Package 'SDEFSR'

July 21, 2025

Type Package

Title Subgroup Discovery with Evolutionary Fuzzy Systems

Version 0.7.22

Date 2021-04-24

Maintainer Angel M. Garcia <agvico@ujaen.es>

Description Implementation of evolutionary fuzzy systems for the data mining task called ``subgroup discovery''. In particular, the algorithms presented in this package are:
M. J. del Jesus, P. Gonzalez, F. Herrera, M. Mesonero (2007) <doi:10.1109/TFUZZ.2006.890662>
M. J. del Jesus, P. Gonzalez, F. Herrera (2007) <doi:10.1109/MCDM.2007.369416>
C. J. Carmona, P. Gonzalez, M. J. del Jesus, F. Herrera (2010) <doi:10.1109/TFUZZ.2010.2060200>
C. J. Carmona, V. Ruiz-Rodado, M. J. del Jesus, A. Weber, M. Grootveld, P. González, D. Elizondo (2015) <doi:10.1016/j.ins.2014.11.030>
It also provide a Shiny App to ease the analysis. The algorithms work with data sets provided in KEEL, ARFF and CSV format and also with data.frame objects.

URL https://github.com/SIMIDAT/SDEFSR

Depends R (>= 3.0.0)

License LGPL (>= 3)

LazyData TRUE

Imports foreign, methods, parallel, stats, utils, ggplot2, shiny (>= 0.11)

Suggests knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.1.1

NeedsCompilation no

Author Angel M. Garcia [aut, cre],
Pedro Gonzalez [aut, cph],
Cristobal J. Carmona [aut, cph],
Francisco Charte [ctb],
Maria J. del Jesus [aut, cph]

Encoding UTF-8

Repository CRAN

Date/Publication 2021-04-30 06:50:14 UTC

Contents

as.data.frame.SDEFSR_Dataset . . . 2
carTra . . . 3
carTst . . . 4
FUGEPSD . . . 4
germanTra . . . 9
germanTst . . . 9
habermanRules . . . 10
habermanTra . . . 11
habermanTst . . . 11
MESDIF . . . 12
NMEEF_SD . . . 16
plot.SDEFSR_Rules . . . 21
print.SDEFSR_Dataset . . . 22
read.dataset . . . 23
SDEFSR . . . 24
SDEFSR_DatasetFromDataFrame . . . 25
SDEFSR_GUI . . . 26
SDIGA . . . 27
sort.SDEFSR_Rules . . . 32
summary.SDEFSR_Dataset . . . 33
[.SDEFSR_Rules . . . 33

Index **35**

`as.data.frame.SDEFSR_Dataset`

S3 function to convert into a data.frame the SDEFSR dataset This function converts a SDEFSR_Dataset object into a data.frame

---

Description

S3 function to convert into a data.frame the SDEFSR dataset

This function converts a SDEFSR_Dataset object into a data.frame

Usage

```
## S3 method for class 'SDEFSR_Dataset'
as.data.frame(x, ...)
```

Arguments

x	The SDEFSR_Dataset object to view
...	Additional arguments passed to the as.data.frame function

Details

Internally, a SDEFSR_Dataset object has a list of of examples and this examples are coded numerically. This function decode these examples and convert the list into a data.frame

Value

a data.frame with the dataset uncoded. Numeric attributes are "numeric" class, while categorical attributes are "factor"

@examples

as.data.frame(habermanTra)

---

carTra *Car evaluation dataset*

Description

Training data for the car dataset

Format

A SDEFSR_Dataset class with 1382 instances, 6 variables (without the target variable) and 4 values for the target Variable. Three labels for each variable are defined.

Details

Car Evaluation Database was derived from a simple hierarchical decision model. The model evaluates cars according to six input attributes: buying, maint, doors, persons, lug_boot, safety.

Source

M. Bohanec and V. Rajkovic: Knowledge acquisition and explanation for multi-attribute decision making. In 8th Intl Workshop on Expert Systems and their Applications, Avignon, France. pages 59-78, 1988.

B. Zupan, M. Bohanec, I. Bratko, J. Demsar: Machine learning by function decomposition. ICML-97, Nashville, TN. 1997 (to appear).

Examples

```
carTra$data
carTra$attributeNames
```

carTst *Car evaluation dataset*

Description

Test data for the car dataset

Format

A SDEFSR_Dataset class with 346 instances, 6 variables (without the target variable) and 4 values for the target Variable. Three labels for each variable are defined.

Details

Car Evaluation Database was derived from a simple hierarchical decision model. The model evaluates cars according to six input attributes: buying, maint, doors, persons, lug_boot, safety.

Source

M. Bohanec and V. Rajkovic: Knowledge acquisition and explanation for multi-attribute decision making. In 8th Intl Workshop on Expert Systems and their Applications, Avignon, France. pages 59-78, 1988.

B. Zupan, M. Bohanec, I. Bratko, J. Demsar: Machine learning by function decomposition. ICML-97, Nashville, TN. 1997 (to appear).

Examples

```
carTst$data
carTst$attributeNames
```

FUGEPSD *Fuzzy Genetic Programming-based learning for Subgroup Discovery (FuGePSD) Algorithm.*

Description

Make a subgroup discovery task using the FuGePSD algorithm.

Usage

```
FUGEPSD(
  paramFile = NULL,
  training = NULL,
  test = NULL,
  output = c("optionsFile.txt", "rulesFile.txt", "testQM.txt"),
  seed = 0,
  nLabels = 3,
  t_norm = "product_t-norm",
  ruleWeight = "Certainty_Factor",
  frm = "Normalized_Sum",
  numGenerations = 300,
  numberOfInitialRules = 100,
  crossProb = 0.5,
  mutProb = 0.2,
  insProb = 0.15,
  dropProb = 0.15,
  tournamentSize = 2,
  globalFitnessWeights = c(0.7, 0.1, 0.05, 0.2),
  minCnf = 0.6,
  ALL_CLASS = TRUE,
  targetVariable = NA
)
```

Arguments

paramFile	The path of the parameters file. NULL If you want to use training and test SDEFSR_Dataset variables
training	A SDEFSR_Dataset class variable with training data.
test	A SDEFSR_Dataset class variable with test data.
output	Character vector with the paths where store information file, rules file and test quality measures file, respectively. For rules and quality measures files, the algorithm generate 4 files, each one with the results of a given filter of fuzzy confidence.
seed	An integer to set the seed used for generate random numbers.
nLabels	Number of linguistic labels for numerical variables. By default 3. We recommend an odd number between 3 and 9.
t_norm	A string with the t-norm to use when computing the compatibilty degree of the rules. Use 'Minimum/Maximum' to specify the minimum t-norm, if not, we use product t-norm that is the default method.
ruleWeight	String with the method to calculate the rule weight. Possible values are: • Certainty_Factor: It uses the Classic Certainty Factor Weight method. • Average_Penalized_Certainty_Factor: It uses Penalized Certainty Factor weight II by Ishibuchi. • No_Weights: There are no weight calculation.

