Package 'RoughSets'

October 12, 2022

Maintainer Christoph Bergmeir `<c.bergmeir@decsai.ugr.es>`

License GPL (>= 2)

Encoding UTF-8

Title Data Analysis Using Rough Set and Fuzzy Rough Set Theories

Description Implementations of algorithms for data analysis based on the
rough set theory (RST) and the fuzzy rough set theory (FRST). We not only
provide implementations for the basic concepts of RST and FRST but also
popular algorithms that derive from those theories. The methods included in the
package can be divided into several categories based on their functionality:
discretization, feature selection, instance selection, rule induction and
classification based on nearest neighbors. RST was introduced by Zdzisław
Pawlak in 1982 as a sophisticated mathematical tool to model and process
imprecise or incomplete information. By using the indiscernibility relation for
objects/instances, RST does not require additional parameters to analyze the
data. FRST is an extension of RST. The FRST combines concepts of vagueness and
indiscernibility that are expressed with fuzzy sets (as proposed by Zadeh, in
1965) and RST.

Version 1.3-7

URL <https://github.com/janusza/RoughSets>

Date 2019-12-15

Suggests class

LinkingTo Rcpp

Depends Rcpp

RoxygenNote 6.1.0

NeedsCompilation yes

Author Andrzej Janusz [aut],
Lala Septem Riza [aut],
Dominik Ślęzak [ctb],
Chris Cornelis [ctb],
Francisco Herrera [ctb],
Jose Manuel Benitez [ctb],
Christoph Bergmeir [ctb, cre],
Sebastian Stawicki [ctb]

1

R **topics documented:**

RoughSets-package   *Getting started with the RoughSets package*

Description

This part contains global explanations about the implementation and use of the `RoughSets` package.
The package `RoughSets` attempts to provide a complete tool to model and analyze information systems based on rough set theory (RST) and fuzzy rough set theory (FRST). From fundamental point of view, this package allows to construct rough sets by defining lower and upper approximations. Furthermore, recent methods for tackling common tasks in data mining, such as data preprocessing (e.g., discretization, feature selection, missing value completion, and instance selection), rule induction, and prediction classes or decision values of new datasets are available as well.

Details

There are two main parts considered in this package which are RST and FRST. RST was introduced by (Pawlak, 1982; Pawlak, 1991) which provides sophisticated mathematical tools to model and analyze information systems that involve uncertainty and imprecision. By employing indiscernibility relation among objects, RST does not require additional parameters to extract information. The detailed explanation about fundamental concepts of RST can be read in Section `A.Introduction-RoughSets`. Secondly, FRST, an extension of RST, was introduced by (Dubois and Prade, 1990) as a combination between fuzzy sets proposed by (Zadeh, 1965) and RST. This concept allows to analyze continuous attributes without performing discretization on data first. Basic concepts of FRST can be seen in `B.Introduction-FuzzyRoughSets`.

Based on the above concepts, many methods have been proposed and applied for dealing with several different domains. In order to solve the problems, methods employ the indiscernibility relation and lower and upper approximation concepts. All methods that have been implemented in this package will be explained by grouping based on their domains. The following is a list of domains considered in this package:

- Basic concepts: This part, we can divide into four different tasks which are indiscernibility relation, lower and upper approximations, positive region and discernibility matrix. All of those tasks have been explained briefly in Section `A.Introduction-RoughSets` and

  `B.Introduction-FuzzyRoughSets`.

- Discretization: It is used to convert real-valued attributes into nominal/symbolic ones in an information system. In RST point of view, this task attempts to maintain the discernibility between objects.

- Feature selection: It is a process for finding a subset of features which have the same quality as the complete feature set. In other words, its purpose is to select the significant features and eliminate the dispensible ones. It is a useful and necessary process when we are facing datasets containing large numbers of features. From RST and FRST perspective, feature selection refers to searching superreducts and reducts. The detailed information about reducts can be read in `A.Introduction-RoughSets` and `B.Introduction-FuzzyRoughSets`.

- Instance selection: This process is aimed to remove noisy, superfluous, or inconsistent instances from training datasets but retain consistent ones. In other words, good accuracy of classification is achieved by removing instances which do not give positive contributions.

- Prediction/classification: This task is used to predict decision values of a new dataset (test data). We consider implementing some methods to perform this task, such as fuzzy-rough nearest neighbor approaches, etc.

- Rule induction: This task refers to generate IF - THEN rules. The rule represents knowledge which is contained in a dataset. One advantage of building rules is that naturally the model is easy to interpret. Then, predicted values over new datasets can be determined by considering the rules.

As we mentioned before, we have embedded many well-known algorithms or techniques for handling the above domains. The algorithms were considered since experimentally it has been proven that they were able to tackle complex tasks. They are implemented as functions that were organized to work with the same data structures. So, users can perform various approaches for a particular task easily and then compare their results. In order to be recognized quickly, generally we have chosen the names of the functions with some conventions. The names contain three parts which are

prefix, suffix, and middle that are separated by a point. The following is a description of each part.

- prefix: There are some different prefixes for names of functions expressing a kind of task to be performed. The function names with prefix BC refer to *basic concepts* which means that the functions are created for implementing the basic concepts of RST and FRST. While prefix D refers to *discretization*, FS, IS, RI, MV, and C refer to *feature selection*, *instance selection*, *rule induction*, *missing value completion*, and *classifier based on nearest neighbor* domains. Furthermore, SF and X mean that functions are used as *supporting functions* which are not related directly with RST and FRST and *auxiliary* functions which are called as a parameter.

- suffix: It is located at the end of names. There are two types available: RST and FRST. RST represents *rough set theory* while FRST shows that the function is applied to *fuzzy rough set theory*. Additionally, some functions that do not have RST or FRST suffix are used for both theories.

- middle: All other words in the middle of the names are used to express the actual name of a particular method/algorithm or functionality. In this case, it could consist of more than one word separated by points.

For instance, the function BC.IND.relation.RST is used to calculate the indiscernibility relation which is one of the basic concepts of RST. Other functions that have names not based on the above rules are S3 functions e.g., summary and predict which are used to summarize objects and predict new data, respectively.

The following description explains domains and their algorithms implemented in the package:

1. **The implementations of RST**: This part outlines some considered algorihtms/methods based on RST. The approaches can be classified based on their tasks as follows:

    (a) The basic concepts: The following is a list showing tasks and their implementations as functions.
        - Indiscernibility relation: It is a relation determining whether two objects are indiscernible by some attributes. It is implemented in BC.IND.relation.RST.
        - Lower and upper approximations: These approximations show whether objects can be classified with certainty or not. It is implemented in BC.LU.approximation.RST.
        - Positive region: It is used to determine objects that are included in positive region and the degree of dependency. It is implemented in BC.positive.reg.RST.
        - Discernibility matrix: It is used to create a discernibility matrix showing attributes that discern each pair of objects. It is implemented in BC.discernibility.mat.RST.

    (b) Discretization: There are a few methods included in the package:
        - D.global.discernibility.heuristic.RST: It implements the global discernibility algorithm which is computing globally semi-optimal cuts using the maximum discernibility heuristic.
        - D.discretize.quantiles.RST: It is a function used for computing cuts of the "quantile-based" discretization into n intervals.
        - D.discretize.equal.intervals.RST: It is a function used for computing cuts of the "equal interval size" discretization into n intervals.

        The output of these functions is a list of cut values which are the values for converting real to nominal values. So, in order to generate a new decision table according to the cut values, we need to call SF.applyDecTable. Additionally, we have implemented

`D.discretization.RST` as a wrapper function collecting all methods considered to perform discretization tasks.

(c) Feature selection: According to its output, it can be classified into the following groups:

- Feature subset: It refers to a superreduct which is not necessarily minimal. In other words, the methods in this group might generate just a subset of attributes.
  - QuickReduct algorithm: It has been implemented in `FS.quickreduct.RST`.
  - Superreduct generation: It is based on some criteria: entropy, gini index, discernibility measure, size of positive region.
    It is implemented in `FS.greedy.heuristic.superreduct.RST`.

  Furthermore, we provide a wrapper function `FS.feature.subset.computation` in order to give a user interface for many methods of RST and FRST that are included in this group.

- Reduct: The following are methods that produce a single decision reduct:
  - Reduct generation based on criteria: It is based on different criteria which are entropy, gini index, discernibility measure, size of positive region. It has been implemented in `FS.greedy.heuristic.reduct.RST`.
  - Permutation reduct: It is based on a permutation schema over all attributes. It has been implemented in `FS.permutation.heuristic.reduct.RST`.

  Furthermore, we provide a wrapper function `FS.reduct.computation` in order to give a user interface toward many methods of RST and FRST that are included in this group.

- All reducts: In order to generate all reducts, we execute `FS.all.reducts.computation`. However, before doing that, we need to call `BC.discernibility.mat.RST` for constructing a decision-relative discernibility matrix

It should be noted that the outputs of the functions are decision reducts. So, for generating a new decision table according to the decision reduct, we need to call `SF.applyDecTable`.

(d) Rule induction: We provide several functions used to generate rules, as follows:

- The function `RI.indiscernibilityBasedRules.RST`: This function requires the output of the feature selection functions.
- The function `RI.CN2Rules.RST`: It is a rule induction method based on the CN2 algorithm.
- The function `RI.LEM2Rules.RST`: It implements a rule induction method based on the LEM2 algorithm.
- The function `RI.AQRules.RST`: It is a rule induction based on the AQ-style algorithm.

After obtaining the rules, we execute `predict.RuleSetRST` considering our rules and given newdata/testing data to obtain predicted values/classes.

