms that exists in data mining, and subgroup discovery is one of those fields not covered by this classification because it has caracteristics of both fields. Now, R has in CRAN repository a package called *rsubgroup* [Martin Atzmueller, (2014)], this package is only an interface to use the subgroup discovery algorithms provided in VIKAMINE [M. Atzmueller, F. Lemmerich (2012)] data mining tool, and has a strong dependency with this tool and use the package *rJava* [Simon Urbanek (2013)] that also provided an interface to use Java files in R. So, this packages has a lot of external dependencies.

Our **SDR** package implements by now other subgroup discovery algorithms that is not available on VIKAMINE and those algorithms are implemented directly in R, so no dependencies with other tools/packages are necessary to perform a subgroup discovery task and the posibilities of growing the package by means of the adition of new algorithms is easier for every user of R (we have a public GitHub repository where everyone can contribute). Also our package provide a web interface available at http://sdrinterface.shinyapps.io and locally by calling the `SDR_GUI()` function. Such interface could, in one hand make a subgroup discovery task via web without having R installed in the system or in the other hand, make an easier task by using the visual controls of the web interface.
Further, our package provide a function to read KEEL [J. Alcala-Fdez et al.(2011)] datasets files, a dataset format that is not supported now in R.

2 Subgroup Discovery

Subgroup discovery could be defined as [Wrobel S. (2001)]:

> 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.

Subgroup discovery tries to find relations between different properties of a set with respect of a target variable, but this relations must be statistically interesting, so it is not necessary to find complete relations but partial relations.
The representation of thes relations is in form of rules, one per relation. This rule can be defined as:

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

Where $Target_{value}$ is the value for the variable of interest (target variable) for the subgroup discovery task and $Cond$ is normally a conjunction of attribute-value pairs which describe the characteristics of the subgroup induced.
Subgroup discovery is halfway between description and classification because subgroup discovery has a target variable but his objective is not predict but describe. Likewise the use of a target variable is not possible in description, because description find relations among all variables.
A key point to understand what subgroup discovery do is his difference with predictive data mining. Predictive data mining tries to split the entire search space in a normally complex way to find a value for a variable of interest in new incoming instances while subgroup discovery tries to find interesting relations between those instances to find what characteristics of those instances are interesting regarding the variable of interest.
An example could be a group of patients and our variable of interest is if those patients has or has not got heart disease. Predictive data mining objective is to predict if new patients will have a heart disease by their characteristcs while subgroup discovery tries to find what subgroup of patients are more likely to have a heart disease and with those characteristics, make a treatment against those characteristics.

2.1 Main elements of subgroup discovery algorithms

All the algorithms that are included into this data mining task must have the next characteristics:

- Type of target variable: The target variable could be binary (two posible values), categorical (n posible values) or numerical (a real value within a range).
- Description language: Rules must be as simple as possible. Due to this, rules are composed normally by conjunctions of attribute-value pairs or in disyuntive normal form. Also, to improve the interpretability of the results, fuzzy logic could be used.
- Quality measures: Are very important because they define when a subgroup is "interesting". They will be briefly described below.
- Search strategy: The search space grows exponentially by the number of variables. Using a search strategy that could find a good solution or the optimal one without searching into the whole search space is important.

2.2 Quality measures for subgroup discovery

A lot of quality measures have been defined for this task and there is no consensus for what quality measure it is the best for the task. The most important quality measures for subgroup discovery are described below[F. Herrera et al. (2010)]:

- N_r: The number of rules generated.
- N_v: The average number of variables that the rules generated have.
- **Support:** It measures the frequency of correctly classified examples covered by the rule:

$$Sup(x) = \frac{n(Cond \wedge T_v)}{n_s} \tag{2}$$

  Where $n(Cond \wedge T_v)$ means the number of correctly covered examples 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:

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

  where $n(Cond)$ means the number of covered examples by the rule.

- **Confidence:** It measure the percentage of examples correctly covered of the total of covered examples:

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

- **Interest:** It measures the interest of the rule determined by the antecedent and consequent:

$$Int(x) = \frac{\sum_{i=1}^{n_v} Gain(A_i)}{n_v \cdot log_2(|dom(G_k)|)} \tag{5}$$

where $Gain$ is the information gain, A_i is the number of values of the variable and $|dom(G_k)|$ is the cardinality of the target variable.