- Default: If none of this are specificied, the default method is Penalized Certainty Factor Weight IV by Ishibuchi.

`frm` A string specifying the Fuzzy Reasoning Method to use. Possible Values are:

- `Normalized_Sum`: It uses the Normalized Sum or Additive Combination Fuzzy Reasoning Method.
- `Arithmetic_Mean`: It uses the Arithmetic Mean Fuzzy Reasoning Method.
- Default: By default, Winning Rule Fuzzy Reasoning Method are selected.

`numGenerations` An integer to set the number of generations to perfom before stop the evolutionary process.

`numberOfInitialRules`
An integer to set the number individuals or rules in the initial population.

`crossProb` Sets the crossover probability. We recommend a number in [0,1].

`mutProb` Sets the mutation probability. We recommend a number in [0,1].

`insProb` Sets the insertion probability. We recommend a number in [0,1].

`dropProb` Sets the dropping probability. We recommend a number in [0,1].

`tournamentSize` Sets the number of individuals that will be chosen in the tournament selection procedure. This number must be greater than or equal to 2.

`globalFitnessWeights`
A numeric vector of length 4 specifying the weights used in the computation of the Global Fitness Parameter.

`minCnf` A value in [0,1] to filter rules with a minimum confidence

`ALL_CLASS` if TRUE, the algorithm returns, at least, the best rule for each target class, even if it does not pass the filters. If FALSE, it only returns, at least, the best rule if there are not rules that passes the filters.

`targetVariable` The name or index position of the target variable (or class). It must be a categorical one.

Details

This function sets as target variable the last one that appear in `SDEFSR_Dataset` object. If you want to change the target variable, you can set the `targetVariable` to change this target variable. The target variable MUST be categorical, if it is not, throws an error. Also, the default behaviour is to find rules for all possible values of the target varaible. `targetClass` sets a value of the target variable where the algorithm only finds rules about this value.

If you specify in `paramFile` something distinct to `NULL` the rest of the parameters are ignored and the algorithm tries to read the file specified. See "Parameters file structure" below if you want to use a parameters file.

@return The algorithm shows in console the following results:

1. Information about the parameters used in the algorithm.
2. Results for each filter:
   (a) Rules generated that passes the filter.
   (b) The test quality measures for each rule in that filter.

Also, this results are saved in a file with rules and other with the quality measures, one file per filter.

@section How does this algorithm work?: This algorithm performs a EFS based on a genetic programming algorithm. This algorithm starts with an initial population generated in a random manner where individuals are represented through the "chromosome = individual" approach includind both antecedent and consequent of the rule. The representation of the consequent has the advantage of getting rules for all target class with only one execution of the algorithm.

The algorithm employs a cooperative-competition approach were rules of the population cooperate and compete between them in order to obtain the optimal solution. So this algorithm performs to evaluation, one for individual rules to competition and other for the total population for cooperation.

The algorithm evolves generating an offspring population of the same size than initial generated by the application of the genetic operators over the main population. Once applied, both populations are joined a token competition is performed in order to maintain the diversity of the rules generated. Also, this token competition reduce the population sice deleting those rules that are not competitive.

After the evolutionary process a screening function is applied over the best population. This screening function filter the rules that have a minimum level of confidence and sensitivity. Those levels are 0.6 for sensitivy and four filters of 0.6, 0.7, 0.8 and 0.9 for fuzzy confidence are performed.

Also, the user can force the algorithm return at least one rule for all target class values, even if not pass the screening function. This behaviour is specified by the ALL_CLASS parameter.

@section Parameters file structure: The `paramFile` argument points to a file which has the neccesary parameters to execute FuGePSD. This file **must** be, at least, this parameters (separated by a carriage return):

- `algorithm` Specify the algorithm to execute. In this case. "MESDIF"
- `inputData` Specify two paths of KEEL files for training and test. In case of specify only the name of the file, the path will be the working directory.
- `seed` Sets the seed for the random number generator
- `nLabels` Sets the number of fuzzy labels to create when reading the files
- `nEval` Set the maximum number of **evaluations of rules** for stop the genetic process
- `popLength` Sets number of individuals of the main population
- `eliteLength` Sets number of individuals of the elite population. Must be less than `popLength`
- `crossProb` Crossover probability of the genetic algorithm. Value in [0,1]
- `mutProb` Mutation probability of the genetic algorithm. Value in [0,1]
- `Obj1` Sets the objetive number 1.
- `Obj2` Sets the objetive number 2.
- `Obj3` Sets the objetive number 3.
- `Obj4` Sets the objetive number 4.
- `RulesRep` Representation of each chromosome of the population. "can" for canonical representation. "dnf" for DNF representation.
- `targetClass` Value of the target variable to search for subgroups. The target variable **is always the last variable.** Use `null` to search for every value of the target variable

An example of parameter file could be:

```
algorithm = FUGEPSD
inputData = "banana-5-1tra.dat" "banana-5-1tst.dat"
outputData = "Parameters_INFO.txt" "Rules.txt" "TestMeasures.txt"
seed = 23783
Number of Labels = 3
T-norm/T-conorm for the Computation of the Compatibility Degree = Normalized_Sum
Rule Weight = Certainty_Factor
Fuzzy Reasoning Method = Normalized_Sum
Number of Generations = 300
Initial Number of Fuzzy Rules = 100
Crossover probability = 0.5
Mutation probability = 0.2
Insertion probability = 0.15
Dropping Condition probability = 0.15
Tournament Selection Size = 2
Global Fitness Weight 1 = 0.7
Global Fitness Weight 2 = 0.1
Global Fitness Weight 3 = 0.05
Global Fitness Weight 4 = 0.2
All Class = true
```

Author(s)

Written on R by Angel M. Garcia <amgv0009@red.ujaen.es>

References

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

Examples

```
FUGEPSD(training = habermanTra,
         test = habermanTst,
         output = c(NA, NA, NA),
         seed = 23783,
         nLabels = 3,
         t_norm = "Minimum/Maximum",
         ruleWeight = "Certainty_Factor",
         frm = "Normalized_Sum",
         numGenerations = 20,
         numberOfInitialRules = 15,
         crossProb = 0.5,
         mutProb = 0.2,
         insProb = 0.15,
         dropProb = 0.15,
         tournamentSize = 2,
         globalFitnessWeights = c(0.7, 0.1, 0.3, 0.2),
         ALL_CLASS = TRUE)
```

```
## Not run:
# Execution with a parameters file called 'ParamFile.txt' in the working directory:

FUGEPSD("ParamFile.txt")


## End(Not run)
```

---

germanTra *German Credit data set*

Description

Training data for the german dataset

Format

A SDEFSR_Dataset class with 800 instances, 20 variables (without the target variable) and 2 values for the target class.

