Subgroup Discovery with Evolutionary Fuzzy Systems in R: The SDEFSR Package

by Ángel M. García, Francisco Charte, Pedro González, Cristóbal J. Carmona and María J. del Jesus

Abstract Subgroup discovery is a data mining task halfway between descriptive and predictive data mining. Nowadays it is very relevant for researchers 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 the 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 parameters of the algorithms.

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, 2013a; Stiglic and Kokol, 2012; Carmona et al., 2011; Gamberger et al., 2003) and e-learning (Poitras et al., 2016; Lemmerich et al., 2011; Carmona et al., 2010b), amongst others (Atzmueller et al., 2016; Jin et al., 2014; Carmona et al., 2013b; Rodriguez et al., 2013; Carmona et al., 2012).

SD is an useful rule learning process for complex search spaces. 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 optimising 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 use of rules to represent the knowledge is known as evolutionary rule-based systems (Freitas, 2003) and has the advantage of allowing the inclusion of domain knowledge, also returning better rules. The use of EAs is very well suited for SD because these algorithms perform a global search in the space in a convenient way, stimulating the obtaining of simple, interesting and precise subgroups.

Fuzzy logic is an extension of traditional set theory whose main aim is to model imprecise knowledge (Zadeh, 1975). The main element is the fuzzy set, which allows belonging degrees in the range [0,1] where zero means not belonging at all and one means absolute belonging. A ling??istic variable is a set of overlapped fuzzy sets which define ling??istic labels that cover all the range of a numeric variable. The main advantage of using fuzzy logic on SD is obtaining a knowledge representation for numeric variables more understandable for experts. It improves the interpretability of rules and the knowledge representation is very close to human reasoning (Hüllermeier, 2011). In addition, it avoids the possible loss of information in variables with continuous domains due to a previous discretisation stage.

Nowadays, there are several frameworks that allow one to perform data mining tasks, but only a few of them have implementations of SD algorithms. The best known 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) also provide some implementations. 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 (Alcalá-Fdez et al., 2011). This file format is not natively supported by R. Similarly, this package provides a Graphical User Interface (GUI) to make this task easier for the user, especially the unexperienced 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 in the last section.

Subgroup discovery

SD was defined by (Wrobel, 2001) as:

> *"In subgroup discovery, we assume we are given a so-called population of individuals (objects, customers, . . .) 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 relationships 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. A rule is 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 a data mining task halfway between description and classification, because it has a target variable but its objective is not to predict but rather to describe. The use of a target variable is not possible in description, because description simply finds relationships between unlabeled objects.

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 relationships 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 could be relevant to develop a treatment against this disease.

Main elements of subgroup discovery algorithms

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

- *Type of the 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 are 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.

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 emphasises 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 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 of the rules generated. Rules with less variables are easier to understand and to remember; also they tend to be more general. 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 generalise 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:

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

    where $n\left(Cond \wedge Target_{value}\right)$ 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:

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

    where $n\left(Cond\right)$ 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:

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

- 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:

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

    where $p\left(Cond\right) = \frac{n(Cond)}{n_s}$, n_c is the number of possible values of the target variable and $Target_{value_k}$ is the k-th value of the target variable.
- Hybrid measures. These measures try to maximise 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 maximise generality and realiability:

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

- *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.

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

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

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

Evolutionary fuzzy systems

Evolutionary fuzzy systems (EFSs) are the union of two powerful tools for aproximate reasoning: EAs and fuzzy logic.

In one side, EAs (Eiben and Smith, 2003) are stochastic algorithms based on the natural evolution to solve complex optimisation problems. They are based on a population of representations of possible solutions, called chromosomes. A basic EA scheme is:

1. Generate the initial population
2. Evaluate the chromosomes of the population. This is the most important and expensive part of the EA. 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 chromomes.
6. Go to step 2 until a stopping criterion is reached. Normally this criterion is a number of evaluations or generations.

These algorithms perform efficiently a global stochastic search through a huge search space. However, it is possible that these algorithms can not find an optimal solution (a global optimum), but a good one (a local optimum) that can solve the problem too. They are well suited for SD because the problem of finding subgroups can be formulated from the optimisation point of view as coding rules as a parameter structure that optimise some measures. Additionally, different kinds of rules exist in SD (with inequality, with intervals, fuzzy rules...). This can change dramatically the size of the search space and EAs can adapt these structures easily without performance degradation. Likewise, the selection of the genetic operators can make EAs great candidates to introduce expert knowledge in the search process (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. 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 allow one 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 (Hüllermeier, 2011). This information is closer to human reasoning and it is possible to calculate with precission the value of each belonging degree. This expressivity allows one to obtain simpler rules because continuous variables are more understandable for humans. A rule can be represented by means of a set of fuzzy variables. To determine if the rule covers an example it is neccesary to calculate the belonging degree of each variable in the rule with respect to the example. If all the variables have a belonging degree grater than zero, the example is covered.

EFSs are the union of both techniques, and work three ways(Herrera, 2008):

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

The SDEFSR package

SDEFSR is a package entirely written on R from scratch. To the best of our knowledge, this package includes all the EFSs for SD presented throughout the literature. 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.