- **Significance:** It reflects the novelty in the distribution of the examples covered by the rule regarding the whole dataset:

$$Sign(x) = 2 \cdot \sum_{k=1}^{n_c} n(Cond \wedge T_{vk}) \cdot log(\frac{n(Cond \wedge T_{vk})}{n(Cond \wedge T_v) \cdot p(Cond)}) \tag{6}$$

where $p(Cond) = \frac{n(Cond)}{n_s}$

- **Unusualness:** It is defined as the weigthed relative accuracy of a rule

$$WRAcc(x) = \frac{n(Cond)}{n_s}(\frac{n(Cond \wedge T_v)}{n(Cond)} - \frac{n(T_v)}{n_s}) \tag{7}$$

where $n(T_v)$ is the number of examples that belong to the target variable.

3 Algorithms of the package

Now, we describe the algorithms that are available in our package. This contains three algorithms: SDIGA [M. del Jesus et al. (2007)], MESDIF [M. del Jesus et al. (2007)] and NMEEF-SD [C.J. Carmona et al. (2010)]. In chapter 4 we describe how to use the algorithms.

3.1 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 disyuntive 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 obteined 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 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 Representation schema

Each chromosome in the population represents a possible rule but it only represents the antecedent part of the rule because the consecuent 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.

3.1.1.2 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.

3.1.1.3 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.

3.1.1.4 Fitness function

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

$$f(x) = \frac{Sop(x) \cdot w_1 + Conf(x) \cdot w_2 + Obj3(x) \cdot w_3}{\sum_{i=1}^{3} w_i} \tag{8}$$

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:
  $$Sop_n(x) = \frac{Ne^+(R_i)}{Ne_{NC}} \tag{9}$$
  - Fuzzy Support:
  $$Sop_d(x) = \frac{\sum_k APC(E^k, R_i)}{Ne_{NC}} \tag{10}$$
  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 12).
- $Conf(x)$ 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 expresion of the correctly covered examples and the sum of APC all examples covered by the rule:
$$Conf_d(x) = \frac{\sum_{E^{CC}} APC(E^k, R_i)}{\sum_{E^C} APC(E^k, R_i)} \tag{11}$$
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 expresssion:

$$APC(e, R_i) = T(TC(\mu_1^1(e_1), ..., \mu_n^1(e_i)), ..., TC(\mu_1^i(e_1), ..., \mu_n^i(e_i))) \tag{12}$$

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 calculated following the triangular belonging degree function.

$$\mu_{a,b,c} = \begin{cases} 0 & x \leq a \\ \frac{x-a}{b-a} & a \leq x \leq b \\ \frac{c-x}{c-b} & b \leq x \leq c \\ 0 & c \leq x \end{cases} \tag{13}$$

3.1.1.5 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.

3.1.2 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.

3.2 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). This algorithm follows the SPEA2 [E. Zitzler et al. (2001)] 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.

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

3.2.1 Components of the genetic algorithm.

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

3.2.1.1 Initial Population generation

The initial population operator performed by MESDIF produce random chromosomes in two ways: one percentage of chromosomes are produced ramdomly 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.

3.2.1.2 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

$$D(i) = \frac{1}{\sigma^k + 2} \tag{14}$$

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

$$Fitness(i) = D(i) + A(i) \tag{15}$$

3.2.1.3 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.

3.2.1.4 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.

3.2.1.5 Genetic operators

The genetic operators of MESDIF are:

- A two-point crossover operator (see Section 3.1.1.2)
- A biased mutation operator, the functionality is the same operator of SDIGA (see Section 3.1.1.3) 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.

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

NMEEF-SD is another multi-objectve genetic algorithm based in the NSGA-II [K. 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). In [C. Carmona et al. (2009)] a study is presented where it reflects the low quality of rules obtained with a DNF representation.

3.3.1 Evolutionary process

The evolutionary process follows this steps

- An initial biased population P_t is generated (see Section 3.2.1.1).
- 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 individuales 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.