2. **The implementations of FRST**: As in the RST part, this part contains several algorithms that can be classified into several groups based on their purpose. The following is a description of all methods that have been implemented in functions:

(a) Basic concepts: The following is a list showing tasks and their implementations:

- Indiscernibility relations: they are fuzzy relations determining to which degree two objects are similar. This package provides several types of relations which are implemented in a single function called `BC.IND.relation.FRST`. We consider several types of relations e.g., fuzzy equivalence, tolerance, and T-similarity relations. These relations can be chosen by assigning type.relation. Additionally, in this function,

we provide several options to calculate aggregation e.g., triangular norm operators (e.g., "lukasiewicz", "min", etc) and user-defined operators.

- Lower and upper approximations: These approximations show to what extent objects can be classified with certainty or not. This task has been implemented in `BC.LU.approximation.FRST`. There are many approaches available in this package that can be selected by assigning the parameter type.LU. The considered methods are implication/t-norm, β-precision fuzzy rough sets (β-PFRS), vaguely quantified rough sets (VQRS), fuzzy variable precision rough sets (FVPRS), ordered weighted average (OWA), soft fuzzy rough sets (SFRS), and robust fuzzy rough sets (RFRS). Furthermore, we provide a facility, which is "custom", where users can create their own approximations by defining functions to calculate lower and upper approximations. Many options to calculate implicator and triangular norm are also available.

- Positive region: It is used to determine the membership degree of each object to the positive region and the degree of dependency. It is implemented in `BC.positive.reg.FRST`.

- Discernibility matrix: It is used to construct the decision-relative discernibility matrix. There are some approaches to construct the matrix, e.g., based on standard approach, Gaussian reduction, alpha reduction, and minimal element in discernibility matrix. They have been implemented in `BC.discernibility.mat.FRST`.

(b) Feature selection: According to the output of functions, we may divide them into three groups: those that produce a superreduct, a set of reducts, or a single reduct. The following is a description of functions based on their types:

- Feature subset: It refers to methods which produce a superreduct which is not necessarily a reduct. In other words methods in this group might generate just a subset of attributes. The following is a complete list of methods considered in this package:

  - positive region based algorithms: It refers to positive regions, as a way to evaluate attributes to be selected. They are implemented in `FS.quickreduct.FRST`. Furthermore, we provide several different measures based on the positive region in this function. All methods included in this part employ the QuickReduct algorithm to obtain selected features. In order to choose a particular algorithm, we need to assign parameter type.method in `FS.quickreduct.FRST`.

  - boundary region based algorithm: This algorithm is based on the membership degree to the fuzzy boundary region. This algorithm has been implemented in `FS.quickreduct.FRST`.

  Furthermore, we provide a wrapper function `FS.feature.subset.computation` in order to give a user interface for many methods of RST and FRST.

- Reduct: It refers to a method that produces a single decision reduct. We provide one algorithm which is the near-optimal reduction proposed by Zhao et al. It is implemented in `FS.nearOpt.fvprs.FRST`. Furthermore, we provide a wrapper function `FS.reduct.computation` in order to provide a user interface toward many methods of RST and FRST.

- All reducts: In order to get all decision reducts, we execute `FS.all.reducts.computation`. However, before doing that, we firstly execute the `BC.discernibility.mat.FRST` function for constructing a decision-relative discernibility matrix.

The output of the above methods is a class containing a decision reduct/feature subset and other descriptions. For generating a new decision table according to the decision reduct, we provide the function `SF.applyDecTable`.

(c) Rule induction: It is a task used to generate rules representing knowledge of a decision table. Commonly, this process is called learning phase in machine learning. The following methods are considered to generate rules:

- `RI.hybridFS.FRST`: It combines fuzzy-rough rule induction and feature selection.
- `RI.GFRS.FRST`: It refers to rule induction based on generalized fuzzy rough sets (GFRS).

After generating rules, we can use them to predict decision values/classes of new data by executing the S3 function `predict.RuleSetFRST`.

(d) Instance selection: The following functions select instances to improve accuracy by removing noisy, superfluous or inconsistent ones from training datasets.

- `IS.FRIS.FRST`: It refers to the fuzzy rough instance selection (FRIS). It evaluates the degree of membership to the positive region of each instance. If an instance's membership degree is less than the threshold, then the instance can be removed.
- `IS.FRPS.FRST`: It refers to the fuzzy-rough prototype selection (FRPS). It employs prototype selection (PS) to improve the accuracy of the k-nearest neighbor (kNN) method.

We provide the function `SF.applyDecTable` that is used to generate a new decision table according to the output of instance selection functions.

(e) Fuzzy-rough nearest neighbors: This part provides methods based on nearest neighbors for predicting decision values/classes of new datasets. In other words, by supplying a decision table as training data we can predict decision values of new data at the same time. We have considered the following methods:

- `C.FRNN.FRST`: It refers to the fuzzy-rough nearest neighbors based on Jensen and Cornelis' technique.
- `C.FRNN.O.FRST`: It refers to the fuzzy-rough ownership nearest neighbor algorithm based on Sarkar's method.
- `C.POSNN.FRST`: The positive region based fuzzy-rough nearest neighbor algorithm based on Verbiest et al's technique.

Furthermore, we provide an additional feature which is missing value completion. Even though algorithms, included in this feature, are not based on RST and FRST, they will be usefull to do data analysis. The following is a list of functions implemented for handling missing values in the data preprocessing step:

- `MV.deletionCases`: it refers to the approach deleting instances.
- `MV.mostCommonValResConcept`: it refers to the approach based on the most common value or mean of an attribute restricted to a concept.
- `MV.mostCommonVal`: it refers to the approach replacing missing attribute values by the attribute mean or common values.
- `MV.globalClosestFit`: it refers to the approach based on the global closest fit approach.
- `MV.conceptClosestFit`: it refers to the approach based on the concept closest fit approach.

Additionally, we provide a wrapper function which is `MV.missingValueCompletion` in order to give a user interface for the methods.

To get started with the package, the user can have a look at the examples included in the documentation on each function. Additionally, to show general usage of the package briefly, we also provide some examples showing general usage in this section.

If you have problems using the package, find a bug, or have suggestions, please contact the package maintainer by email, instead of writing to the general R lists or to other internet forums and mailing lists.

There are many demos that ship with the package. To get a list of them, type:

```
demo()
```

Then, to start a demo, type demo(<demo_name_here>). All the demos are presented as R scripts in the package sources in the "demo" subdirectory.

Currently, there are the following demos available:

- Basic concepts of RST and FRST:
  demo(BasicConcept.RST), demo(BasicConcept.FRST),
  demo(DiscernibilityMatrix.RST), demo(DiscernibilityMatrix.FRST).

- Discretization based on RST:
  demo(D.local.discernibility.matrix.RST), demo(D.max.discernibility.matrix.RST),
  demo(D.global.discernibility.heuristic.RST), demo(D.discretize.quantiles.RST),
  demo(D.discretize.equal.intervals.RST).

- Feature selection based on RST:
  demo(FS.permutation.heuristic.reduct.RST), demo(FS.quickreduct.RST),
  demo(FS.greedy.heuristic.reduct.RST), demo(FS.greedy.heuristic.reduct.RST).

- Feature selection based on FRST:
  demo(FS.QuickReduct.FRST.Ex1), demo(FS.QuickReduct.FRST.Ex2),
  demo(FS.QuickReduct.FRST.Ex3), demo(FS.QuickReduct.FRST.Ex4),
  demo(FS.QuickReduct.FRST.Ex5), demo(FS.nearOpt.fvprs.FRST).

- Instance selection based on FRST:
  demo(IS.FRIS.FRST), demo(IS.FRPS.FRST)

- Classification using the Iris dataset:
  demo(FRNN.O.iris), demo(POSNN.iris), demo(FRNN.iris).

- Rule induction based on RST:
  demo(RI.indiscernibilityBasedRules.RST).

- Rule induction based on FRST:
  demo(RI.classification.FRST), demo(RI.regression.FRST).

- Missing value completion: demo(MV.simpleData).

Some decision tables have been embedded in this package which can be seen in RoughSetData.

Finally, you may visit the package webpage http://sci2s.ugr.es/dicits/software/RoughSets, where we provide a more extensive introduction as well as additional explanations of the procedures.

Author(s)

Lala Septem Riza <lala.s.riza@decsai.ugr.es>,

Andrzej Janusz <andrzejanusz@gmail.com>,

Chris Cornelis <chriscornelis@decsai.ugr.es>,

Francisco Herrera <herrera@decsai.ugr.es>,

Dominik Slezak <slezak@mimuw.edu.pl>,

and Jose Manuel Benitez <j.m.benitez@decsai.ugr.es>

DiCITS Lab, SCI2S group, CITIC-UGR, DECSAI, University of Granada,

http://dicits.ugr.es, http://sci2s.ugr.es

Institute of Mathematics, University of Warsaw.

References

D. Dubois and H. Prade, "Rough Fuzzy Sets and Fuzzy Rough Sets", International Journal of General Systems, vol. 17, p. 91 - 209 (1990).

L.A. Zadeh, "Fuzzy Sets", Information and Control, vol. 8, p. 338 - 353 (1965).

Z. Pawlak, "Rough Sets", International Journal of Computer and Information System, vol. 11, no. 5, p. 341 - 356 (1982).

Z. Pawlak, "Rough Sets: Theoretical Aspects of Reasoning About Data, System Theory, Knowledge Engineering and Problem Solving", vol. 9, Kluwer Academic Publishers, Dordrecht, Netherlands (1991).