Algorithms included in the package

SDEFSR implements the following SD algorithms (Ventura and Luna, 2016):

- SDIGA (del Jesus et al., 2007). This is a mono-objective EA (Back et al., 1997) based on an Iterative Rule Learning (IRL) approach in which only the best rule is extracted from an execution of the EA. This EA is executed iteratively until a stopping criterion is reached.
- MESDIF (Berlanga et al., 2006). A multi-objective EA (Deb, 2001) based on the SPEA2 (Zitzler et al., 2002) algorithm. It returns the best *n* rules (where *n* is an user parameter) in the pareto front.
- NMEEF-SD (Carmona et al., 2010a). Another multi-objective EA, based on the NSGA-II (Deb et al., 2000) algorithm which returns rules in the pareto front with a confidence greater than a given threshold. It has a reinitialisation operator to promote diversity and maximise the covering of all examples for target variable.
- FuGePSD (Carmona et al., 2015). An algorithm that uses genetic programming in which a competitive-cooperative scheme is implemented in order to obtain the best global rules. The key operation of this algorithm is the Token Competition (Leung et al., 1992), which promotes the extraction of precise, general and also diverse rules from the evolutionary process.

All these methods share the following characteristics:

- They use fuzzy logic to improve the interpretability of results, making them robust when working with noisy data (Luengo et al., (In Press) and also allowing one to include expert knowledge on the evolutionary learning process.
- Rules can be represented by a conjunction of attribute-value pairs (canonical form) or in disjunctive normal form (DNF). In (Carmona et al., 2009) there is an analysis justifying not using the DNF rule representation on some SD algorithms.
- It is possible to specify the quality measures used to guide the evolutionary process.
- All of the algorithms can deal with categorical (or multi-class) target variables.

Package architecture, extensibility, limitations and comparison with similar packages

The main advantage of the **SDEFSR** package is that it provides all EFSs for SD that exist in the literature. These algorithms are not included 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. These 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`. Stores 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 categoricals 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 a `"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 method that reads the dataset fails. The user can select other target class when executing the algorithms.
  - `examplesPerClass`. A list with the number of examples belonging to each class.
  - `lostData`. A logical that indicates the presence of lost data.
  - `covered`. A logical vector with length the number of examples that indicates which examples are 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.
  - `crispSets`. A list that indicates the crisp sets obtained from the fuzzy sets. As this value is infered from `fuzzySets`, this is NA by default.
  - `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 represents 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 neccesary to make an analysis of the generated rules. This object is necessary for the exported functions `plotRules()` that plots an FPR vs TPR graph that allows the visualisation of rules, and the well-known method `sort()` 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"`. In summary, exported functions and S3 objects are presented in Table 1.

A potential future extension of the package will be the inclusion of the confusion matrix for each rule. With this matrix it would be possible to infer the rest of the quality measures. Also, additional quality measures could be easily added. Statistical validation of the results are now implemented in package **rsubgroup**, which is already available on CRAN. Therefore, **SDEFSR** delegates this task to that package.

The **rsubgroup** package contains SD algorithms that can complement the algorithms available in the **SDEFSR** package. **rsubgroup** includes more established algorithms for SD like beam search (Lowerre, 1976) or SD-Map (Atzmueller and Puppe, 2006). This opens a wide range of SD algorithms that a user can execute in R. Nevertheless, both packages have great differences when calling SD

S3 Objects and methods	SD Algorithms	Other methods
"SDEFSR_Dataset"	SDIGA()	read.dataset()
print.SDEFSR_Dataset()	MESDIF()	SDEFSR_DatasetFromDataFrame()
summary.SDEFSR_Dataset()	NMEEF_SD()	plotRules()
"SDEFSR_Rules"	FUGEPSD()	
"[.SDEFSR_Rules"		
"sort.SDEFSR_Rules()"		

Table 1: Methods and rules exported by the **SDEFSR** package.

methods and the results. This means that a future package, which joins both into one standard framework would be interesting.

Below, the neccesary methodology to perform the inclusion of new algorithms in the SDEFSR package is shown:

1. The SD algorithm must use an "SDEFSR_Dataset" object as input and return an "SDEFSR_Rules" object with the results. Optionally, these results can be displayed on the console in a human-readable way.
2. The method can be executed with a parameter that contains the path of a parameter file. This file must contain the same parameters as the algorithm. This means that the algorithm must be executed either by a parameter file or by writting all neccesary parameters.
3. The majority of the parameters must have default values to ease the use of the algorithm.
4. The SD algorithm must not depend on other packages that are not in the "base" set of packages.
5. The source code of the algorithm must be added in a separate file with the name of such method.

As many others packages in R, this package does not have any automated test that control the quality of the code or results obtained. Instead, as the algorithms implemented exists in other platforms (KEEL), we check the quality of the methods comparing the results with the original implementation against a significant number of datasets with a 5-fold cross validation schema. This test showed that the results of the methods in **SDEFSR** are very close or equal to the results obtained from the reference implementations. Developers who want to add a new method to the package, must demonstrate the validity of the results obtained.

An example of use

This section describes an example of use of the package, covering the installation and loading of the package to the execution of a SD algorithm and the analysis of the rules generated.