Details

A numerical version of the Statlog German Credit Data data set. Here, the task is to classify customers as good (1) or bad (2), depending on 20 features about them and their bancary accounts.

Source

https://sci2s.ugr.es/keel/dataset.php?cod=88

Examples

```
germanTra$data
```

---

germanTst *German Credit data set*

Description

Test data for the german dataset

Format

A SDEFSR_Dataset class with 200 instances, 20 variables (without the target variable) and 2 values for the target class.

Details

A numerical version of the Statlog German Credit Data data set. Here, the task is to classify customers as good (1) or bad (2), depending on 20 features about them and their bancary accounts.

Source

https://sci2s.ugr.es/keel/dataset.php?cod=88

Examples

```
germanTra$data
```

`habermanRules` *Haberman survival rule set*

Description

Rules generated by the SDIGA algorithm with the default parameters for the `haberman` dataset.

Details

The rule set contains only two rules. One for each target variable

Source

Haberman, S. J. (1976). Generalized Residuals for Log-Linear Models, Proceedings of the 9th International Biometrics Conference, Boston, pp. 104-122.

Landwehr, J. M., Pregibon, D., and Shoemaker, A. C. (1984), Graphical Models for Assessing Logistic Regression Models (with discussion), Journal of the American Statistical Association 79: 61-83.

Lo, W.-D. (1993). Logistic Regression Trees, PhD thesis, Department of Statistics, University of Wisconsin, Madison, WI.

Examples

```
habermanRules
```

habermanTra *Haberman survival data set*

Description

Training data for the Haberman dataset.

Format

A SDEFSR_Dataset class with 306 instances, 3 variables (without the target variable) and 2 values for the target variable. Three fuzzy labels for each numerical variable are defined.

Details

This data set contains cases from a study that was conducted between 1958 and 1970 at the University of Chicago's Billings Hospital on the survival of patients who had undergone surgery for breast cancer. The task is to determine if the patient survived 5 years or longer (positive) or if the patient died within 5 year (negative)

Source

Haberman, S. J. (1976). Generalized Residuals for Log-Linear Models, Proceedings of the 9th International Biometrics Conference, Boston, pp. 104-122.

Landwehr, J. M., Pregibon, D., and Shoemaker, A. C. (1984), Graphical Models for Assessing Logistic Regression Models (with discussion), Journal of the American Statistical Association 79: 61-83.

Lo, W.-D. (1993). Logistic Regression Trees, PhD thesis, Department of Statistics, University of Wisconsin, Madison, WI.

Examples

```
habermanTra$data
habermanTra$fuzzySets
```

habermanTst *Haberman survival data set*

Description

Test data for the Haberman dataset.

Format

A SDEFSR_Dataset class with 62 instances, 3 variables (without the target variable) and 2 values for the target variable. Three fuzzy labels for each numerical variable are defined.

Details

This data set contains cases from a study that was conducted between 1958 and 1970 at the University of Chicago's Billings Hospital on the survival of patients who had undergone surgery for breast cancer. The task is to determine if the patient survived 5 years or longer (positive) or if the patient died within 5 year (negative)

Source

Haberman, S. J. (1976). Generalized Residuals for Log-Linear Models, Proceedings of the 9th International Biometrics Conference, Boston, pp. 104-122.

Landwehr, J. M., Pregibon, D., and Shoemaker, A. C. (1984), Graphical Models for Assessing Logistic Regression Models (with discussion), Journal of the American Statistical Association 79: 61-83.

Lo, W.-D. (1993). Logistic Regression Trees, PhD thesis, Department of Statistics, University of Wisconsin, Madison, WI.

Examples

```
habermanTra$data
habermanTra$fuzzySets
```

`MESDIF` *Multiobjective Evolutionary Subgroup DIscovery Fuzzy rules (MESDIF) Algorithm*

Description

Performs a subgroup discovery task executing the MESDIF algorithm.

Usage

```
MESDIF(
  paramFile = NULL,
  training = NULL,
  test = NULL,
  output = c("optionsFile.txt", "rulesFile.txt", "testQM.txt"),
  seed = 0,
  nLabels = 3,
  nEval = 10000,
  popLength = 100,
  eliteLength = 3,
  crossProb = 0.6,
  mutProb = 0.01,
  RulesRep = "can",
  Obj1 = "CSUP",
  Obj2 = "CCNF",
```

```
  Obj3 = "null",
  Obj4 = "null",
  targetVariable = NA,
  targetClass = "null"
)
```

Arguments

paramFile	The path of the parameters file. NULL If you want to use training and test SDEFSR_Dataset variables
training	A SDEFSR_Dataset class variable with training data.
test	A SDEFSR_Dataset class variable with test data. NULL if you only want to use training data.
output	character vector with the paths where store information file, rules file and quality measures file, respectively.
seed	An integer to set the seed used for generate random numbers.
nLabels	Number of linguistic labels that represents numerical variables.
nEval	An integer for set the maximum number of evaluations in the evolutive process. Large values of this parameter increments the computing time.
popLength	An integer to set the number of individuals in the population.
eliteLength	An integer to set the number of individuals in the elite population.
crossProb	Sets the crossover probability. A number in [0,1].
mutProb	Sets the mutation probability. A number in [0,1].
RulesRep	Representation used in the rules. "can" for canonical rules, "dnf" for DNF rules.
Obj1	Sets the Objective number 1. See Objective values for more information about the possible values.
Obj2	Sets the Objective number 2. See Objective values for more information about the possible values.
Obj3	Sets the Objective number 3. See Objective values for more information about the possible values.
Obj4	Sets the Objective number 4. See Objective values for more information about the possible values.
targetVariable	The name or index position of the target variable (or class). It must be a categorical one.
targetClass	A string specifing the value of the target variable. null for search for all possible values.