Examples

```
##############################################################
## A.1 Example: Basic concepts of rough set theory
##############################################################
## Using hiring data set, see RoughSetData
data(RoughSetData)
decision.table <- RoughSetData$hiring.dt

## define considered attributes which are first, second, and
## third attributes
attr.P <- c(1,2,3)

## compute indiscernibility relation
IND <- BC.IND.relation.RST(decision.table, feature.set = attr.P)

## compute lower and upper approximations
roughset <- BC.LU.approximation.RST(decision.table, IND)

## Determine regions
region.RST <- BC.positive.reg.RST(decision.table, roughset)

## The decision-relative discernibility matrix and reduct
disc.mat <- BC.discernibility.mat.RST(decision.table, range.object = NULL)
```

```
###############################################################
## A.2 Example: Basic concepts of fuzzy rough set theory
###############################################################
## Using pima7 data set, see RoughSetData
data(RoughSetData)
decision.table <- RoughSetData$pima7.dt

## In this case, let us consider the first and second attributes
conditional.attr <- c(1, 2)

## We are using the "lukasiewicz" t-norm and the "tolerance" relation
## with "eq.1" as fuzzy similarity equation
control.ind <- list(type.aggregation = c("t.tnorm", "lukasiewicz"),
                     type.relation = c("tolerance", "eq.1"))

## Compute fuzzy indiscernibility relation
IND.condAttr <- BC.IND.relation.FRST(decision.table, attributes = conditional.attr,
                             control = control.ind)

## Compute fuzzy lower and upper approximation using type.LU : "implicator.tnorm"
## Define index of decision attribute
decision.attr = c(9)

## Compute fuzzy indiscernibility relation of decision attribute
## We are using "crisp" for type of aggregation and type of relation
control.dec <- list(type.aggregation = c("crisp"), type.relation = "crisp")

IND.decAttr <- BC.IND.relation.FRST(decision.table, attributes = decision.attr,
                             control = control.dec)

## Define control parameter containing type of implicator and t-norm
control <- list(t.implicator = "lukasiewicz", t.tnorm = "lukasiewicz")

## Compute fuzzy lower and upper approximation
FRST.LU <- BC.LU.approximation.FRST(decision.table, IND.condAttr, IND.decAttr,
               type.LU = "implicator.tnorm", control = control)

## Determine fuzzy positive region and its degree of dependency
fuzzy.region <- BC.positive.reg.FRST(decision.table, FRST.LU)

###############################################################
## B Example : Data analysis based on RST and FRST
## In this example, we are using wine dataset for both RST and FRST
###############################################################
## Load the data
## Not run: data(RoughSetData)
dataset <- RoughSetData$wine.dt

## Shuffle the data with set.seed
set.seed(5)
dt.Shuffled <- dataset[sample(nrow(dataset)),]

## Split the data into training and testing
```

```
idx <- round(0.8 * nrow(dt.Shuffled))
  wine.tra <-SF.asDecisionTable(dt.Shuffled[1:idx,],
decision.attr = 14, indx.nominal = 14)
  wine.tst <- SF.asDecisionTable(dt.Shuffled[
 (idx+1):nrow(dt.Shuffled), -ncol(dt.Shuffled)])

## DISCRETIZATION
cut.values <- D.discretization.RST(wine.tra,
type.method = "global.discernibility")
d.tra <- SF.applyDecTable(wine.tra, cut.values)
d.tst <- SF.applyDecTable(wine.tst, cut.values)

## FEATURE SELECTION
red.rst <- FS.feature.subset.computation(d.tra,
  method="quickreduct.rst")
fs.tra <- SF.applyDecTable(d.tra, red.rst)

## RULE INDUCTION
rules <- RI.indiscernibilityBasedRules.RST(d.tra,
  red.rst)

## predicting newdata
pred.vals <- predict(rules, d.tst)

#################################################
## Examples: Data analysis using the wine dataset
## 2. Learning and prediction using FRST
#################################################

## FEATURE SELECTION
reduct <- FS.feature.subset.computation(wine.tra,
 method = "quickreduct.frst")

## generate new decision tables
wine.tra.fs <- SF.applyDecTable(wine.tra, reduct)
wine.tst.fs <- SF.applyDecTable(wine.tst, reduct)

## INSTANCE SELECTION
indx <- IS.FRIS.FRST(wine.tra.fs,
 control = list(threshold.tau = 0.2, alpha = 1))

## generate a new decision table
wine.tra.is <- SF.applyDecTable(wine.tra.fs, indx)

## RULE INDUCTION (Rule-based classifiers)
control.ri <- list(
 type.aggregation = c("t.tnorm", "lukasiewicz"),
 type.relation = c("tolerance", "eq.3"),
 t.implicator = "kleene_dienes")

decRules.hybrid <- RI.hybridFS.FRST(wine.tra.is,
  control.ri)
```

```
## predicting newdata
predValues.hybrid <- predict(decRules.hybrid,
  wine.tst.fs)

#################################################
## Examples: Data analysis using the wine dataset
## 3. Prediction using fuzzy nearest neighbor classifiers
#################################################

## using FRNN.O
control.frnn.o <- list(m = 2,
  type.membership = "gradual")

predValues.frnn.o <- C.FRNN.O.FRST(wine.tra.is,
  newdata = wine.tst.fs, control = control.frnn.o)

## Using FRNN
control.frnn <- list(type.LU = "implicator.tnorm",k=20,
  type.aggregation = c("t.tnorm", "lukasiewicz"),
  type.relation = c("tolerance", "eq.1"),
  t.implicator = "lukasiewicz")

predValues.frnn <- C.FRNN.FRST(wine.tra.is,
  newdata = wine.tst.fs, control = control.frnn)

## calculating error
real.val <- dt.Shuffled[(idx+1):nrow(dt.Shuffled),
  ncol(dt.Shuffled), drop = FALSE]

err.1 <- 100*sum(pred.vals!=real.val)/nrow(pred.vals)
err.2 <- 100*sum(predValues.hybrid!=real.val)/
  nrow(predValues.hybrid)
err.3 <- 100*sum(predValues.frnn.o!=real.val)/
  nrow(predValues.frnn.o)
err.4 <- 100*sum(predValues.frnn!=real.val)/
  nrow(predValues.frnn)

cat("The percentage error = ", err.1, "\n")
cat("The percentage error = ", err.2, "\n")
cat("The percentage error = ", err.3, "\n")
cat("The percentage error = ", err.4, "\n")
## End(Not run)
```

```
A.Introduction-RoughSets
```
Introduction to Rough Set Theory

Description

This part attempts to introduce rough set theory (RST) and its application to data analysis. While the classical RST proposed by Pawlak in 1982 is explained in detail in this section, some recent

advancements will be treated in the documentation of the related functions.

Details

In RST, a data set is represented as a table called an information system $\mathcal{A} = (U, A)$, where U is a non-empty set of finite objects known as the universe of discourse (note: it refers to all instances/rows in datasets) and A is a non-empty finite set of attributes, such that $a : U \rightarrow V_a$ for every $a \in A$. The set V_a is the set of values that attribute a may take. Information systems that involve a decision attribute, containing classes for each object, are called decision systems or decision tables. More formally, it is a pair $\mathcal{A} = (U, A \cup \{d\})$, where $d \notin A$ is the decision attribute. The elements of A are called conditional attributes. The information system representing all data in a particular system may contain redundant parts. It could happen because there are the same or indiscernible objects or some superfluous attributes. The indiscernibility relation is a binary relation showing the relation between two objects. This relation is an equivalence relation. Let $\mathcal{A} = (U, A)$ be an information system, then for any $B \subseteq A$ there is an equivalence relation $R_B(x, y)$:

$$R_B(x, y) = \{(x, y) \in U^2 | \forall a \in B, a(x) = a(y)\}$$

If $(x, y) \in R_B(x, y)$, then x and y are indiscernible by attributes from B. The equivalence classes of the B-indiscernibility relation are denoted $[x]_B$. The indiscernibility relation will be further used to define basic concepts of rough set theory which are lower and upper approximations.

Let $B \subseteq A$ and $X \subseteq U$, X can be approximated using the information contained within B by constructing the B-lower and B-upper approximations of X:

$$R_B \downarrow X = \{x \in U | [x]_B \subseteq X\}$$

$$R_B \uparrow X = \{x \in U | [x]_B \cap X \neq \emptyset\}$$

The tuple $\langle R_B \downarrow X, R_B \uparrow X \rangle$ is called a rough set. The objects in $R_B \downarrow X$ mean that they can be with certainty classified as members of X on the basis of knowledge in B, while the objects in $R_B \uparrow X$ can be only classified as possible members of X on the basis of knowledge in B.

In a decision system, for X we use decision concepts (equivalence classes of decision attribute) $[x]_d$. We can define B-lower and B-upper approximations as follows.

$$R_B \downarrow [x]_d = \{x \in U | [x]_B \subseteq [x]_d\}$$

$$R_B \uparrow [x]_d = \{x \in U | [x]_B \cap [x]_d \neq \emptyset\}$$

The positive, negative and boundary of B regions can be defined as:

$$POS_B = \bigcup_{x \in U} R_B \downarrow [x]_d$$

The boundary region, BND_B, is the set of objects that can possibly, but not certainly, be classified.

$$BND_B = \bigcup_{x \in U} R_B \uparrow [x]_d - \bigcup_{x \in U} R_B \downarrow [x]_d$$

Furthermore, we can calculate the degree of dependency of the decision on a set of attributes. The decision attribute d depends totally on a set of attributes B, denoted $B \Rightarrow d$, if all attribute values from d are uniquely determined by values of attributes from B. It can be defined as follows. For $B \subseteq A$, it is said that d depends on B in a degree of dependency $\gamma_B = \frac{|POS_B|}{|U|}$.