Installing and loading the SDEFSR package

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

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

The development version is available on GitHub at https://github.com/SIMIDAT/SDEFSR. To install and use the development version you need to install the **devtools** (Wickham and Chang, 2015) package and then use the command:

```
> devtools::install_github("SIMIDAT/SDEFSR")
```

The package can be loaded using either the library() or require() functions. Once the package has been loaded, there are six sample datasets stored as "SDEFSR_Dataset" objects available: 'carTra', 'carTst', 'germanTra', 'germanTst', 'habermanTra' and 'habermanTst' that correspond to the 'car', 'german' and 'haberman' (Alcalá-Fdez et al., 2011) training and test datasets respectively. These are contained in the 'carTra.rda', 'carTst.rda', 'germanTra.rda', 'germanTst.rda', 'habermanTra.rda' and 'habermanTst.rda' files respectively, which are lazily loaded with the package. Also, rules generated by the SDIGA algorithm with default parameters over the 'haberman' dataset are loaded as 'habermanRules' as a "SDEFSR_Rules" object. These rules are stored in the 'habermanRules.rda' file.

Loading a dataset

In order to use SD algorithms available in the **SDEFSR** package, a `"SDEFSR_Dataset"` object is necessary. 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"). The source code for reading ARFF files has been taken from the **mldr** package(Charte and Charte, 2015). 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, these 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 these 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:

```
> 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")
```

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.

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(irisFromKEEL)
Summary of the SDEFSR_Dataset object: 'irisFromKEEL'
        - relation: iris
        - nVars: 4
        - Ns: 150
        - attributeNames: SepalLength, SepalWidth, PetalLength, PetalWidth, Class
        - class_names: Iris-setosa, Iris-versicolor, Iris-virginica
        - examplesPerClass: 50, 50, 50
```

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

```
> irisFromKEEL$nVars
[1] 4
> irisFromKEEL$attributeNames
[1] "SepalLength" "SepalWidth"  "PetalLength" "PetalWidth"  "Class"
```

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

Executing subgroup discovery algorithms

It is possible to execute a SD algorithm in two ways: through a parameter 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 parameter file, among other useful information, on the help pages of each function.

Assuming the "SDEFSR_Dataset" object 'irisFromKEEL' that has been loaded, and that the 'params.txt' parameter file is stored in the working directory, the easiest way to run the MESDIF() algorithm, for example, will be:

```
> ruleSet <- MESDIF(paramFile = "param.txt")
#or
> ruleSet <- MESDIF(training = irisFromKEEL)
```

The first way will execute the algorithm with the parameters and datasets defined in the parameter file. The second one will execute the algorithm with the specified "SDEFSR_Dataset" object, and default values for remainder parameters. By default, the target variable used is the last defined in the dataset and the algorithm searches rules for all the values of the target variable.

An example of an execution with all parameters and the result obtained could be:

```
> ruleSet <- MESDIF(paramFile = NULL, training = irisFromKEEL, 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 = "Class",
+               targetClass = "Iris-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: Class
Target Class: Iris-virginica
--------------------------------

Searching rules for only one value of the target class...

GENERATED RULE 1
Variable SepalWidth = Label 1 ( 2 , 3.2 , 4.4 )
THEN Iris-virginica

GENERATED RULE 2
Variable PetalWidth = Label 2 ( 1.3 , 2.5 , 3.7 )
THEN Iris-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: 0.351955
- 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:
- 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: 0.620831
- CConfidence: 0.679166
- True Positive Rate: 0.72
- False Positive Rate: 0.385
```

The output has three defined sections:

- The first one provides to the user 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 results, the "Global" section shows the average results for the quality measures analysed,

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.

The `output` parameter must be included if the authors desire the modification of the names of paths in the stored files. It accepts a vector with the names or paths to be used instead of the default ones. Additionally to this output, the function returns the "`SDEFSR_Rules`" object which contains the rules generated and the quality measures associated to each rule.

Analysing 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 (Kralj et al., 2005). Reliable rules have low values of FPR and high TPR values, and too general variables have high values for both TPR and FPR. To plot the rules, we can use the function `plotRules()`. (This function depends on the package **ggplot2**. If this is not installed, the user will be queried to install it.) The resulting plot is shown in Figure 1.

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