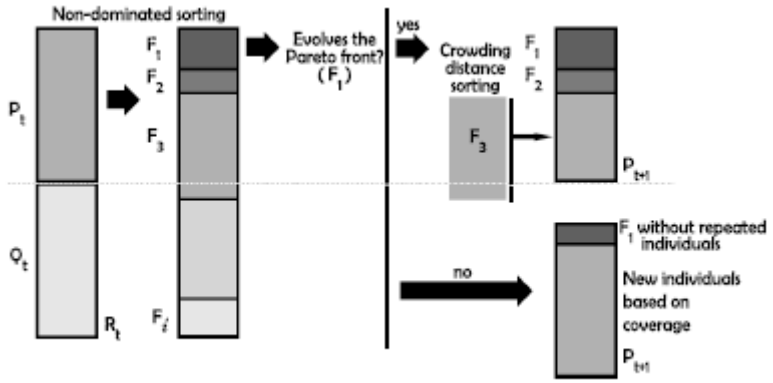

Figure 2: Operation schema of NMEEF-SD

3.3.2 Genetic operators

The genetic operators of NMEEF-SD are a two-point crossover operator (see Section 3.1.1.2) and a biased mutation operator (see Section 3.1.1.3).

3.3.3 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.

3.4 FuGePSD

FuGePSD [C. 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.

3.4.1 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 Propulation, which is generated via genetic operators.
3. Join original population and offspring and execute the token competition procedure
4. Get global fitnees 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 specifity to the rules.

3.4.2 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 indiviual with a random value.
- Insertion: Inserts a new node on an indivudual, 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.

3.4.3 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 setted at that token and this token cannot be catchet 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:

$$PenalizedFitness(R_i) = nusualness(R_i) * \frac{count(R_i)}{ideal(R_i)} \tag{16}$$

   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 $PenalizedFitness = 0$.

3.4.4 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.

4 Use of SDR 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 SDR is now available at CRAN servers, so it can be installed as any other package by simply typing:

```
install.packages("SDR")
```

Also, the develop version is available into GitHub at http://github.com/aklxao2/SDR, 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/SDR')
```

SDR depends only on the package **shiny** [Chang(2015)]. This package is neccesary 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 `SDR_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: `carTra`, `carTst`, `germanTra`, `germanTst`, `habermanTra` and `habermanTst` that corresponds to `car`, `german` and `haberman` training and test datasets.
This package use an internal representation assigned to the `"keel"` class . All the information about the datasets are available in this class.

4.2 Load a KEEL dataset

To use the algorithms available in this package, we could load a dataset if it is different of the three examples availables. Assuming the files `'irisTra.dat'` and `'irisTst.dat'`, corresponding to the classical iris dataset, the load of this files will be as follows:

```
irisTraining <- read.keel("irisTra.dat")
irisTest <- read.keel("irisTst.dat")
```

As we mentioned above, the algorithms in the package uses fuzzy logic, and the definitions of the fuzzy sets are implicit to every dataset. This fuzzy sets definitions are defined with the same length and the same type (all are triangular fuzzy sets). By default, the function `read.keel()` creates three fuzzy sets for every variable. To change the number of sets per variable, you can use the argument `'nLabels'`:

```
irisTraining <- read.keel("irisTra.dat", nLabels = 5)
irisTest <- read.keel("irisTst.dat", nLabels = 5)
```

If you want to get more KEEL datasets format, please visit: http://sci2s.ugr.es/keel/datasets.php
This function can read also ARFF the same way a KEEL file.

4.3 Creating a KEEL object from a data.frame

The other way to load a dataset on SDR is by a data.frame. This is possible by using the function `keelFromDataFrame()`:

```
irisTraining <- keelFromDataFrame(data = iris, relation = "iris",)
```

Also we can specify the number of fuzzy labels to by used, the names of the variables if it is not on the columns of the data.frame, the type of variables that has the dataset ('c' for categorical, 'r' for real and 'e' for integer) and a vector indicating the names of the target class