A decision reduct is a set $B \subseteq A$ such that $\gamma_B = \gamma_A$ and $\gamma_{B'} < \gamma_B$ for every $B' \subset B$. One algorithm to determine all reducts is by constructing the decision-relative discernibility matrix. The discernibility matrix $M(\mathcal{A})$ is an $n \times n$ matrix (c_{ij}) where

$c_{ij} = \{a \in A : a(x_i) \neq a(x_j)\}$ if $d(x_i) \neq d(x_j)$ and

$c_{ij} = \oslash$ otherwise

The discernibility function $f_{\mathcal{A}}$ for a decision system $\mathcal{A}$ is a boolean function of m boolean variables $\bar{a}_1, \ldots, \bar{a}_m$ corresponding to the attributes $a_1, \ldots, a_m$ respectively, and defined by

$$f_{\mathcal{A}}(\bar{a_1}, \ldots, \bar{a_m}) = \wedge\{\vee \bar{c}_{ij} : 1 \leq j < i \leq n, c_{ij} \neq \oslash\}$$

where $\bar{c}_{ij} = \{\bar{a} : a \in c_{ij}\}$. The decision reducts of A are then the prime implicants of the function $f_{\mathcal{A}}$. The complete explanation of the algorithm can be seen in (Skowron and Rauszer, 1992).

The implementations of the RST concepts can be seen in `BC.IND.relation.RST`, `BC.LU.approximation.RST`, `BC.positive.reg.RST`, and `BC.discernibility.mat.RST`.

References

A. Skowron and C. Rauszer, "The Discernibility Matrices and Functions in Information Systems", in: R. Slowinski (Ed.), Intelligent Decision Support: Handbook of Applications and Advances of Rough Sets Theory, Kluwer Academic Publishers, Dordrecht, Netherland, p. 331 - 362 (1992).

Z. Pawlak, "Rough Sets", International Journal of Computer and Information System, vol. 11, no.5, p. 341 - 356 (1982).

Z. Pawlak, "Rough Sets: Theoretical Aspects of Reasoning about Data, System Theory, Knowledge Engineering and Problem Solving", vol. 9, Kluwer Academic Publishers, Dordrecht, Netherlands (1991).

---

as.character.RuleSetRST

The `as.character` *method for RST rule sets*

Description

A function for converting a set of rules into their character representation.

Usage

```
## S3 method for class 'RuleSetRST'
as.character(x, ...)
```

Arguments

x	a *"RuleSetRST"* object. See `RI.LEM2Rules.RST`.
...	the other parameters.

Value

Converts rules from a set into their character representation.

Author(s)

Andrzej Janusz

Examples

```
############################################################
## Example : Converting a set of decision rules
############################################################
data(RoughSetData)
hiring.data <- RoughSetData$hiring.dt

rules <- RI.LEM2Rules.RST(hiring.data)

as.character(rules)
```

---

as.list.RuleSetRST   *The* as.list *method for RST rule sets*

Description

A function for converting a set of rules into a list.

Usage

```
## S3 method for class 'RuleSetRST'
as.list(x, ...)
```

Arguments

x   a "RuleSetRST" object. See RI.LEM2Rules.RST.

...   the other parameters.

Value

Converts rules from a set into a list.

Author(s)

Andrzej Janusz

Examples

```
############################################################
## Example : Converting a set of decision rules
############################################################
data(RoughSetData)
hiring.data <- RoughSetData$hiring.dt

rules <- RI.LEM2Rules.RST(hiring.data)

as.list(rules)
```

`B.Introduction-FuzzyRoughSets`

Introduction to Fuzzy Rough Set Theory

Description

This part introduces briefly fuzzy rough set theory (FRST) and its application to data analysis. Since recently there are a lot of FRST variants that have been proposed by researchers, in this introduction, we only provide some basic concepts of FRST based on (Radzikowska and Kerre, 2002).

Details

Just like in RST (see `A.Introduction-RoughSets`), a data set is represented as a table called an information system $\mathcal{A} = (U, A)$, where U is a non-empty set of finite objects as the universe of discourse (note: it refers to all instances/experiments/rows in datasets) and A is a non-empty finite set of attributes, such that $a : U \to V_a$ for every $a \in A$. The set V_a is the set of values that attribute a may take. Information systems that involve a decision attribute, containing classes or decision values of each objects, are called decision systems (or said as decision tables). More formally, it is a pair $\mathcal{A} = (U, A \cup \{d\})$, where $d \notin A$ is the decision attribute. The elements of A are called conditional attributes. However, different from RST, FRST has several ways to express indiscernibility.

Fuzzy indiscernibility relation (FIR) is used for any fuzzy relation that determines the degree to which two objects are indiscernible. We consider some special cases of FIR.

- fuzzy tolerance relation: this relation has properties which are reflexive and symmetric where
  reflexive: $R(x, x) = 1$
  symmetric: $R(x, y) = R(y, x)$

- similarity relation (also called fuzzy equivalence relation): this relation has properties not only reflexive and symmetric but also transitive defined as
  $min(R(x, y), R(y, z)) \leq R(x, z)$

- $\mathcal{T}$-similarity relation (also called fuzzy $\mathcal{T}$-equivalence relation): this relation is a fuzzy tolerance relation that is also $\mathcal{T}$-transitive.
  $\mathcal{T}(R(x, y), R(y, z)) \leq R(x, z)$, for a given triangular norm $\mathcal{T}$.

The following equations are the tolerance relations on a quantitative attribute a, R_a, proposed by (Jensen and Shen, 2009).

- `eq.1`: $R_a(x, y) = 1 - \frac{|a(x) - a(y)|}{|a_{max} - a_{min}|}$

- `eq.2`: $R_a(x, y) = exp(-\frac{(a(x) - a(y))^2}{2\sigma_a^2})$

- `eq.3`: $R_a(x, y) = max(min(\frac{a(y) - a(x) + \sigma_a}{\sigma_a}, \frac{a(x) - a(y) + \sigma_a}{\sigma_a}), 0)$

where σ_a^2 is the variance of feature a and a_{min} and a_{max} are the minimal and maximal values of data supplied by user. Additionally, other relations have been implemented in `BC.IND.relation.FRST`

For a qualitative (i.e., nominal) attribute a, the classical manner of discerning objects is used, i.e., $R_a(x, y) = 1$ if $a(x) = a(y)$ and $R_a(x, y) = 0$, otherwise. We can then define, for any subset B of A, the fuzzy B-indiscernibility relation by

$$R_B(x, y) = \mathcal{T}(R_a(x, y)),$$

where $\mathcal{T}$ is a t-norm operator, for instance minimum, product and Lukasiewicz t-norm. In general, $\mathcal{T}$ can be replaced by any aggregation operator, like e.g., the average.

In the context of FRST, according to (Radzikowska and Kerre, 2002) lower and upper approximation are generalized by means of an implicator $\mathcal{I}$ and a t-norm $\mathcal{T}$. The following are the fuzzy B-lower and B-upper approximations of a fuzzy set A in U

$$(R_B \downarrow A)(y) = inf_{x \in U}\mathcal{I}(R_B(x, y), A(x))$$

$$(R_B \uparrow A)(y) = sup_{x \in U}\mathcal{T}(R_B(x, y), A(x))$$

The underlying meaning is that $R_B \downarrow A$ is the set of elements *necessarily* satisfying the concept (strong membership), while $R_B \uparrow A$ is the set of elements *possibly* belonging to the concept (weak membership). Many other ways to define the approximations can be found in `BC.LU.approximation.FRST`. Mainly, these were designed to deal with noise in the data and to make the approximations more robust.

Based on fuzzy B-indiscernibility relations, we define the fuzzy B-positive region by, for $y \in X$,

$$POS_B(y) = (\cup_{x \in U}R_B \downarrow R_d x)(y)$$

We can define the degree of dependency of d on B, γ_B by

$$\gamma_B = \frac{|POS_B|}{|U|} = \frac{\sum_{x \in U}POS_B(x)}{|U|}$$

A decision reduct is a set $B \subseteq A$ such that $\gamma_B = \gamma_A$ and $\gamma_{B'} = \gamma_B$ for every $B' \subset B$.

As we know from rough set concepts (See `A.Introduction-RoughSets`), we are able to calculate the decision reducts by constructing the decision-relative discernibility matrix. Based on (Tsang et al, 2008), the discernibility matrix can be defined as follows. The discernibility matrix is an $n \times n$ matrix (c_{ij}) where for $i, j = 1, \ldots, n$

1) $c_{ij} = \{a \in A : 1 - R_a(x_i, x_j) \geq \lambda_i\}$ if $\lambda_j < \lambda_i$.

2) $c_{ij} = \oslash$, otherwise.

with $\lambda_i = (R_A \downarrow R_d x_i)(x_i)$ and $\lambda_j = (R_A \downarrow R_d x_j)(x_j)$

Other approaches of discernibility matrix can be read at `BC.discernibility.mat.FRST`.

The other implementations of the FRST concepts can be seen at `BC.IND.relation.FRST`, `BC.LU.approximation.FRST`, and `BC.positive.reg.FRST`.

References

A. M. Radzikowska and E. E. Kerre, "A Comparative Study of Fuzzy Rough Sets", Fuzzy Sets and Systems, vol. 126, p. 137 - 156 (2002).

D. Dubois and H. Prade, "Rough Fuzzy Sets and Fuzzy Rough Sets", International Journal of General Systems, vol. 17, p. 91 - 209 (1990).

E. C. C. Tsang, D. G. Chen, D. S. Yeung, X. Z. Wang, and J. W. T. Lee, "Attributes Reduction Using Fuzzy Rough Sets", IEEE Trans. Fuzzy Syst., vol. 16, no. 5, p. 1130 - 1141 (2008).