```
rulesOrderedBySignificance <- sort(x = ruleSet, decreasing = TRUE, by = "Significance")
```

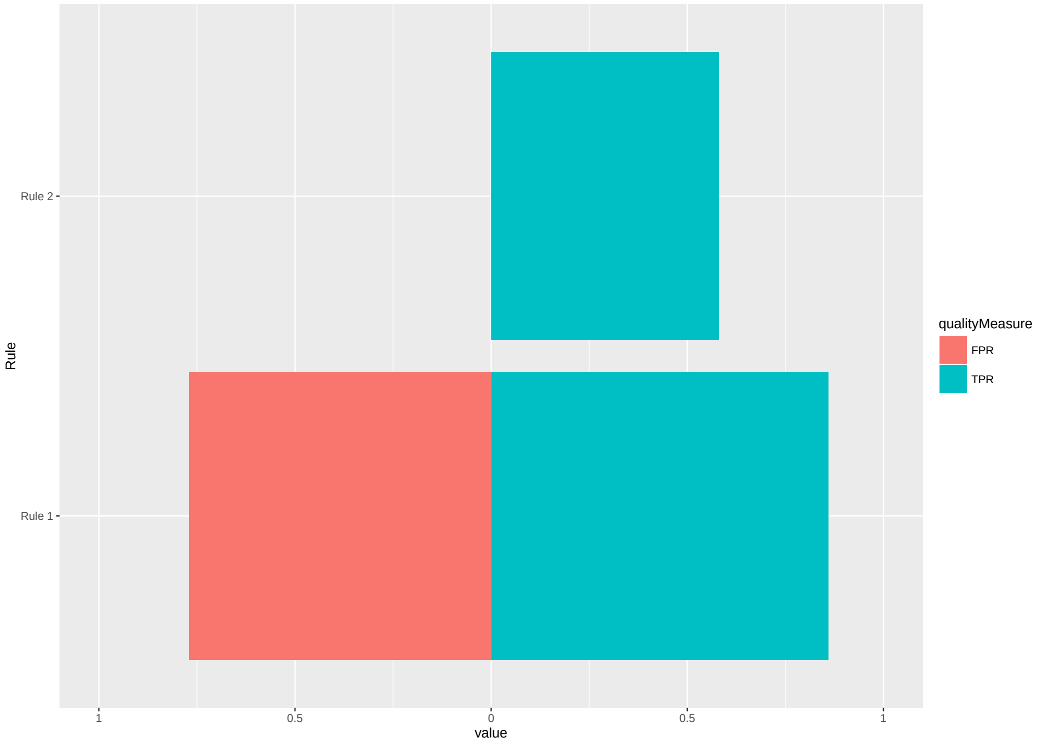


Figure 1: Plot generated after executing `plotRules()` on the "SDEFSR_Rules" object obtained in the example.

Filtering rules by number of attribute-value pairs or keeping those rules with a quality measure greater than a given threshold are interesting functionalities to extract 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, there are not rules that match the conditions. Therefore, the number of rules must be 0.

```
> length(filteredRules)
    [1] 0
```

The user interface

The **SDEFSR** package provides the user with a GUI to ease the use of SD algorithms. It also allows basic exploratory analisys of the data to be performed. This GUI is accessible by calling the function:

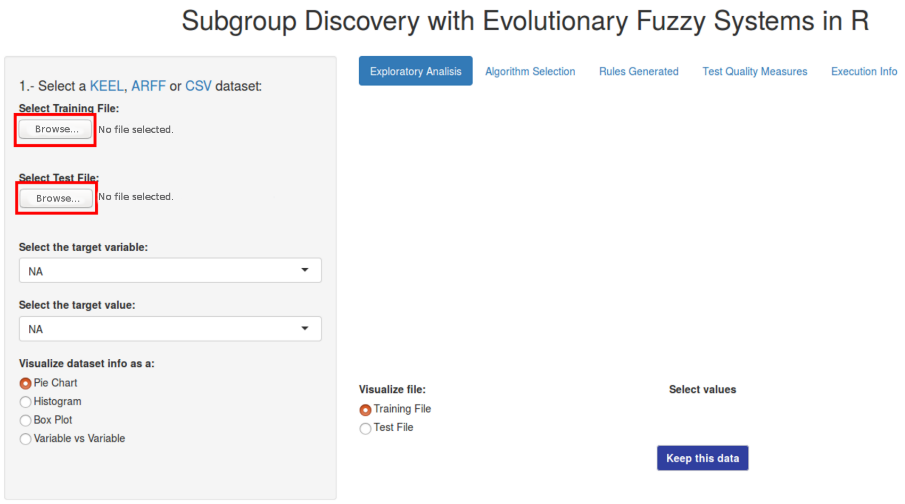
```
> SDEFSR_GUI()
```

The GUI was generated using the **shiny** package, therefore this function depends on this package. It depends on package **ggplot2** too. If **shiny** or **ggplot2** are not installed in the system, the user is given the option to install them.