Details

This function sets as target variable the last one that appear in SDEFSR_Dataset object. If you want to change the target variable, you can set the targetVariable to change this target variable. The target variable MUST be categorical, if it is not, throws an error. Also, the default behaviour is to find rules for all possible values of the target varaible. targetClass sets a value of the target variable where the algorithm only finds rules about this value.

If you specify in paramFile something distinct to NULL the rest of the parameters are ignored and the algorithm tries to read the file specified. See "Parameters file structure" below if you want to use a parameters file.

Value

The algorithm shows in the console the following results:

1. The parameters used in the algorithm
2. The rules generated.
3. The quality measures for test of every rule and the global results. This globals results shows the number of rules generated and means results for each quality measure.

Also, the algorithms save those results in the files specified in the output parameter of the algorithm or in the outputData parameter in the parameters file.

Additionally a SDEFSR_Rules object is returned with this information.

How does this algorithm work?

This algorithm performs a multi-objective genetic algorithm based on elitism (following the SPEA2 approach). The elite population has a fixed size and it is filled by non-dominated individuals.

An individual is non-dominated when `(! all(ObjI1 <= ObjI2) & any(ObjI1 < ObjI2))` where ObjI1 is the objective value for our individual and ObjI2 is the objetive value for another individual. The number of dominated individuals by each one determine, in addition with a niches technique that considers the proximity among values of the objectives a fitness value for the selection.

The number of non-dominated individuals might be greater or less than elite population size and in those cases MESDIF implements a truncation operator and a fill operator respectively. Then, genetic operators are applied.

At the final of the evolutive process it returns the rules stored in elite population. Therefore, the number of rules is fixed with the eliteLength parameter.

Parameters file structure

The paramFile argument points to a file which has the necesary parameters for MESDIF works. This file **must** have, at least, those parameters (separated by a carriage return):

- algorithm Specify the algorithm to execute. In this case. "MESDIF"
- inputData Specify two paths of KEEL files for training and test. In case of specify only the name of the file, the path will be the working directory.
- seed Sets the seed for the random number generator
- nLabels Sets the number of fuzzy labels to create when reading the files
- nEval Set the maximun number of **evaluations of rules** for stop the genetic process
- popLength Sets number of individuals of the main population
- eliteLength Sets number of individuals of the elite population. Must be less than popLength
- crossProb Crossover probability of the genetic algorithm. Value in [0,1]
- mutProb Mutation probability of the genetic algorithm. Value in [0,1]

- `Obj1` Sets the objective number 1.
- `Obj2` Sets the objective number 2.
- `Obj3` Sets the objective number 3.
- `Obj4` Sets the objective number 4.
- `RulesRep` Representation of each chromosome of the population. "can" for canonical representation. "dnf" for DNF representation.
- `targetVariable` The name or index position of the target variable (or class). It must be a categorical one.
- `targetClass` Value of the target variable to search for subgroups. The target variable **is always the last variable.** Use `null` to search for every value of the target variable

An example of parameter file could be:

```
algorithm = MESDIF
inputData = "irisd-10-1tra.dat" "irisd-10-1tst.dat"
outputData = "irisD-10-1-INFO.txt" "irisD-10-1-Rules.txt" "irisD-10-1-TestMeasures.txt"
seed = 0
nLabels = 3
nEval = 500
popLength = 100
eliteLength = 3
crossProb = 0.6
mutProb = 0.01
RulesRep = can
Obj1 = comp
Obj2 = unus
Obj3 = null
Obj4 = null
targetClass = Iris-setosa
```

@section Objective values: You can use the following quality measures in the ObjX value of the parameter file using this values:

- Unusualness -> `unus`
- Crisp Support -> `csup`
- Crisp Confidence -> `ccnf`
- Fuzzy Support -> `fsup`
- Fuzzy Confidence -> `fcnf`
- Coverage -> `cove`
- Significance -> `sign`

If you dont want to use a objective value you must specify `null`

References

- Berlanga, F., Del Jesus, M., Gonzalez, P., Herrera, F., & Mesonero, M. (2006). Multiobjective Evolutionary Induction of Subgroup Discovery Fuzzy Rules: A Case Study in Marketing.
- Zitzler, E., Laumanns, M., & Thiele, L. (2001). SPEA2: Improving the Strength Pareto Evolutionary Algorithm.

Examples

```
 MESDIF( paramFile = NULL,
        training = habermanTra,
        test = habermanTst,
        output = c(NA, NA, NA),
        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"
        )

## Not run:
Execution for all classes, see 'targetClass' parameter
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 = "null"
        )

## End(Not run)
```

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

Description

Perfoms a subgroup discovery task executing the algorithm NMEEF-SD

Usage

```
NMEEF_SD(
  paramFile = NULL,
  training = NULL,
  test = NULL,
  output = c("optionsFile.txt", "rulesFile.txt", "testQM.txt"),
  seed = 0,
  nLabels = 3,
  nEval = 10000,
  popLength = 100,
  mutProb = 0.1,
  crossProb = 0.6,
  Obj1 = "CSUP",
  Obj2 = "CCNF",
  Obj3 = "null",
  minCnf = 0.6,
  reInitCoverage = "yes",
  porcCob = 0.5,
  StrictDominance = "yes",
  targetVariable = NA,
  targetClass = "null"
)
```

Arguments

paramFile	The path of the parameters file. NULL If you want to use training and test SDEFSR_Dataset variables
training	A SDEFSR_Dataset class variable with training data.
test	A SDEFSR_Dataset class variable with training data.
output	character vector with the paths of where store information file, rules file and test quality measures file, respectively.
seed	An integer to set the seed used for generate random numbers.
nLabels	Number of linguistic labels for numerical variables.
nEval	An integer for set the maximum number of evaluations in the evolutionary process.
popLength	An integer to set the number of individuals in the population.
mutProb	Sets the mutation probability. A number in [0,1].
crossProb	Sets the crossover probability. A number in [0,1].
Obj1	Sets the Objective number 1. See Objective values for more information about the possible values.
Obj2	Sets the Objective number 2. See Objective values for more information about the possible values.
Obj3	Sets the Objective number 3. See Objective values for more information about the possible values.