L. A. Zadeh, "Fuzzy Sets", Information and Control, vol. 8, p. 338 - 353 (1965).

R. Jensen and Q. Shen, "New Approaches to Fuzzy-Rough Feature Selection", IEEE Trans. on Fuzzy Systems, vol. 19, no. 4, p. 824 - 838 (2009).

Z. Pawlak, "Rough Sets", International Journal of Computer and Information Sciences, vol. 11, no. 5, p. 341 - 356 (1982).

BC.boundary.reg.RST   *Computation of a boundary region*

Description

This function implements a fundamental part of RST: computation of a boundary region and the degree of dependency. This function can be used as a basic building block for development of other RST-based methods. A more detailed explanation of this notion can be found in `A.Introduction-RoughSets`.

Usage

```
BC.boundary.reg.RST(decision.table, roughset)
```

Arguments

decision.table   an object inheriting from the ″DecisionTable″ class, which represents a decision system. See `SF.asDecisionTable`.

roughset   an object inheriting from the ″LowerUpperApproximation″ class, which represents lower and upper approximations of decision classes in the data. Such objects are typically produced by calling the `BC.LU.approximation.RST` function.

Value

An object of a class ″BoundaryRegion″ which is a list with the following components:

- boundary.reg: an integer vector containing indices of data instances belonging to the boundary region,
- degree.dependency: a numeric value giving the degree of dependency,
- type.model: a varacter vector identifying the utilized model. In this case, it is ″RST″ which means the rough set theory.

Author(s)

Dariusz Jankowski, Andrzej Janusz

References

Z. Pawlak, "Rough Sets", International Journal of Computer and Information Sciences, vol. 11, no. 5, p. 341 - 356 (1982).

See Also

BC.IND.relation.RST, BC.LU.approximation.RST, BC.LU.approximation.FRST

Examples

```
#########################################################
data(RoughSetData)
hiring.data <- RoughSetData$hiring.dt

## We select a single attribute for computation of indiscernibility classes:
A <- c(2)

## compute the indiscernibility classes:
IND.A <- BC.IND.relation.RST(hiring.data, feature.set = A)

## compute the lower and upper approximation:
roughset <- BC.LU.approximation.RST(hiring.data, IND.A)

## get the boundary region:
pos.boundary = BC.boundary.reg.RST(hiring.data, roughset)
pos.boundary
```

BC.discernibility.mat.FRST

The decision-relative discernibility matrix based on fuzzy rough set theory

---

Description

This is a function that is used to build the decision-relative discernibility matrix based on FRST. It is a matrix whose elements contain discernible attributes among pairs of objects. By means of this matrix, we are able to produce all decision reducts of the given decision system.

Usage

```
BC.discernibility.mat.FRST(decision.table,
  type.discernibility = "standard.red", control = list())
```

Arguments

decision.table   a "DecisionTable" class representing the decision table. See SF.asDecisionTable.
                  It should be noted that this case only supports the nominal/symbolic decision at-
                  tribute.
type.discernibility
                  a string representing a type of discernibility. See in Section Details.
control   a list of other parameters.

- type.relation: a type of fuzzy indiscernibility relation. The default value is type.relation = c("tolerance", "eq.1").
  See `BC.IND.relation.FRST`.
- type.aggregation: a type of aggregation operator. The default value is type.aggregation = c("t.tnorm", "lukasiewicz").
  See `BC.IND.relation.FRST`.
- t.implicator: a type of implicator operator. The default value is "lukasiewicz".
  See `BC.LU.approximation.FRST`.
- type.LU: a type of method of lower and upper approximations. The default value is "implicator.tnorm".
  See `BC.LU.approximation.FRST`.
- alpha.precision: a numeric value representing a precision variable. It is used when using "alpha.red" as type.discernibility. The default value is 0.05.
  See `BC.LU.approximation.FRST`.
- show.discernibilityMatrix: a boolean value determining whether the discernibility matrix will be shown or not (NULL). The default value is FALSE.
- epsilon: a numeric between 0 and 1 representing the ϵ value on type.discernibility = "gaussian.red". It should be noted that when having nominal values on all attributes then ϵ should be 0. The default value is 0.
- delta: a numeric representing the δ value on "gaussian" equations (see `BC.IND.relation.FRST`). The default value is 2.
- range.object: a vector representing considered objects to construct the k-relative discernibility matrix. The default value is NULL which means that we are using all objects in the decision table.

Details

In this function, we provide several approaches in order to generate the decision-relative discernibility matrix. Theoretically, all reducts are found by constructing the matrix that contains elements showing discernible attributes among objects. The discernible attributes are determined by a specific condition which depends on the selected algorithm. A particular approach can be executed by selecting a value of the parameter type.discernibility. The following shows the different values of the parameter type.discernibility corresponding approaches considered in this function.

- "standard.red": It is adopted from (Tsang et al, 2008)'s approach. The concept has been explained briefly in `B.Introduction-FuzzyRoughSets`. In order to use this algorithm, we assign the control parameter with the following components:

  control = list(type.aggregation, type.relation, type.LU, t.implicator)

  The detailed description of the components can be seen in `BC.IND.relation.FRST` and

  `BC.LU.approximation.FRST`. Furthermore, in this case the authors suggest to use the "min" t-norm (i.e., type.aggregation = c("t.tnorm", "min")) and the implicator operator "kleene_dienes" (i.e., t.implicator = "kleene_dienes").

- "alpha.red": It is based on (Zhao et al, 2009)'s approach where all reductions will be found by building an α-discernibility matrix. This matrix contains elements which are defined by

1) if x_i and x_j belong to different decision concept,

$c_{ij} = \{R : \mathcal{T}(R(x_i, x_j), \lambda) \leq \alpha\}$,

where $\lambda = (R_\alpha \downarrow A)(u)$ which is lower approximation of FVPRS (See `BC.LU.approximation.FRST`).

2) $c_{ij} = \oslash$, otherwise.

To generate the discernibility matrix based on this approach, we use the `control` parameter with the following components:

`control = list(type.aggregation, type.relation, t.implicator, alpha.precision)`

where the lower approximation λ is fixed to `type.LU = "fvprs"`. The detailed description of the components can be seen in `BC.IND.relation.FRST` and `BC.LU.approximation.FRST`. Furthermore, in this case the authors suggest to use $\mathcal{T}$-similarity relation

e.g., `type.relation = c("transitive.closure", "eq.3")`,

the "lukasiewicz" t-norm (i.e., `type.aggregation = c("t.tnorm", "lukasiewicz")`), and `alpha.precision` from 0 to 0.5.

- `"gaussian.red"`: It is based on (Chen et al, 2011)'s approach. The discernibility matrix contains elements which are defined by:

  1) if x_i and x_j belong to different decision concept,

  $c_{ij} = \{R : R(x_i, x_j) \leq \sqrt{1 - \lambda^2(x_i)}\}$,

  where $\lambda = inf_{u \in U} \mathcal{I}_{cos}(R(x_i, u), A(u)) - \epsilon$. To generate fuzzy relation R, we use the fixed parameters as follows:

  `t.tnorm = "t.cos"` and `type.relation = c("transitive.kernel", "gaussian")`.

  2) $c_{ij} = \oslash$, otherwise.

  In this case, we need to define `control` parameter as follows.

  `control <- list(epsilon)`

  It should be noted that when having nominal values on all attributes then `epsilon` (ϵ) should be 0.

- `"min.element"`: It is based on (Chen et al, 2012)'s approach where we only consider finding the minimal element of the discernibility matrix by introducing the binary relation $DIS(R)$ the relative discernibility relation of conditional attribute R with respect to decision attribute d, which is computed as

  $DIS(R) = \{(x_i, x_j) \in U \times U : 1 - R(x_i, x_j) > \lambda_i, x_j \notin [x_i]_d\}$,

  where $\lambda_i = (Sim(R) \downarrow [x_i]_d)(x_i)$ with $Sim(R)$ a fuzzy equivalence relation. In other words, this algorithm does not need to build the discernibility matrix. To generate the fuzzy relation R and lower approximation λ, we use the `control` parameter with the following components:

  `control = list(type.aggregation, type.relation, type.LU, t.implicator)`.

  The detailed description of the components can be seen in `BC.IND.relation.FRST` and `BC.LU.approximation.FRST`.

Value

A class `"DiscernibilityMatrix"` containing the following components:

- `disc.mat`: a matrix showing the decision-relative discernibility matrix $M(\mathcal{A})$ which contains $n \times n$ where n is the number of objects. It will be printed when choosing `show.discernibilityMatrix = TRUE`.

- `disc.list`: the decision-relative discernibility represented in a list.

- discernibility.type: a string showing the chosen type of discernibility methods.
- type.model: in this case, it is "FRST".

Author(s)

Lala Septem Riza

References

D. Chen, L. Zhang, S. Zhao, Q. Hu, and P. Zhu, "A Novel Algorithm for Finding Reducts with Fuzzy Rough Sets", IEEE Trans. on Fuzzy Systems, vol. 20, no. 2, p. 385 - 389 (2012).

D. G. Chen, Q. H. Hu, and Y. P. Yang, "Parameterized Attribute Reduction with Gaussian Kernel Based Fuzzy Rough Sets", Information Sciences, vol. 181, no. 23, p. 5169 - 5179 (2011).

E. C. C. Tsang, D. G. Chen, D. S. Yeung, X. Z. Wang, and J. W. T. Lee, "Attributes Reduction Using Fuzzy Rough Sets", IEEE Trans. Fuzzy Syst., vol. 16, no. 5, p. 1130 - 1141 (2008).