```
> SDEFSR_GUI()
Package 'shiny' is not installed and must be installed to run this GUI.
Do you want to install it? (Y/n): Y
```

⋮

Once the package has been installed, the GUI is launched. In Figure 2 the initial state of the GUI is shown. It is structured into two areas. On the left the user can select the training and test files to be used, the target variable, and the value to be used by the SD algorithm as target variable. Finally, the group of radio buttons allows the user to change between different graphics to perform a basic exploratory analysis. On the right there is a tab panel where tabs are organised by the logical process of execution and visualisation of results of a SD algorithm.



1.- Select a KEEL, ARFF or CSV dataset:

Select Training File: No file selected.

Select Test File: No file selected.

Select the target variable: NA

Select the target value: NA

Visualize dataset info as a:
● Pie Chart
○ Histogram
○ Box Plot
○ Variable vs Variable

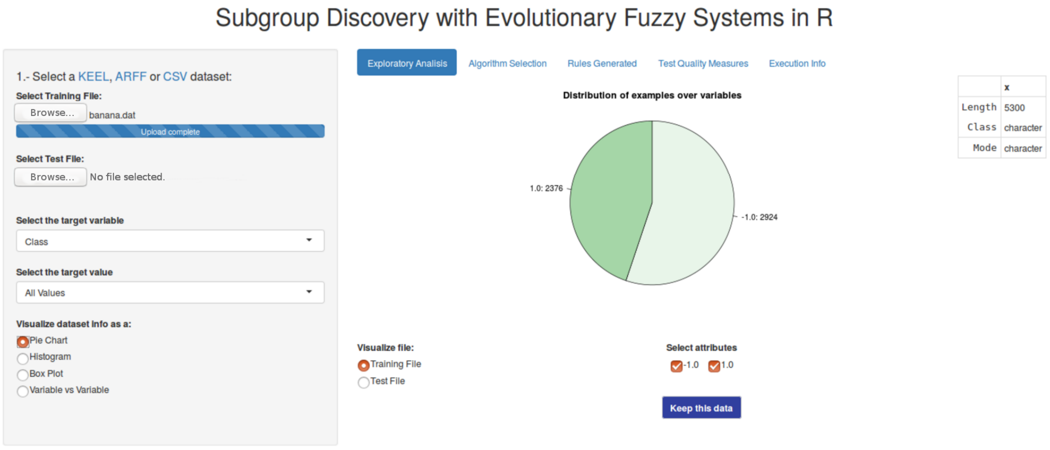
Visualize file:
● Training File
○ Test File

Select values

Figure 2: Initial screen of the SDEFSR user interface.

At this moment, the user only can perform the loading of datasets as a training or test file. The in Figure 2 within the red box intitiate the reading of files with the the same file formats as the `read.dataset()` function. Once a dataset has been loaded, an initial plot appears. This will be a pie chart if the last variable is categorical or a histogram otherwise. Figure 3 shows an example of such a graph. This graph could show information about all the variables in the dataset. For example, the pie chart shows the value and the number of examples that belongs to this value. To the right of this plot, a table provides some information about the distribution of the data samples. This becomes interesting with numerical variables where basic statistical information is displayed. To change the variable being visualised, use the "Select the target variable" dropdown menu.

The '`Keep this data`' button brings an interesting function. This button allows to filter examples whose values contain unselected values from the '`Select attributes`' field for categorical variables or values from numerical variables that are not within the range specified on the '`Show range`' slider.



1.- Select a KEEL, ARFF or CSV dataset:

Select Training File: banana.dat

Select Test File: No file selected.

Select the target variable: Class

Select the target value: All Values

Visualize dataset info as a:
● Pie Chart
○ Histogram
○ Box Plot
○ Variable vs Variable

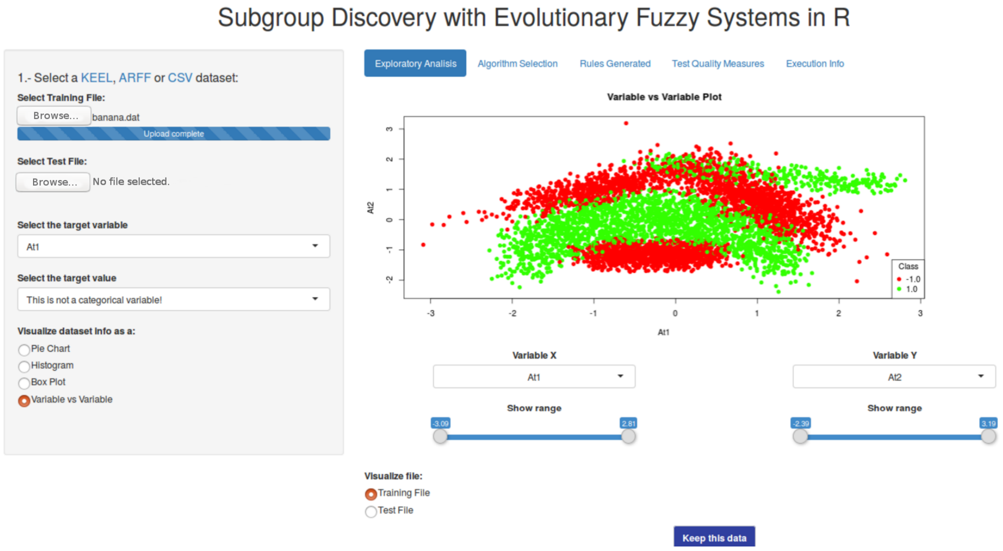
Visualize file:
● Training File
○ Test File

Select attributes
☑ -1.0 ☑ 1.0

	x
Length	5300
Class	character
Mode	character

Figure 3: Screenshot of the GUI once a dataset has been loaded.

Another functionality is the '`Variable vs Variable`' dataset visualisation. As shown in Figure 4 , it is possible to select two variables of the dataset and visualise their behaviour with respect to the target class. In this case, the target class is the last variable of the dataset. The plot shown depends on the types of variables choosen. If both are numerical, a scatter plot like in Figure 4 is shown and if both are categorical, a bar plot is shown. A numerical variable versus a categorical variable is an undefined functionality, thus, no plot is shown.



1.- Select a KEEL, ARFF or CSV dataset:

Select Training File: banana.dat

Select Test File: No file selected.

Select the target variable: At1

Select the target value: This is not a categorical variable!

Visualize dataset info as a:
○ Pie Chart
○ Histogram
○ Box Plot
● Variable vs Variable

Variable X: At1 — Show range: -3.09 to 2.81

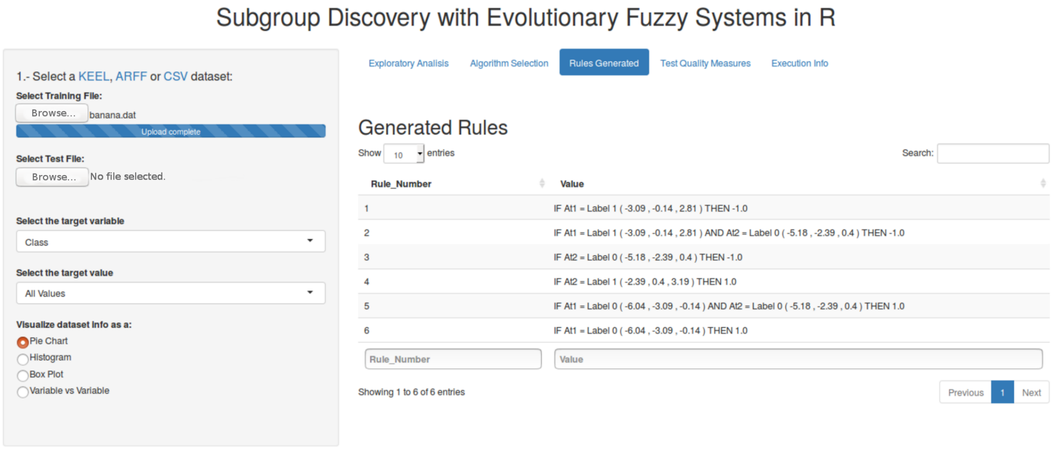
Variable Y: At2 — Show range: -2.39 to 3.19

Visualize file:
● Training File