```
irisTraining <- keelFromDataFrame(data = iris, relation = "iris", nLabels = 5, names = c("Sepal.Length, Sepal
```

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 diferent, if you need help with the parameters, refer to the help of the function using `help(function)` or `?function`.
The subgroup discovery algorithms have two ways of execution: one of them is by indicating the path of a

parameters file, with all the neccesary parameters. The other is by putting in the function all parameters one by one. The first mode of use is indicated when we have a dataset prepared for executing directly because this way does not allow us to modify the properties of the dataset, so the second way is indicated when we load and/or modify a dataset before execute the algorithm.
When we execute the algorithm, the results are shown in the console. This results are divided in three fields:

- First, an echo of the parameters used in the execution are shown.
- Second, the subgroups or rules genereted by the algorithm.
- Finally, some quality measures about the rules generated applied to test data are shown. The final value are a summary of this quality measures for all rules

As the results line could be extremly large, the algorithms also save this results into three files, divided in the same categories described above.

```
MESDIF(paramFile = "MESDIFparameters.txt")
```

```
library("SDR")
MESDIF( paramFile = NULL,
        training = habermanTra,
        test = habermanTst,
        output = c("optionsFile.txt", "rulesFile.txt", "testQM.txt"),
        seed = 0,
        nLabels = 3,
        nEval = 300,
        popLength = 100,
        eliteLength = 3,
        crossProb = 0.6,
        mutProb = 0.01,
        RulesRep = "can",
        Obj1 = "CSUP",
        Obj2 = "CCNF",
        Obj3 = "null",
        Obj4 = "null",
        targetClass = "positive"
        )
```

```
## --------------------------------
## Algorithm: MESDIF
## Relation: haberman
## Training dataset: training
## Test dataset: test
## Rules Representation:  CAN
## Number of evaluations: 300
## Number of fuzzy partitions: 3
## Population Length: 100
## Elite Population Length: 3
## Crossover Probability: 0.6
## Mutation Probability: 0.01
## Obj1: CSUP   (Weigth: )
## Obj2: CCNF   (Weigth: )
## Obj3: null   (Weigth: )
## Obj4: null
## Number of examples in training: 244
## Number of examples in test: 62
## Target Variable: Survival
## Target Class: positive
## --------------------------------
##
##
```

```
##  Searching rules for only one value of the target class...
##
##
## GENERATED RULE 1
## Variable Age = Label 2 ( 56.5 , 83 , 109.5 )
## Variable Year = Label 0 ( 52.5 , 58 , 63.5 )
## Variable Positive = Label 0 ( -26 , 0 , 26 )
##
##  THEN positive
##
##
##
## GENERATED RULE 2
## Variable Positive = Label 0 ( -26 , 0 , 26 )
##  THEN positive
##
##
##
## GENERATED RULE 3
## Variable Year = Label 0 ( 52.5 , 58 , 63.5 )
## Variable Positive = Label 2 ( 26 , 52 , 78 )
##
##  THEN positive
##
##
##
##
##
##  Testing rules...
##
##
## Rule 1 :
##   - N_vars: 4
##   - Coverage: 0
##   - Significance: 0
##   - Unusualness: 0
##   - Accuracy: 0.5
##   - CSupport: 0
##   - FSupport: 0.002481
##   - CConfidence: 0
##   - FConfidence: 0.070144
##
##
## Rule 2 :
##   - N_vars: 2
##   - Coverage: 0.903226
##   - Significance: 0.223676
##   - Unusualness: -0.023413
##   - Accuracy: 0.241379
##   - CSupport: 0.209677
##   - FSupport: 0.186104
##   - CConfidence: 0.232143
##   - FConfidence: 0.218818
##
##
## Rule 3 :
##   - N_vars: 3
##   - Coverage: 0
##   - Significance: 0
```