S. Zhao, E. C. C. Tsang, and D. Chen, "The Model of Fuzzy Variable Precision Rough Sets", IEEE Trans. on Fuzzy Systems, vol. 17, no. 2, p. 451 - 467 (2009).

See Also

BC.discernibility.mat.RST, BC.LU.approximation.RST, and BC.LU.approximation.FRST

Examples

```
###########################################################################
## Example 1: Constructing the decision-relative discernibility matrix
## In this case, we are using The simple Pima dataset containing 7 rows.
###########################################################################
data(RoughSetData)
decision.table <- RoughSetData$pima7.dt

## using "standard.red"
control.1 <- list(type.relation = c("tolerance", "eq.1"),
                  type.aggregation = c("t.tnorm", "min"),
                  t.implicator = "kleene_dienes", type.LU = "implicator.tnorm")
res.1 <- BC.discernibility.mat.FRST(decision.table, type.discernibility = "standard.red",
                                    control = control.1)

## using "gaussian.red"
control.2 <- list(epsilon = 0)
res.2 <- BC.discernibility.mat.FRST(decision.table, type.discernibility = "gaussian.red",
                                    control = control.2)

## using "alpha.red"
control.3 <- list(type.relation = c("tolerance", "eq.1"),
                  type.aggregation = c("t.tnorm", "min"),
                  t.implicator = "lukasiewicz", alpha.precision = 0.05)
res.3 <- BC.discernibility.mat.FRST(decision.table, type.discernibility = "alpha.red",
                                    control = control.3)
```

```
## using "min.element"
control.4 <- list(type.relation = c("tolerance", "eq.1"),
                  type.aggregation = c("t.tnorm", "lukasiewicz"),
                  t.implicator = "lukasiewicz", type.LU = "implicator.tnorm")
res.4 <- BC.discernibility.mat.FRST(decision.table, type.discernibility = "min.element",
                                    control = control.4)

#######################################################################
## Example 2: Constructing the decision-relative discernibility matrix
## In this case, we are using the Hiring dataset containing nominal values
#######################################################################
## Not run: data(RoughSetData)
decision.table <- RoughSetData$hiring.dt

control.1 <- list(type.relation = c("crisp"),
                  type.aggregation = c("crisp"),
                  t.implicator = "lukasiewicz", type.LU = "implicator.tnorm")
res.1 <- BC.discernibility.mat.FRST(decision.table, type.discernibility = "standard.red",
                                    control = control.1)

control.2 <- list(epsilon = 0)
res.2 <- BC.discernibility.mat.FRST(decision.table, type.discernibility = "gaussian.red",
                                    control = control.2)
## End(Not run)
```

BC.discernibility.mat.RST

Computation of a decision-relative discernibility matrix based on the rough set theory

---

Description

This function implements a fundamental part of RST: a decision-relative discernibility matrix. This notion was proposed by (Skowron and Rauszer, 1992) as a middle-step in many RST algorithms for computaion of reducts, discretization and rule induction. A more detailed explanation of this notion can be found in A.Introduction-RoughSets.

Usage

```
BC.discernibility.mat.RST(decision.table, range.object = NULL,
  return.matrix = FALSE, attach.data = FALSE)
```

Arguments

decision.table an object inheriting from the DecisionTable class, which represents a decision
                system. See SF.asDecisionTable.

range.object   an integer vector indicating objects for construction of the k-relative discernibil-
                ity matrix. The default value is NULL which means that all objects in the decision
                table are used.

return.matrix	a logical value determining whether the discernibility matrix should be retunred in the output. If it is set to FALSE (the default) only a list containing unique clauses from the CNF representation of the discernibility function is returned.
attach.data	a logical indicating whether the original decision table should be attached as an additional element of the resulting list named as dec.table.

Value

An object of a class DiscernibilityMatrix which has the following components:

- disc.mat: the decision-relative discernibility matrix which for pairs of objects from different decision classes stores names of attributes which can be used to discern between them. Only pairs of objects from different decision classes are considered. For other pairs the disc.mat contains NA values. Moreover, since the classical discernibility matrix is symmetric only the pairs from the lower triangular part are considered.

- disc.list: a list containing unique clauses from the CNF representation of the discernibility function,

- dec.table: an object of a class DecisionTable, which was used to compute the discernibility matrix,

- discernibility.type: a type of discernibility relation used in the computations,

- type.model: a character vector identifying the type of model which was used. In this case, it is "RST" which means the rough set theory.

Author(s)

Lala Septem Riza and Andrzej Janusz

References

A. Skowron and C. Rauszer, "The Discernibility Matrices and Functions in Information Systems", in: R. Słowinski (Ed.), Intelligent Decision Support: Handbook of Applications and Advances of Rough Sets Theory, Kluwer Academic Publishers, Dordrecht, Netherland, p. 331 - 362 (1992).

See Also

BC.IND.relation.RST, BC.LU.approximation.RST, BC.LU.approximation.FRST and BC.discernibility.mat.FRST

Examples

```
#########################################################################
## Example 1: Constructing the decision-relative discernibility matrix
#########################################################################
data(RoughSetData)
hiring.data <- RoughSetData$hiring.dt

## building the decision-relation discernibility matrix
disc.matrix <- BC.discernibility.mat.RST(hiring.data, return.matrix = TRUE)
disc.matrix
```

BC.IND.relation.FRST   *The indiscernibility relation based on fuzzy rough set theory*

Description

This is a function used to implement a fundamental concept of FRST which is fuzzy indiscernibility relations. It is used for any fuzzy relations that determine the degree to which two objects are indiscernibility. The detailed description about basic concepts of FRST can be found in B.Introduction-FuzzyRoughSets.

Usage

```
BC.IND.relation.FRST(decision.table, attributes = NULL,
  control = list())
```

Arguments

decision.table   a "DecisionTable" class representing a decision table. See SF.asDecisionTable.

attributes   a numerical vector expressing indexes of subset of attributes to be considered. The default value is NULL which means that all conditional attributes will be considered.

control   a list of other parameters consisting of the following parameters:

- type.relation: a list containing string values that express the type of the fuzzy relation and its equation. The default value is type.relation = c("tolerance", "eq.1"). See in the Section Details.
- type.aggregation: a list expressing type of aggregation. The default value is type.aggregation = c("t.tnorm", "lukasiewicz"). See in the Section Details.

Details

Briefly, the indiscernibility relation is a relation that shows a degree of similarity among the objects. For example, $R(x_i, x_j) = 0$ means the object x_i is completely different from x_j, and otherwise if $R(x_i, x_j) = 1$, while between those values we consider a degree of similarity. To calculate this relation, several methods have been implemented in this function which are approaches based on fuzzy tolerance, equivalence and T-equivalence relations. The fuzzy tolerance relations proposed by (Jensen and Shen, 2009) include three equations while (Hu, 2004) proposed five T_{cos}-transitive kernel functions as fuzzy T-equivalence relations. The simple algorithm of fuzzy equivalence relation is implemented as well. Furthermore, we facilitate users to define their own equation for similarity relation.

To calculate a particular relation, we should pay attention to several components in the parameter control. The main component in the control parameter is type.relation that defines what kind of approach we are going to use. The detailed explanation about the parameters and their equations is as follows:

- "tolerance": It refers to fuzzy tolerance relations proposed by (Jensen and Shen, 2009). In order to represent the "tolerance" relation, we must set type.relation as follows:

  type.relation = c("tolerance", <chosen equation>)

  where the chosen equation called as t.similarity is one of the "eq.1", "eq.2", and "eq.3" equations which have been explained in B.Introduction-FuzzyRoughSets.

- "transitive.kernel": It refers to the relations employing kernel functions (Genton, 2001). In order to represent the relation, we must set the type.relation parameter as follows.

  type.relation = c("transitive.kernel", <chosen equation>, <delta>)

  where the chosen equation is one of five following equations (called t.similarity):

  - "gaussian": It means Gaussian kernel which is $R_G(x, y) = \exp(-\frac{|x-y|^2}{\delta})$
  - "exponential": It means exponential kernel which is $R_E(x, y) = \exp(-\frac{|x-y|}{\delta})$
  - "rational": It means rational quadratic kernel which is $R_R(x, y) = 1 - \frac{|x-y|^2}{|x-y|^2+\delta}$
  - "circular": It means circular kernel which is if $|x-y| < \delta$, $R_C(x, y) = \frac{2}{\pi} \arccos(\frac{|x-y|}{\delta}) - \frac{2}{\pi} \frac{|x-y|}{\delta} \sqrt{1 - (\frac{|x-y|}{\delta})^2}$
  - "spherical": It means spherical kernel which is if $|x-y| < \delta$, $R_S(x, y) = 1 - \frac{3}{2} \frac{|x-y|}{\delta} + \frac{1}{2}(\frac{|x-y|}{\delta})^3$

  and delta is a specified value. For example: let us assume we are going to use "transitive.kernel" as the fuzzy relation, "gaussian" as its equation and the delta is 0.2. So, we assign the type.relation parameter as follows:

  type.relation = c("transitive.kernel", "gaussian", 0.2)

  If we omit the delta parameter then we are using "gaussian" defined as $R_E(x, y) = \exp(-\frac{|x-y|}{2\sigma^2})$, where σ is the variance. Furthermore, when using this relation, usually we set

  type.aggregation = c("t.tnorm", "t.cos").