○ Test File

Figure 4: Screenshot of the 'Variable vs Variable' functionality.

On the "Algorithm Selection" tab, the user can choose a SD algorithm to execute, and easily modify all the available parameters.

After the execution of a SD algorithm the "Rules Generated" tab is automatically selected as shown in Figure 5. Here the user can see the rules over a DataTable. The most important function is the "Search" field where the user can find rules with a specific variable. For example, with the results obtained executing MESDIF with the 'banana' (Alcalá-Fdez et al., 2011) dataset, which is an artificial dataset whose classes form a banana shape, and leaving the default parameters of the GUI. Typing "At1" on the search box filter rules with the variable At1 on the antecedent. Similarly, typing "THEN 1.0" filter rules with the value 1.0 on the consequent.



1.- Select a KEEL, ARFF or CSV dataset:

Select Training File: banana.dat

Select Test File: No file selected.

Select the target variable: Class

Select the target value: All Values

Visualize dataset info as a:
● Pie Chart
○ Histogram
○ Box Plot
○ Variable vs Variable

Generated Rules

Show 10 entries

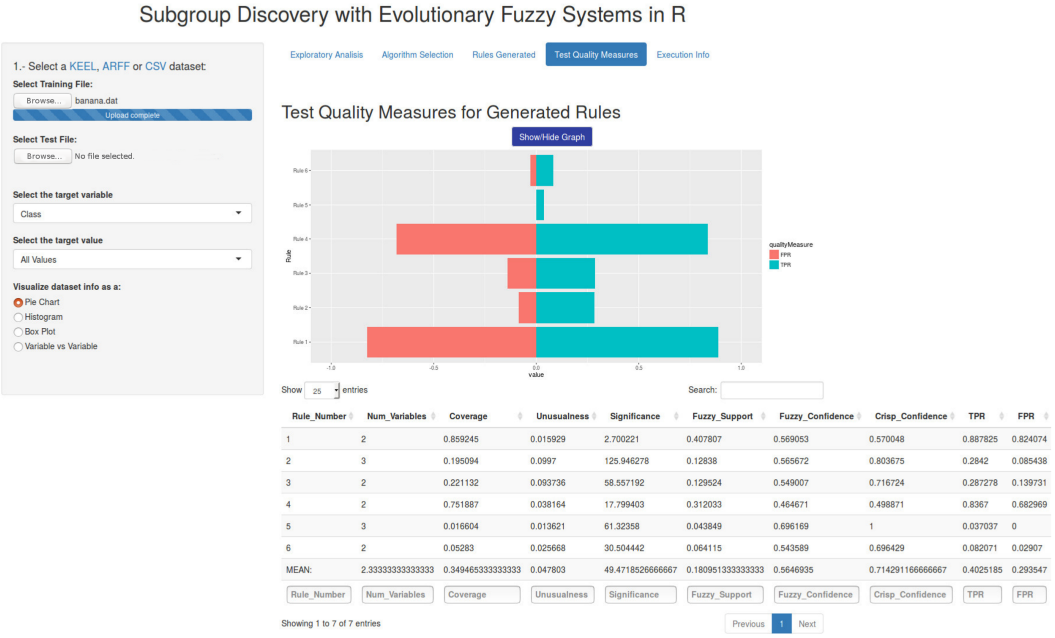
Rule_Number	Value
1	IF At1 = Label 1 (-3.09 , -0.14 , 2.81) THEN -1.0
2	IF At1 = Label 1 (-3.09 , -0.14 , 2.81) AND At2 = Label 0 (-5.18 , -2.39 , 0.4) THEN -1.0
3	IF At2 = Label 0 (-5.18 , -2.39 , 0.4) THEN -1.0
4	IF At2 = Label 1 (-2.39 , 0.4 , 3.19) THEN 1.0
5	IF At1 = Label 0 (-6.04 , -3.09 , -0.14) AND At2 = Label 0 (-5.18 , -2.39 , 0.4) THEN 1.0
6	IF At1 = Label 0 (-6.04 , -3.09 , -0.14) THEN 1.0
Rule_Number	Value

Showing 1 to 6 of 6 entries

Figure 5: Screen of the "Rules Generated" tab with generated rules.

Tab "Test Quality Measures" shows another DataTable with the quality measures for each rule. The filter options are similar to the DataTable of "Rules Generated". As shown in Figure 6, the button 'Show/Hide Graph' shows a FPR vs TPR plot of the generated rules, similar to the plotRules() function.

Finally, "Execution Info" shows information about the current execution, i.e., the parameters used in this execution. This information is like the execution information of a console execution (See Sec. 40.4.4).



1.- Select a KEEL, ARFF or CSV dataset:

Select Training File: banana.dat

Select Test File: No file selected.

Select the target variable: Class

Select the target value: All Values

Visualize dataset info as a:
● Pie Chart
○ Histogram
○ Box Plot
○ Variable vs Variable

Test Quality Measures for Generated Rules

Show 25 entries

Search:

Rule_Number	Num_Variables	Coverage	Unusualness	Significance	Fuzzy_Support	Fuzzy_Confidence	Crisp_Confidence	TPR	FPR
1	2	0.859245	0.015929	2.700221	0.407807	0.569053	0.570048	0.887825	0.824074
2	3	0.195094	0.0997	125.946278	0.12838	0.565672	0.803675	0.2842	0.085438
3	2	0.221132	0.093736	58.557192	0.129524	0.549007	0.716724	0.287278	0.139731
4	2	0.751887	0.038164	17.799403	0.312033	0.464671	0.498871	0.8367	0.682969
5	3	0.016604	0.013621	61.32358	0.043849	0.696169	1	0.037037	0
6	2	0.05283	0.025668	30.504442	0.064115	0.543589	0.696429	0.082071	0.02907
MEAN:	2.33333333333333	0.349465333333333	0.047803	49.4718526666667	0.180951333333333	0.5646935	0.714291166666667	0.4025185	0.293547
Rule_Number	Num_Variables	Coverage	Unusualness	Significance	Fuzzy_Support	Fuzzy_Confidence	Crisp_Confidence	TPR	FPR

Showing 1 to 7 of 7 entries

Figure 6: Screen of the "Quality Measures" tab with the FPR vs TPR graph displayed.

Summary

In this paper the **SDEFSR** package has been introduced. This package implements all the EFS-based algorithms for SD that exist in the specialised literature. The package can use datasets in ARFF, ".dat" of KEEL and CSV formats or a `data.frame` object. The main contribution of this package is the ease of use of the algorithms by means of functions with recommended parameters by default and different ways of execution, saving the user from the need to know the names of all the parameters of each algorithm. Also, a GUI is presented in order to make this task even easier.