`minCnf`	Sets the minimum confidence that must have a rule in the Pareto front for being returned. A number in [0,1].
`reInitCoverage`	Sets if the algorithm must perform the reinitialitation based on coverage when it is needed. A string with "yes" or "no".
`porcCob`	Sets the maximum percentage of variables that participate in the rules generated in the reinitialitation based on coverage. A number in [0,1]
`StrictDominance`	Sets if the comparison between individuals must be done by strict dominance or not. A string with "yes" or "no".
`targetVariable`	The name or index position of the target variable (or class). It must be a categorical one.
`targetClass`	A string specifing the value the target variable. `null` for search for all possible values.

Details

This function sets as target variable the last one that appear in `SDEFSR_Dataset` object. If you want to change the target variable, you can set the `targetVariable` to change this target variable. The target variable MUST be categorical, if it is not, throws an error. Also, the default behaviour is to find rules for all possible values of the target varaible. `targetClass` sets a value of the target variable where the algorithm only finds rules about this value.

If you specify in `paramFile` something distinct to `NULL` the rest of the parameters are ignored and the algorithm tries to read the file specified. See "Parameters file structure" below if you want to use a parameters file.

Value

The algorithm shows in the console the following results:

1. The parameters used in the algorithm
2. The rules generated.
3. The quality measures for test of every rule and the global results.

   Also, the algorithms save those results in the files specified in the `output` parameter of the algorithm or in the `outputData` parameter in the parameters file.

How does this algorithm work?

NMEEF-SD is a multiobjetctive genetic algorithm based on a NSGA-II approach. The algorithm first makes a selection based on binary tournament and save the individuals in a offspring population. Then, NMEEF-SD apply the genetic operators over individuals in offspring population

For generate the population which participate in the next iteration of the evolutionary process NMEEF-SD calculate the dominance among all individuals (join main population and offspring) and then, apply the NSGA-II fast sort algorithm to order the population by fronts of dominance, the first front is the non-dominated front (or Pareto), the second is where the individuals dominated by one individual are, the thirt front dominated by two and so on.

To promove diversity NMEEF-SD has a mechanism of reinitialization of the population based on coverage if the Pareto doesnt evolve during a 5

At the final of the evolutionary process, the algorithm returns only the individuals in the Pareto front which has a confidence greater than a minimum confidence level.

Parameters file structure

The `paramFile` argument points to a file which has the necesary parameters for NMEEF-SD works. This file **must** be, at least, those parameters (separated by a carriage return):

- `algorithm` Specify the algorithm to execute. In this case. "NMEEFSD"
- `inputData` Specify two paths of KEEL files for training and test. In case of specify only the name of the file, the path will be the working directory.
- `seed` Sets the seed for the random number generator
- `nLabels` Sets the number of fuzzy labels to create when reading the files
- `nEval` Set the maximun number of **evaluations of rules** for stop the genetic process
- `popLength` Sets number of individuals of the main population
- `ReInitCob` Sets if NMEEF-SD do the reinitialization based on coverage. Values: "yes" or "no"
- `crossProb` Crossover probability of the genetic algorithm. Value in [0,1]
- `mutProb` Mutation probability of the genetic algorithm. Value in [0,1]
- `RulesRep` Representation of each chromosome of the population. "can" for canonical representation. "dnf" for DNF representation.
- `porcCob` Sets the maximum percentage of variables participe in a rule when doing the reinitialization based on coverage. Value in [0,1]
- `Obj1` Sets the objective number 1.
- `Obj2` Sets the objective number 2.
- `Obj3` Sets the objective number 3.
- `minCnf` Minimum confidence for returning a rule of the Pareto. Value in [0,1]
- `StrictDominance` Sets if the comparison of individuals when calculating dominance must be using strict dominance or not. Values: "yes" or "no"
- `targetClass` Value of the target variable to search for subgroups. The target variable **is always the last variable.**. Use `null` to search for every value of the target variable

An example of parameter file could be:

```
algorithm = NMEEFSD
inputData = "irisd-10-1tra.dat" "irisd-10-1tra.dat" "irisD-10-1tst.dat"
outputData = "irisD-10-1-INFO.txt" "irisD-10-1-Rules.txt" "irisD-10-1-TestMeasures.txt"
seed = 1
RulesRep = can
nLabels = 3
nEval = 500
popLength = 51
crossProb = 0.6
mutProb = 0.1
```

```
ReInitCob = yes
porcCob = 0.5
Obj1 = comp
Obj2 = unus
Obj3 = null
minCnf = 0.6
StrictDominance = yes
targetClass = Iris-setosa
```

Objective values

You can use the following quality measures in the ObjX value of the parameter file using this values:

- Unusualness -> `unus`
- Crisp Support -> `csup`
- Crisp Confidence -> `ccnf`
- Fuzzy Support -> `fsup`
- Fuzzy Confidence -> `fcnf`
- Coverage -> `cove`
- Significance -> `sign`

If you dont want to use a objetive value you must specify `null`

References

Carmona, C., Gonzalez, P., del Jesus, M., & Herrera, F. (2010). NMEEF-SD: Non-dominated Multi-objective Evolutionary algorithm for Extracting Fuzzy rules in Subgroup Discovery.

Examples

```
NMEEF_SD(paramFile = NULL,
            training = habermanTra,
            test = habermanTst,
            output = c(NA, NA, NA),
            seed = 0,
            nLabels = 3,
            nEval = 300,
            popLength = 100,
            mutProb = 0.1,
            crossProb = 0.6,
            Obj1 = "CSUP",
            Obj2 = "CCNF",
            Obj3 = "null",
            minCnf = 0.6,
            reInitCoverage = "yes",
            porcCob = 0.5,
            StrictDominance = "yes",
            targetClass = "positive"
            )
## Not run:
```

```
        NMEEF_SD(paramFile = NULL,
                 training = habermanTra,
                 test = habermanTst,
                 output = c("optionsFile.txt", "rulesFile.txt", "testQM.txt"),
                 seed = 0,
                 nLabels = 3,
                 nEval = 300,
                 popLength = 100,
                 mutProb = 0.1,
                 crossProb = 0.6,
                 Obj1 = "CSUP",
                 Obj2 = "CCNF",
                 Obj3 = "null",
                 minCnf = 0.6,
                 reInitCoverage = "yes",
                 porcCob = 0.5,
                 StrictDominance = "yes",
                 targetClass = "null"
                 )

    ## End(Not run)
```

`plot.SDEFSR_Rules` *Plot a rule set generated by a SDEFSR algorithm*

Description

This function plots the rule set by means of a bar graph that shows TPR vs FPR quality measure of each rule

Usage

```
## S3 method for class 'SDEFSR_Rules'
plot(x, ...)
```

Arguments

`x`	an `SDEFSR_Rules` object generated by a subgroup discovery algorithm of the SDEFSR package
`...`	additional arguments passed to the plot

Details

This function works depending on the package ggplot2 that allow to generate such graph. If the package ggplot2 is not installed, the this function ask the user to install it. After install, load the package and show the graph.