```
##   - Unusualness: 0
##   - Accuracy: 0.5
##   - CSupport: 0
##   - FSupport: 0.004399
##   - CConfidence: 0
##   - FConfidence: 0.639344
##
##
## Global:
##   - N_rules: 3
##   - N_vars: 3
##   - Coverage: 0.301075
##   - Significance: 0.074559
##   - Unusualness: -0.007804
##   - Accuracy: 0.413793
##   - CSupport: 0.209677
##   - FSupport: 0.064328
##   - FConfidence: 0.309435
##   - CConfidence: 0.077381
```

5 The user interface

As we mentioned at the begin of this paper, the **SDR** 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.

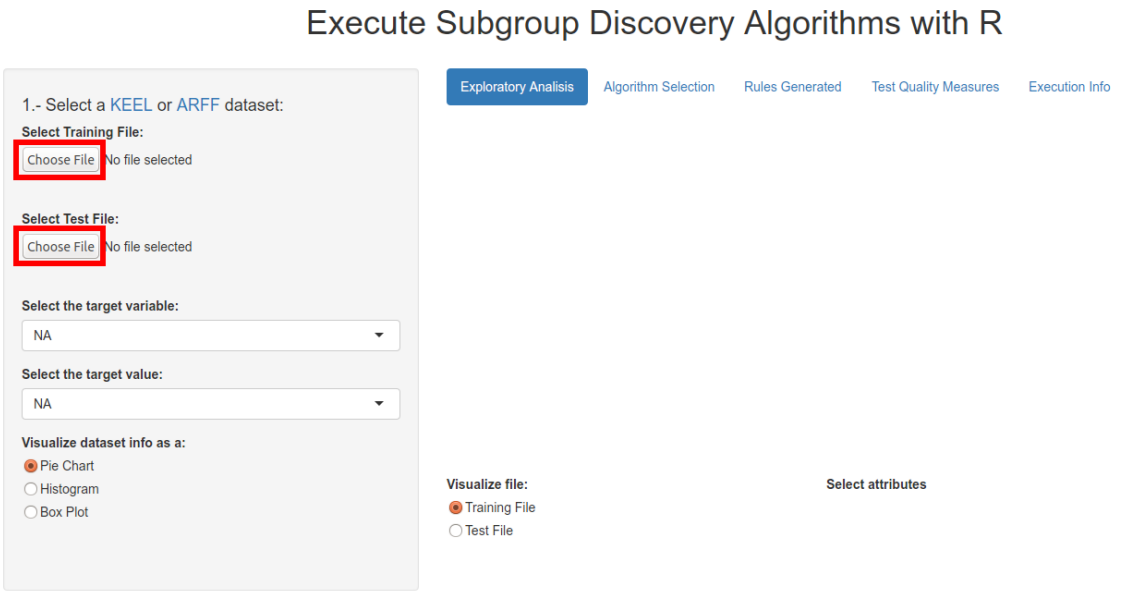

Figure 3: Initial screen of the SDR user interface.

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 posible 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 historgram. For numerical variables, a histogram and a bloxpot 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.

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 **SDR** package has been introduced. This package can use four subgroup discovery algorithms without any other dependencies to others tools/packages. Also, the posibility of read and load datasets in the KEEL dataset format is provided, but it can read dataset from ARFF format or load a

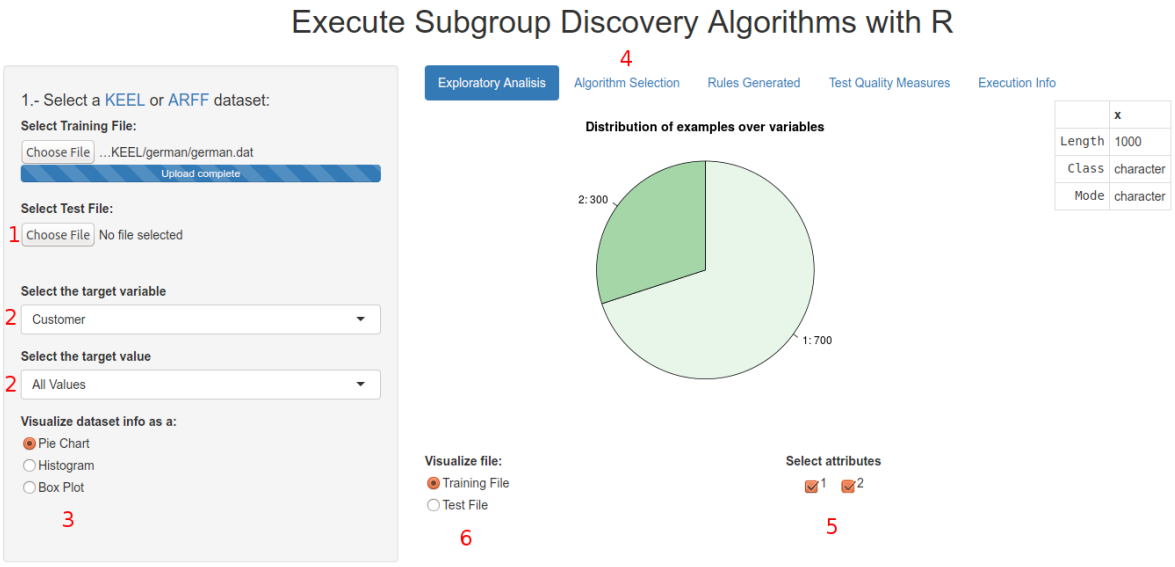

Figure 4: Screen of the user interface once loaded a dataset.

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 devolpment 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 participe adding tools or algorithms into the package, as we will do in futures releases.

References

[Wrobel S. (2001)] *Inductive logic programming for knowledge discovery in databases.* Springer, chap Relational Data Mining, pp 74-101.

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

[Simon Urbanek (2013)] *rJava: Low-level R to Java interface.* URL http://CRAN.R-project.org/package=rJava

[Chang(2015)] W. Chang. *shiny: Web Application Framework for R*, 2015. URL http://CRAN.R-project.org/package=shiny. R package version 0.11.

[M. Atzmueller, F. Lemmerich (2012)] *VIKAMINE - Open-Source Subgroup Discovery, Pattern Mining and Analytics* in *Machine Learning and Knowledge Discover in Databases* pp. 842-845