- "kernel.frst": It refers to T-equivalence relation proposed by (Hu, 2004). In order to represent the relation, we must set type.relation parameter as follows.

  type.relation = c("kernel.frst", <chosen equation>, <delta>)

  where the chosen equation is one of the kernel functions, but they have different names corresponding to previous ones: "gaussian.kernel", "exponential.kernel", "rational.kernel", "circular.kernel", and "spherical.kernel". And delta is a specified value. For example: let us assume we are going to use "gaussian.kernel" as its equation and the delta is 0.2. So, we assign the type.relation parameter as follows:

  type.relation = c("kernel.frst", "gaussian.kernel", 0.2)

  In this case, we do not need to define type of aggregation. Furthemore, regarding the distance used in the equations if objects x and y contains mixed values (nominal and continuous) then we use the Gower distance and we use the euclidean distance for continuous only.

- "transitive.closure": It refers to similarity relation (also called fuzzy equivalence relation). We consider a simple algorithm to calculate this relation as follows:

  Input: a fuzzy relation R

  Output: a min-transitive fuzzy relation R^m

  Algorithm:

  1. For every x, y: compute

  $R'(x, y) = max(R(x, y), max_{z \in U} min(R(x, z), R(z, y)))$

2. If $R' \neq R$, then $R \leftarrow R'$ and go to 1, else $R^m \leftarrow R'$

For interested readers, other algorithms can be seen in (Naessens et al, 2002). Let `"eq.1"` be the R fuzzy relations, to define it as parameter is

`type.relation = c("transitive.closure", "eq.1")`. We can also use other equations that have been explained in `"tolerance"` and `"transitive.kernel"`.

- `"crisp"`: It uses crisp equality for all attributes and we set the parameter `type.relation =` `"crisp"`. In this case, we only have $R(x_i, x_j) = 0$ which means the object x_i is completely different from x_j, and otherwise if $R(x_i, x_j) = 1$.

- `"custom"`: this value means that we define our own equation for the indiscernibility relation. The equation should be defined in parameter `FUN.relation`.

  `type.relation = c("custom", <FUN.relation>)`

  The function `FUN.relation` should consist of three arguments which are `decision.table`, `x`, and `y`, where `x` and `y` represent two objects which we want to compare. It should be noted that the user must ensure that the values of this equation are always between 0 and 1. An example can be seen in Section `Examples`.

Beside the above `type.relation`, we provide several options of values for the `type.aggregation` parameter. The following is a description about it.

- `type.aggregation = c("crisp")`: It uses crisp equality for all attributes.

- `type.aggregation = c("t.tnorm", <t.tnorm operator>)`: It means we are using `"t.tnorm"` aggregation which is a triangular norm operator with a specified operator `t.tnorm` as follows:

  - `"min"`: standard t-norm i.e., $min(x_1, x_2)$.
  - `"hamacher"`: hamacher product i.e., $(x_1 * x_2)/(x_1 + x_2 - x_1 * x_2)$.
  - `"yager"`: yager class i.e., $1 - min(1, ((1 - x_1) + (1 - x_2)))$.
  - `"product"`: product t-norm i.e., $(x_1 * x_2)$.
  - `"lukasiewicz"`: lukasiewicz's t-norm (default) i.e., $max(x_2 + x_1 - 1, 0)$.
  - `"t.cos"`: T_{cos}t-norm i.e., $max(x_1 * x_2 - \sqrt{1 - x_1^2}\sqrt{1 - x_2^2}, 0)$.
  - `FUN.tnorm`: It is a user-defined function for `"t.tnorm"`. It has to have two arguments, for example:
    `FUN.tnorm <- function(left.side, right.side)`
    `if ((left.side + right.side) > 1)`
    `return(min(left.side, right.side))`
    `else return(0)`

  The default value is `type.aggregation = c("t.tnorm", "lukasiewicz")`.

- `type.aggregation = c("custom", <FUN.agg>)`: It is used to define our own function for a type of aggregations. `<FUN.agg>` is a function having one argument representing data that is produced by fuzzy similarity equation calculation. The data is a list of one or many matrices which depend on the number of considered attributes and has dimension: the number of object $\times$ the number of object. For example:

  `FUN.agg <- function(data) return(Reduce("+", data)/length(data))`

  which is a function to calculate average along all attributes. Then, we can set `type.aggregation` as follows:

  `type.aggregation = c("general.custom", <FUN.agg>)`. An example can be seen in Section `Examples`.

Furthermore, the use of this function has been illustrated in Section Examples. Finally, this function is important since it is a basic function needed by other functions, such as BC.LU.approximation.FRST and BC.positive.reg.FRST for calculating lower and upper approximation and determining positive regions.

Value

A class ″IndiscernibilityRelation″ which contains

- IND.relation: a matrix representing the indiscernibility relation over all objects.
- type.relation: a vector representing the type of relation.
- type.aggregation: a vector representing the type of aggregation operator.
- type.model: a string showing the type of model which is used. In this case it is ″FRST″ which means fuzzy rough set theory.

Author(s)

Lala Septem Riza

References

M. Genton, "Classes of Kernels for Machine Learning: a Statistics Perspective", J. Machine Learning Research, vol. 2, p. 299 - 312 (2001).

H. Naessens, H. De Meyer, and B. De Baets, "Algorithms for the Computation of T-Transitive Closures", IEEE Trans. on Fuzzy Systems, vol. 10, No. 4, p. 541 - 551 (2002).

R. Jensen and Q. Shen, "New Approaches to Fuzzy-Rough Feature Selection", IEEE Trans. on Fuzzy Systems, vol. 19, no. 4, p. 824 - 838 (2009).

Q. Hu, D. Yu, W. Pedrycz, and D. Chen, "Kernelized Fuzzy Rough Sets and Their Applications", IEEE Trans. Knowledge Data Eng., vol. 23, no. 11, p. 1649 - 1471 (2011).

See Also

BC.LU.approximation.FRST, BC.IND.relation.RST, BC.LU.approximation.RST,

and BC.positive.reg.FRST

Examples

```
#############################################################
## Example 1: Dataset containing nominal values for
## all attributes.
#############################################################
## Decision table is represented as data frame
dt.ex1 <- data.frame(c(1,0,2,1,1,2,2,0), c(0, 1,0, 1,0,2,1,1),
                      c(2,1,0,0,2,0,1,1), c(2,1,1,2,0,1,1,0), c(0,2,1,2,1,1,2,1))
colnames(dt.ex1) <- c(″aa″, ″bb″, ″cc″, ″dd″, ″ee″)
decision.table <- SF.asDecisionTable(dataset = dt.ex1, decision.attr = 5,
     indx.nominal = c(1:5))

## In this case, we only consider the second and third attributes.
```

```
attributes <- c(2, 3)

## calculate fuzzy indiscernibility relation ##
## in this case, we are using "crisp" as a type of relation and type of aggregation
control.ind <- list(type.relation = c("crisp"), type.aggregation = c("crisp"))
IND <- BC.IND.relation.FRST(decision.table, attributes = attributes, control = control.ind)

###########################################################
## Example 2: Dataset containing real-valued attributes
###########################################################
dt.ex2 <- data.frame(c(-0.4, -0.4, -0.3, 0.3, 0.2, 0.2),
                     c(-0.3, 0.2, -0.4, -0.3, -0.3, 0),
       c(-0.5, -0.1, -0.3, 0, 0, 0),
       c("no", "yes", "no", "yes", "yes", "no"))
colnames(dt.ex2) <- c("a", "b", "c", "d")
decision.table <- SF.asDecisionTable(dataset = dt.ex2, decision.attr = 4)

## in this case, we only consider the first and second attributes
attributes <- c(1, 2)

## Calculate fuzzy indiscernibility relation
## in this case, we choose "tolerance" relation and "eq.1" as similarity equation
## and "lukasiewicz" as t-norm of type of aggregation
control.1 <- list(type.aggregation = c("t.tnorm", "lukasiewicz"),
                    type.relation = c("tolerance", "eq.1"))
IND.1 <- BC.IND.relation.FRST(decision.table, attributes = attributes,
                               control = control.1)

## Calculate fuzzy indiscernibility relation: transitive.kernel
control.2 <- list(type.aggregation = c("t.tnorm", "t.cos"),
                    type.relation = c("transitive.kernel", "gaussian", 0.2))
IND.2 <- BC.IND.relation.FRST(decision.table, attributes = attributes,
                               control = control.2)

## Calculate fuzzy indiscernibility relation: kernel.frst
control.3 <- list(type.relation = c("kernel.frst", "gaussian.kernel", 0.2))
IND.3 <- BC.IND.relation.FRST(decision.table, attributes = attributes,
                               control = control.3)

## calculate fuzzy transitive closure
control.4 <- list(type.aggregation = c("t.tnorm", "lukasiewicz"),
                    type.relation = c("transitive.closure", "eq.1"))
IND.4 <- BC.IND.relation.FRST(decision.table, attributes = attributes,
                               control = control.4)

## Calculate fuzzy indiscernibility relation: using user-defined relation
## The customized function should have three arguments which are : decision.table
## and object x, and y.
## This following example shows that we have an equation for similarity equation:
## 1 - abs(x - y) where x and y are two objects that will be compared.
## In this case, we do not consider decision.table in calculation.
FUN.relation <- function(decision.table, x, y) {
          return(1 - (abs(x - y)))
```

```
        }
control.5 <- list(type.aggregation = c("t.tnorm", "lukasiewicz"),
                    type.relation = c("custom", FUN.relation))
IND.5 <- BC.IND.relation.FRST(decision.table, attributes = attributes,
                              control = control.5)

## In this case, we calculate aggregation as average of all objects
## by executing the Reduce function
FUN.average <- function(data){
    return(Reduce("+", data)/length(data))
}
control.6 <- list(type.aggregation = c("custom", FUN.average),
                    type.relation = c("tolerance", "eq.1"))
IND.6 <- BC.IND.relation.FRST(decision.table, attributes = attributes,
                              control = control.6)

## using user-defined tnorms
FUN.tnorm <- function(left.side, right.side) {
                if ((left.side + right.side) > 1)
                    return(min(left.side, right.side))
                else return(0)}
control.7 <- list(type.aggregation = c("t.tnorm", FUN.tnorm),
                    type.relation = c("tolerance", "eq.1"))
IND.7 <- BC.IND.relation.FRST(decision.table, attributes = attributes,
                              control = control.7)

## Calculate fuzzy indiscernibility relation: kernel fuzzy rough set
control.8 <- list(type.relation = c("kernel.frst", "gaussian.kernel", 0.2))
IND.8 <- BC.IND.relation.FRST(decision.table, attributes = attributes,
                              control = control.8)
#############################################################
## Example 3: Dataset containing continuous and nominal attributes
## Note. we only consider type.relation = c("tolerance", "eq.1")
## but other approaches have the same way.
#############################################################
data(RoughSetData)
decision.table <- RoughSetData$housing7.dt

## in this case, we only consider the attribute: 1, 2, 3, 4
attributes <- c(1,2,3,4)

## Calculate fuzzy indiscernibility relation
control.9 <- list(type.aggregation = c("t.tnorm", "lukasiewicz"),
                    type.relation = c("tolerance", "eq.1"))
IND.9 <- BC.IND.relation.FRST(decision.table, attributes = attributes, control = control.9)
```