A TPR vs FPR graph show the precision of a rule. Quality rules has big TPR values and small FPR values. Big values of both quality measures indicates that the rule is too much general and it is too obvious. Small values of both indicates that the rule is too much specific and it would be an invalid rule.

Value

A TPR vs FPR graph generated by ggplot2

Examples

```
plot(habermanRules)
```

`print.SDEFSR_Dataset` *S3 function to print in console the contents of the dataset This function shows the matrix of data uncoded.*

Description

S3 function to print in console the contents of the dataset

This function shows the matrix of data uncoded.

Usage

```
## S3 method for class 'SDEFSR_Dataset'
print(x, ...)
```

Arguments

`x`	The `SDEFSR_Dataset` object to view
`...`	Additional arguments passed to the print function

Details

This function show the matix of data. Internally, a `SDEFSR_Dataset` object has a list of of examples and this examples are coded numerically. This function decode these examples and convert the list into a matrix.

Value

a matrix with the dataset uncoded.

@examples

print(habermanTra)

`read.dataset` *Reads a KEEL, ARFF or CSV data format file.*

Description

This function reads a KEEL (.dat), ARFF (.arff) or CSV dataset file and store the information in a `SDEFSR_Dataset` class.

Usage

```
read.dataset(file, sep = ",", quote = "\"", dec = ".", na.strings = "?")
```

Arguments

`file`	The path of the file to read
`sep`	The separator character to use
`quote`	The character used to take quotes
`dec`	The character used to define decimal characters
`na.strings`	The character to detect lost data

Details

A KEEL data file must have the following structure:

- @relation: Name of the data set
- @attribute: Description of an attribute (one for each attribute)
- @inputs: List with the names of the input attributes
- @output: Name of the output attribute (Not used in this algorithms implementation)
- @data: Starting tag of the data

The rest of the file contains all the examples belonging to the data set, expressed in comma seppa-rated values format. ARFF file format is a well-know dataset format from WEKA data mining tool. CSV is a format which means comma-separated values. Where each examples is on a line and each value of the variables of the examples are separated by commas.

Author(s)

Angel M. Garcia <agvico@ujaen.es>

References

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. Journal of Multiple-Valued Logic and Soft Computing 17:2-3 (2011) 255-287.

See Also

KEEL Dataset Repository (Standard Classification): https://sci2s.ugr.es/keel/category.php?cat=clas

Examples

```
    ## Not run:
       Reads a KEEL dataset from a file.
       read.dataset(file = "C:\KEELFile.dat")

       read.dataset(file = "C:\KEELFile.dat", nLabels = 7)

     Reads an ARFF dataset from a file.
       read.dataset(file = "C:\ARFFFile.arff")

       read.dataset(file = "C:\ARFFFile.arff", nLabels = 7)

## End(Not run)
```

SDEFSR *SDEFSR: A package for Subgroup Discovery with Evolutionary Fuzzy Systems in R*

Description

The SDEFSR package provide a tool for read KEEL datasets and four evolutionary fuzzy rule-based algorithms for subgroup discovery.

Details

The algorithms provided works with datasets in KEEL, ARFF or CSV format and also with `data.frame` objects.

The package also provide a Shiny app for making the same tasks that the package can do and can display some additional information about data for making an exploratory analysis.

The algorithms provided are Evolutionary Fuzzy Systems (EFS) which take advantages of evolutionary algorithms for maximize more than one quality measure and fuzzy logic, which makes a representation of numerical variables that are more understandable for humans and more robust to noise.

The algorithms in the SDEFSR package support target variable with more than two values. However, this target variables must be categorical. Thus, if you have a numeric target variable, a discretization must be perfomed before executing the method.

SDEFSR functions

- MESDIF Multiobjective Evolutionary Subgroup DIscovery Fuzzy rules (MESDIF) Algorithm.
- NMEEF_SD Non-dominated Multi-objective Evolutionary algorithm for Extracting Fuzzy rules in Subgroup Discovery (NMEEF-SD).
- read.dataset reads a KEEL, ARFF or CSV format file.
- SDIGA Subgroup Discovery Iterative Genetic Algorithm (SDIGA).
- SDEFSR_GUI Launch the Shiny app in your browser.
- FUGEPSD Fuzzy Genetic Programming-based learning for Subgroup Discovery (FuGePSD) Algorithm.
- plot.SDEFSR_Rules Plot the discovered rules by a SDEFSR algorithm.
- sort.SDEFSR_Rules Sort the discovered rules by a given quality measure.
- SDEFSR_DatasetFromDataFrame Reads a data.frame and create a SDEFSR_Dataset object to be execute by an algorithm of this package.

Author(s)

Angel M. Garcia-Vico <agvico@ujaen.es>

SDEFSR_DatasetFromDataFrame

Creates a SDEFSR_Dataset *object from a* data.frame

Description

Creates a SDEFSR_Dataset object from a data.frame and create fuzzy labels for numerical variables too.

Usage

```
SDEFSR_DatasetFromDataFrame(
  data,
  relation,
  names = NA,
  types = NA,
  classNames = NA
)
```

Arguments

data	A data.frame object with all neccesary information. See details.
relation	A string that indicate the name of the relation.
names	An optional character vector indicating the name of the attributes.
types	An optional character vector indicating 'c' if variable is categorical, 'r' if is real and 'e' if it is an integer
classNames	An optional character vector indicating the values of the target class.

Details

The information of the data.frame must be stored with instances in rows and variables in columns If you dont specify any of the optional parameter the function try to obtain them automatically.

For `'names'` if it is NA, the function takes the name of the columns by `colnames`.