The development of the package will continue in the future, including more functionality to work with datasets in more different formats, adding new SD algorithms, improving the performance of existing ones, and also bringing all this functionality to the GUI, which will be extended with more advanced tools for exploratory analysis.

Acknowledgments

This paper has been partially supported by the project TIN2015-68854-R (FEDER Founds) of the Spanish Ministry of Economy and Competitiveness.

The code of this package, available at `https://github.com/SIMIDAT/SDEFSR` is MIT-licensed.

Bibliography

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. [p307, 313, 319]

M. Atzmueller. *rsubgroup: Subgroup Discovery and Analytics*, 2014. URL `http://CRAN.R-project.org/package=rsubgroup`. R package version 0.6. [p307]

M. Atzmueller. Subgroup discovery. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 5(1):35–49, 2015. [p310]

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. [p307]

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. [p307, 312]

M. Atzmueller, F. Puppe, and H.-P. Buscher. Towards knowledge-intensive subgroup discovery. In *Proceedings of the Lernen - Wissensentdeckung - Adaptivit??t - Fachgruppe Maschinelles Lernen*, pages 111–117, 2004. [p308]

M. Atzmueller, S. Doerfel, and F. Mitzlaff. Description-oriented community detection using exhaustive subgroup discovery. *Information Sciences*, 329:965–984, 2016. [p307]

T. Back, D. B. Fogel, and Z. Michalewicz. *Handbook of evolutionary computation*. IOP Publishing Ltd., 1997. [p311]

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. [p307, 311]

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. [p311]

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. [p311]

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. [p307]

C. J. Carmona, P. González, M. Del Jesus, M. Navío-Acosta, and L. Jiménez-Trevino. Evolutionary fuzzy rule extraction for subgroup discovery in a psychiatric emergency department. *Soft Computing*, 15(12):2435–2448, 2011. [p307]

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. [p307]

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, 2013a. [p307]

C. J. Carmona, P. González, B. García-Domingo, M. Del Jesus, and J. Aguilera. Mefes: an evolutionary proposal for the detection of exceptions in subgroup discovery. an application to concentrating photovoltaic technology. *Knowledge-Based Systems*, 54:73–85, 2013b. [p307]

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. [p309, 310]

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. [p307, 311]

F. Charte and D. Charte. Working with multilabel datasets in R: The mldr package. *The R Journal*, 7 (2):149–162, dec 2015. URL `http://journal.r-project.org/archive/2015-2/charte-charte.pdf`. [p314]

K. Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001. [p311]

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. [p311]

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. [p307, 311]

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. [p307]

A. Eiben and J. Smith. Introduction to evolutionary algorithms, 2003. [p310]

D. B. Fogel. *Evolutionary computation: toward a new philosophy of machine intelligence*, volume 1. John Wiley & Sons, 2006. [p307]

A. A. Freitas. A survey of evolutionary algorithms for data mining and knowledge discovery. In *Advances in evolutionary computing*, pages 819–845. Springer, 2003. [p307]

D. Gamberger and N. Lavrac. Expert-guided subgroup discovery: Methodology and application. *Journal of Artificial Intelligence Research*, pages 501–527, 2002. [p308]