[J. Alcala-Fdez et al.(2011)] J. Alcala-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. Garcia, L. Sanchez, F. Herrera. *KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework.* In *Journal of Multiple-Valued Logic and Soft Computing 17:2-3 (2011) 255-287.*

[F. Herrera et al. (2010)] F. Herrera, M. J. del Jesus, P. Gonzalez, C. J. Carmona. *An overview on subgroup discovery: foundations and applications.* In *Knowledge and Information Systems.* December 2011, Volume 29, Issue 3, pp 495-525.

[M. del Jesus et al. (2007)] M. del Jesus, P. Gonzalez, F. Herrera, M. Mesonero. *Evolutionary fuzzy rule induction process for subgroup discovery: a case study in marketing.* In *IEEE Trans Fuzzy Syst.* pp. 15(4):578-592, 2007.

[C.J. Carmona et al. (2010)] C. J. Carmona, P. Gonzalez, M. J. del Jesus, F. Herrera. *NMEEF-SD: Non-dominated Multi-objective Evolutionary algorithm for Extracting Fuzzy rules in Subgroup Discovery.* In *Fuzzy Systems, IEEE Transactions.* Volume:18, Issue:5, pp.958-970, 2010

[M. del Jesus et al. (2007)] M. del Jesus, P. Gonzalez, F. Herrera. *Multiobjective genetic algorithm for extracting subgroup discovery fuzzy rules.* In *Proceedings of the IEEE symposium on computational intelligence in multi-criteria decision making.* pp. 50-57, 2007

[E. Zitzler et al. (2001)] E. Zitzler, M. Laumanns, L. Thiele. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm.* 2001

[K. Deb et al. (2000)] K. Deb, S. Agrawal, A. Pratap, T. Mayerivan. *A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II* 2000

[C. Carmona et al. (2009)] C. Carmona, P. Gonzalez , M. del Jesus, F. Herrera. *An analysis of evolutionary algorithms with different types of fuzzy rules in subgroup discovery.* In *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE).* ICC Jeju, Jeju Island, Korea, 2009

[C. Carmona et al. (2015)] C. Carmona, V. Ruiz-Rodado, M. del Jesus, A. Weber, M. Grootveld, P. Gonzalez, 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.* In *Information Sciences* Volume:298, pp. 180-197, 2015