For `'types'` if it is NA, the function takes the type of an attribute asking the type of the column of the data.frame. If it is `'character'` it is assumed that it is categorical, and if `'numeric'` it is assumed that it is a real number. PLEASE, PAY ATTENTION TO THIS WAY OF WORK. It can cause tranformation errors taking a numeric variable as categorical or vice-versa.

For `'classNames'` if it is NA, the function returns unique values of the last attribute of the data.frame that is considered the class attribute.

Value

A `SDEFSR_Dataset` object with all the information of the dataset.

Author(s)

Angel M Garcia <amgv0009@red.ujaen.es>

See Also

`read.dataset`

Examples

```
library(SDEFSR)
df <- data.frame(matrix(runif(1000), ncol = 10))
#Add class attribute
df[,11] <- c("0", "1", "2", "3")
SDEFSR_DatasetObject <- SDEFSR_DatasetFromDataFrame(df, "random")
invisible()
```

`SDEFSR_GUI` *Launch a web interface for use the algorithms easily.*

Description

Launches a Shiny-based interface for the package in your browser.

Usage

```
SDEFSR_GUI()
```

Details

The package SDEFSR provide simple, shiny-based web interface for performs the taks easily. The interface only work with new datasets loaded directly in the platform.

The web application is structured as follows:

- The first you have to do is load your training and test files. This files must be valids KEEL format files.
- After chose your datasets, you can view information about the dataset or execute the algorithm
- You can choose the target variable or the variable to visualize and choose the target value or execute the algorithm for all the values.
- Choosed the target variable, you can choose the algorithm to execute and change his parameters with the controls provided.
- After you can execute the algorithm. The results are exposed in three tabs that are at the top of the page, just at the right of the "Exploratory Analysis" tab.

The tables can be sorted for each value and also you can search and filter values.

Examples

```
## Not run:
library(SDEFSR)
SDEFSR_GUI()

## End(Not run)
```

SDIGA *Subgroup Discovery Iterative Genetic Algorithm (SDIGA)*

Description

Perfoms a subgroup discovery task executing the algorithm SDIGA

Usage

```
SDIGA(
  parameters_file = NULL,
  training = NULL,
  test = NULL,
  output = c("optionsFile.txt", "rulesFile.txt", "testQM.txt"),
  seed = 0,
  nLabels = 3,
  nEval = 10000,
  popLength = 100,
  mutProb = 0.01,
  RulesRep = "can",
```

```
  Obj1 = "CSUP",
  w1 = 0.7,
  Obj2 = "CCNF",
  w2 = 0.3,
  Obj3 = "null",
  w3 = 0,
  minConf = 0.6,
  lSearch = "yes",
  targetVariable = NA,
  targetClass = "null"
)
```

Arguments

parameters_file	The path of the parameters file. NULL If you want to use training and test SDEFSR_Dataset variables
training	A SDEFSR_Dataset class variable with training data.
test	A SDEFSR_Dataset class variable with training data.
output	character vector with the paths of where store information file, rules file and test quality measures file, respectively.
seed	An integer to set the seed used for generate random numbers.
nLabels	Number of linguistic labels that represents numerical variables.
nEval	An integer for set the maximum number of evaluations in the evolutive process.
popLength	An integer to set the number of individuals in the population.
mutProb	Sets the mutation probability. A number in [0,1].
RulesRep	Representation used in the rules. "can" for canonical rules, "dnf" for DNF rules.
Obj1	Sets the Objective number 1. See Objective values for more information about the possible values.
w1	Sets the weight of Obj1.
Obj2	Sets the Objective number 2. See Objective values for more information about the possible values.
w2	Sets the weight of Obj2.
Obj3	Sets the Objective number 3. See Objective values for more information about the possible values.
w3	Sets the weight of Obj3.
minConf	Sets the minimum confidence that must have the rule returned by the genetic algorithm after the local optimitation phase. A number in [0,1].
lSearch	Sets if the local optimitation phase must be performed. A string with "yes" or "no".
targetVariable	A string with the name or an integer with the index position of the target variable (or class). It must be a categorical one.
targetClass	A string specifing the value the target variable. null for search for all possible values.

Details

This function sets as target variable the last one that appear in SDEFSR_Dataset object. If you want to change the target variable, you can set the targetVariable to change this target variable. The target variable MUST be categorical, if it is not, throws an error. Also, the default behaviour is to find rules for all possible values of the target varaible. targetClass sets a value of the target variable where the algorithm only finds rules about this value.

If you specify in paramFile something distinct to NULL the rest of the parameters are ignored and the algorithm tries to read the file specified. See "Parameters file structure" below if you want to use a parameters file.

Value

The algorithm shows in the console the following results:

1. The parameters used in the algorithm
2. The rules generated.
3. The quality measures for test of every rule and the global results.

Also, the algorithms save those results in the files specified in the output parameter of the algorithm or in the outputData parameter in the parameters file.

How does this algorithm work?

This algorithm has a genetic algorithm in his core. This genetic algorithm returns only the best rule of the population and it is executed so many times until a stop condition is reached. The stop condition is that the rule returned must cover at least one new example (not covered by previous rules) and must have a confidence greater than a minimum.

After returning the rule, a local improvement could be applied for make the rule more general. This local improve is done by means of a hill-climbing local search.

The genetic algorithm cross only the two best individuals. But the mutation operator is applied over all the population, individuals from cross too.

Parameters file structure

The parameters_file argument points to a file which has the necesary parameters for SDIGA works. This file **must** be, at least, those parameters (separated by a carriage return):

- algorithm Specify the algorithm to execute. In this case. "SDIGA"
- inputData Specify two paths of KEEL files for training and test. In case of specify only the name of the file, the path will be the working directory.
- seed Sets the seed for the random number generator
- nLabels Sets the number of fuzzy labels to create when reading the files
- nEval Set the maximun number of **evaluations of rules** for stop the genetic process
- popLength Sets number of individuals of the main population
- mutProb Mutation probability of the genetic algorithm. Value in [0,1]

- RulesRep Representation of each chromosome of the population. "can" for canonical representation. "dnf" for DNF representation.
- Obj1 Sets the objective number 1.
- w1 Sets the weigth assigned to the objective number 1. Value in [0,1]
- Obj2 Sets the objective number 2.
- w2 Sets the weigth assigned to the objective number 2. Value in [0,1]
- Obj3 Sets the objective number 3.
- w3 Sets the weigth assigned to the objective number 3. Value in [0,1]
- minConf Sets the minimum confidence of the rule for checking the stopping criteria of the iterative process
- lSearch Perform the local search algorithm after the execution of the genetic algorithm? Values: "yes" or "no"
- targetVariable The name or index position of the target variable (or class). It must be a categorical one.
- targetClass Value of the target variable to search for subgroups. The target variable **is always the last variable.**. Use null to search for every value of the target variable