D. Gamberger, N. Lavra??, and G. Krsta??i?? Active subgroup mining: A case study in coronary heart disease risk group detection. *Artificial Intelligence in Medicine*, 28(1):27–57, 2003. [p307]

L. Geng and H. J. Hamilton. Interestingness measures for data mining: A survey. *ACM Computing Surveys (CSUR)*, 38(3):9, 2006. [p309]

D. Goldberg. Genetic algorithm in search, optimization and machine learing. *MA: Addison-Wesley Longman Publishing Co., Inc*, 1989. [p307]

F. Herrera. Genetic fuzzy systems: taxonomy, current research trends and prospects. *Evolutionary Intelligence*, 1(1):27–46, 2008. [p310]

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. [p310]

E. Hüllermeier. Fuzzy methods in machine learning and data mining: Status and prospects. *Fuzzy sets and Systems*, 156(3):387–406, 2005. [p308]

E. Hüllermeier. Fuzzy sets in machine learning and data mining. *Applied Soft Computing*, 11(2): 1493–1505, 2011. [p307, 310]

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. [p307]

H. John. Adaptation in natural and artificial systems, 1992. [p307]

B. Kavšek and N. Lavrač. Apriori-sd: Adapting association rule learning to subgroup discovery. *Applied Artificial Intelligence*, 20(7):543–583, 2006. [p307]

J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992. [p307]

P. Kralj, N. Lavrac, and B. Zupan. Subgroup visualization. In *8th International Multiconference Information Society (IS-05)*, pages 228–231, 2005. [p316]

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. [p308]

N. Lavrač, B. Kavšek, P. Flach, and L. Todorovski. Subgroup discovery with cn2-sd. *The Journal of Machine Learning Research*, 5:153–188, 2004b. [p307]

F. Lemmerich, M. Ifland, and F. Puppe. Identifying and presenting influence factors on student drop-outs by subgroup discovery. In *LWA 2011 - Technical Report of the Symposium "Lernen, Wissen, Adaptivitat - Learning, Knowledge, and Adaptivity 2011" of the GI Special Interest Groups KDML, IR and WM*, pages 129–132, 2011. [p307]

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. [p311]

B. T. Lowerre. *The Harpy speech recognition system*. PhD thesis, Carnegie-Mellon Univ., Pittsburgh, PA., 1976. [p312]

J. Luengo, A. Garc??a-Vico, M. D. P??rez-Godoy, and C. Carmona. The influence of noise on the evolutionary fuzzy systems for subgroup discovery. *Soft Computing*, pages 1–25, (In Press). [p311]

M. Meeng and A. Knobbe. Flexible enrichment with cortana–software demo. In *Proceedings of BeneLearn*, pages 117–119, 2011. [p307]

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. [p307]

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. [p307]

H.-P. Schwefel. Sixth-generation computer technology series, 1995. [p307]

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. [p307]

S. Ventura and J. M. Luna. Pattern mining with evolutionary algorithms, 2016. [p311]

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. [p313]

S. Wrobel. Inductive logic programming for knowledge discovery in databases. In *Relational data mining*, pages 74–101. Springer, 2001. [p308]

L. A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning???i. *Information sciences*, 8(3):199–249, 1975. [p307, 308, 310]

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. [p311]

Ángel M. García
Department of Computer Science. University of Jaén
Jaén
Spain
`agvico@ujaen.es`

Francisco Charte
Department of Computer Science and Artificial Intelligence. University of Granada
Granada
Spain
`fcharte@ugr.es`

Pedro González
Department of Computer Science. University of Jaén
Jaén
Spain
`pglez@ujaen.es`

Cristóbal J. Carmona
Department of Civil Engineering, Languages and Systems Area, University of Burgos.
Burgos
Spain
`cjcarmona@ubu.es`

María J. del Jesus
Department of Computer Science. University of Jaén
Jaén
Spain
`mjjesus@ujaen.es`