An example of parameter file could be:

```
algorithm = SDIGA
inputData = "irisD-10-1tra.dat" "irisD-10-1tst.dat"
outputData = "irisD-10-1-INFO.txt" "irisD-10-1-Rules.txt" "irisD-10-1-TestMeasures.txt"
seed = 0
nLabels = 3
nEval = 500
popLength = 100
mutProb = 0.01
minConf = 0.6
RulesRep = can
Obj1 = Comp
Obj2 = Unus
Obj3 = null
w1 = 0.7
w2 = 0.3
w3 = 0.0
lSearch = yes
```

Objective values

You can use the following quality measures in the ObjX value of the parameter file using this values:

- Unusualness -> unus
- Crisp Support -> csup
- Crisp Confidence -> ccnf
- Fuzzy Support -> fsup

- Fuzzy Confidence -> `fcnf`
- Coverage -> `cove`
- Significance -> `sign`

If you dont want to use a objetive value you must specify `null`

References

M. J. del Jesus, P. Gonzalez, F. Herrera, and M. Mesonero, "Evolutionary Fuzzy Rule Induction Process for Subgroup Discovery: A case study in marketing," IEEE Transactions on Fuzzy Systems, vol. 15, no. 4, pp. 578-592, 2007.

Examples

```
SDIGA(parameters_file = NULL,
      training = habermanTra,
      test = habermanTst,
      output = c(NA, NA, NA),
      seed = 0,
      nLabels = 3,
      nEval = 300,
      popLength = 100,
      mutProb = 0.01,
      RulesRep = "can",
      Obj1 = "CSUP",
      w1 = 0.7,
      Obj2 = "CCNF",
      w2 = 0.3,
      Obj3 = "null",
      w3 = 0,
      minConf = 0.6,
      lSearch = "yes",
      targetClass = "positive")
## Not run:
SDIGA(parameters_file = NULL,
      training = habermanTra,
      test = habermanTst,
      output = c("optionsFile.txt", "rulesFile.txt", "testQM.txt"),
      seed = 0,
      nLabels = 3,
      nEval = 300,
      popLength = 100,
      mutProb = 0.01,
      RulesRep = "can",
      Obj1 = "CSUP",
      w1 = 0.7,
      Obj2 = "CCNF",
      w2 = 0.3,
      Obj3 = "null",
      w3 = 0,
      minConf = 0.6,
      lSearch = "yes",
```

```
      targetClass = "positive")

## End(Not run)
```

`sort.SDEFSR_Rules` *@title Return an ordered rule set by a given quality measure @description This function sorts a rule set in descendant order by a given quality measure that are available on the object*

Description

@title Return an ordered rule set by a given quality measure

@description This function sorts a rule set in descendant order by a given quality measure that are available on the object

Usage

```
## S3 method for class 'SDEFSR_Rules'
sort(x, decreasing = TRUE, ...)
```

Arguments

`x`	The rule set passed as a `SDEFSR_Rules` object
`decreasing`	A logical indicating if the sort should be increasing or decreasing. By default, decreasing.
`...`	Additional parameters as "by", a String with the name of the quality measure to order by. Valid values are: `nVars`, `Coverage`, `Unusualness`, `Significance`, `FuzzySupport`, `Support`, `FuzzyConfidence`, `Confidence`, `Tpr`, `Fpr`.

Details

The additional argument in "..." is the 'by' argument, which is a s string with the name of the quality measure to order by. Valid values are: `nVars`, `Coverage`, `Unusualness`, `Significance`, `FuzzySupport`, `Support`, `FuzzyConfidence`, `Confidence`, `Tpr`, `Fpr`.

Value

another `SDEFSR_Rules` object with the rules sorted

Examples

```
sort(habermanRules)
```

summary.SDEFSR_Dataset *S3 function to summary a SDEFSR_Dataset object*

Description

Summary relevant data of a SDEFSR_Dataset dataset.

Usage

```
## S3 method for class 'SDEFSR_Dataset'
summary(object, ...)
```

Arguments

object	A SDEFSR_Dataset class.
...	Additional arguments to the summary function.

Details

This function show important information about the SDEFSR_Dataset dataset for the user. Note that it does not show all the information available. The rest is only for the algorithms. The values that appear are accessible by the $ operator, e.g. dataset$relation or dataset$examplesPerClass.

Examples

```
summary(carTra)
```

[.SDEFSR_Rules *Filter rules in a* SDEFSR_Rules *object returning a new* SDEFSR_Rules *object*

Description

Generates a new SDEFSR_Rules object containing the rules that passed the filter specified

Usage

```
## S3 method for class 'SDEFSR_Rules'
SDEFSR_RulesObject[condition = T]
```

Arguments

SDEFSR_RulesObject	The SDEFSR_RulesObject object to filter
condition	Expression to filter the SDEFSR_Rules object

Details

This functions allows to filter the rule set by a given quality measure. The quality measures that are available are: `nVars`, `Coverage`, `Unusualness`, `Significance`, `FuzzySupport`, `Support`, `FuzzyConfidence`, `Confidence`, `TPr` and `FPr`

Examples

```
library(SDEFSR)
#Apply filter by unusualness
habermanRules[Unusualness > 0.05]

#Also, you can make the filter as complex as you can
#Filter by Unusualness and TPr
habermanRules[Unusualness > 0.05 & TPr > 0.9]
```

Index

[.SDEFSR_Rules, 33

as.data.frame.SDEFSR_Dataset, 2

carTra, 3
carTst, 4

FUGEPSD, 4, *25*

germanTra, 9
germanTst, 9

habermanRules, 10
habermanTra, 11
habermanTst, 11

MESDIF, 12, *25*

NMEEF_SD, 16, *25*

plot.SDEFSR_Rules, 21, *25*
print.SDEFSR_Dataset, 22

read.dataset, 23, *25, 26*

SDEFSR, 24
SDEFSR_DatasetFromDataFrame, *25*, 25
SDEFSR_GUI, *25*, 26
SDIGA, *25*, 27
sort.SDEFSR_Rules, *25*, 32
summary.SDEFSR_Dataset, 33