BC.IND.relation.RST   *Computation of indiscernibility classes based on the rough set theory*

Description

This function implements a fundamental part of RST: the indiscernibility relation. This binary relation indicates whether it is possible to discriminate any given pair of objects from an information system.

This function can be used as a basic building block for development of other RST-based methods. A more detailed explanation of the notion of indiscernibility relation can be found in `A.Introduction-RoughSets`.

Usage

```
BC.IND.relation.RST(decision.table, feature.set = NULL)
```

Arguments

decision.table   an object inheriting from the "DecisionTable" class, which represents a decision system. See `SF.asDecisionTable`.

feature.set   an integer vector indicating indexes of attributes which should be used or an object inheriting from the FeatureSubset class. The computed indiscernibility classes will be relative to this attribute set. The default value is NULL which means that all conditional attributes should be considered. It is usually reasonable to discretize numeric attributes before the computation of indiscernibility classes.

Value

An object of a class "IndiscernibilityRelation" which is a list with the following components:

- IND.relation: a list of indiscernibility classes in the data. Each class is represented by indices of data instances which belong to that class

- type.relation: a character vector representing a type of relation used in computations. Currently, only "equivalence" is provided.

- type.model: a character vector identifying the type of model which is used. In this case, it is "RST" which means the rough set theory.

Author(s)

Andrzej Janusz

References

Z. Pawlak, "Rough Sets", International Journal of Computer and Information Sciences, vol. 11, no. 5, p. 341 - 356 (1982).

See Also

`BC.LU.approximation.RST`, `FS.reduct.computation`, `FS.feature.subset.computation`

Examples

```
#############################################
data(RoughSetData)
hiring.data <- RoughSetData$hiring.dt

## In this case, we only consider the second and third attribute:
A <- c(2,3)
## We can also compute a decision reduct:
B <- FS.reduct.computation(hiring.data)

## Compute the indiscernibility classes:
IND.A <- BC.IND.relation.RST(hiring.data, feature.set = A)
IND.A

IND.B <- BC.IND.relation.RST(hiring.data, feature.set = B)
IND.B
```

BC.LU.approximation.FRST

The fuzzy lower and upper approximations based on fuzzy rough set theory

---

Description

This is a function implementing a fundamental concept of FRST: fuzzy lower and upper approximations. Many options have been considered for determining lower and upper approximations, such as techniques based on implicator and t-norm functions proposed by (Radzikowska and Kerre, 2002).

Usage

```
BC.LU.approximation.FRST(decision.table, IND.condAttr, IND.decAttr,
  type.LU = "implicator.tnorm", control = list())
```

Arguments

decision.table   a "DecisionTable" class representing the decision table. See `SF.asDecisionTable`.

IND.condAttr   a "IndiscernibilityRelation" class of the conditional attributes which is produced by `BC.IND.relation.FRST`.

IND.decAttr   a "IndiscernibilityRelation" class of the decision attribute which is produced by `BC.IND.relation.FRST`.

type.LU   a string representing a chosen method to calculate lower and upper approximations. See the explanation in Section `Details`.

control   a list of other parameters. In order to understand how to express the `control` parameter, see the explanation in Section `Details`. The descriptions of those components and their values is as follows.

- `t.tnorm`: a type of triangular functions which have been explained in `BC.IND.relation.FRST.`
- `t.implicator`: a type of implicator functions. The following are values of this parameter:
  - `"kleene_dienes"` means $max(1 - x_1, x_2)$.
  - `"lukasiewicz"` means $min(1 - x_1 + x_2, 1)$. It is the default value.
  - `"zadeh"` means $max(1 - x_1, min(x_1, x_2))$.
  - `"gaines"` means $(x_1 <= x_2?1 : x_2/x_1)$.
  - `"godel"` means $(x_1 <= x_2?1 : x_2)$.
  - `"kleene_dienes_lukasiewicz"` means $1 - x_1 + x_1 * x_2$.
  - `"mizumoto"` means $(1 - x_1 + x_1 * x_2)$.
  - `"dubois_prade"` means $(x_2 == 0?1 - x_1 : (x_1 == 1?x_2 : 1))$.

  Where we consider the following rule: $x_1 - > x_2$.
- `q.some`: a vector of alpha and beta parameters of vaguely quantified rough set for quantifier some. The default value is `q.some = c(0.1, 0.6)`.
- `q.most`: a vector of alpha and beta parameters of vaguely quantified rough set for quantifier most. The default value is `q.most = c(0.2, 1)`.
- `alpha`: a numeric between 0 and 1 representing the threshold parameter of the fuzzy variable precision rough sets (FVPRS) (see Section `Details`). The default value is 0.05.
- `m.owa`: an integer number (m) which is used in the OWA fuzzy rough sets (see Section `Details`).
  The default value is `m.owa = round(0.5 * ncol(decision.table))`.
- `w.owa`: a vector representing the weight vector in the OWA fuzzy rough sets (see Section `Details`). The default value is NULL, which means we use the `m.owa` type.
- `type.rfrs`: a type of robust fuzzy rough sets which is one of the following methods: `"k.trimmed.min"`, `"k.mean.min"`, `"k.median.min"`, `"k.trimmed.max"`, `"k.mean.max"`, and `"k.median.max"` (see Section `Details`). The default value is `"k.trimmed.min"`.
- `k.rfrs`: a number between 0 and the number of data which is used to define considered data on robust fuzzy rough sets (RFRS) (see Section `Details`). The default value is `k.rfrs = round(0.5*nrow(decision.table))`.
- `beta.quasi`: a number between 0 and 1 representing β-precision t-norms and t-conorms in β-PFRS. The default value is 0.05.

Details

Fuzzy lower and upper approximations as explained in `B.Introduction-FuzzyRoughSets` are used to define to what extent the set of elements can be classified into a certain class strongly or weakly. We can perform various methods by choosing the parameter `type.LU`. The following is a list of all `type.LU` values:

- `"implicator.tnorm"`: It means implicator/t-norm based model proposed by (Radzikowska and Kerre, 2002). The explanation has been given in `B.Introduction-FuzzyRoughSets`. Other parameters in control related with this approach are `t.tnorm` and `t.implicator`.

In other words, when we are using "implicator.tnorm" as type.LU, we should consider parameters t.tnorm and t.implicator. The possible values of these parameters can be seen in the description of parameters.

- "vqrs": It means vaguely quantified rough sets proposed by (Cornelis et al, 2007). Basically, this concept proposed to replace fuzzy lower and upper approximations based on Radzikowska and Kerre's technique (see B.Introduction-FuzzyRoughSets) with the following equations, respectively.

$$(R_{Q_u} \downarrow A)(y) = Q_u(\frac{|R_y \cap A|}{|R_y|})$$

$$(R_{Q_l} \uparrow A)(y) = Q_l(\frac{|R_y \cap A|}{|R_y|})$$

where the quantifier Q_u and Q_l represent the terms most and some.

- "owa": It refers to ordered weighted average based fuzzy rough sets. This method was introduced by (Cornelis et al, 2010) and computes the approximations by an aggregation process proposed by (Yager, 1988). The OWA-based lower and upper approximations of A under R with weight vectors W_l and W_u are defined as

$$(R \downarrow W_l A)(y) = OWA_{W_l} \langle I(R(x,y), A(y)) \rangle$$

$$(R \uparrow W_u A)(y) = OWA_{W_u} \langle T(R(x,y), A(y)) \rangle$$

We provide two ways to define the weight vectors as follows:

  - m.owa: Let $m.owa$ be m and $m \leq n$, this model is defined by
    $W_l = <w_i^l> = w_{n+1-i}^l = \frac{2^{m-i}}{2^m-1}$ for $i = 1, \ldots, m$ and 0 for $i = m+1, \ldots, n$
    $W_u = <w_i^u> = w_i^u = \frac{2^{m-i}}{2^m-1}$ for $i = 1, \ldots, m$ and 0 for $i = m+1, \ldots, n$
    where n is the number of data.
  - custom: In this case, users define the own weight vector. It should be noted that the weight vectors $<w_i>$ should satisfy $w_i \in [0, 1]$ and their summation is 1.

- "fvprs": It refers to fuzzy variable precision rough sets (FVPRS) introduced by (Zhao et al, 2009). It is a combination between variable precision rough sets (VPRS) and FRST. This function implements the construction of lower and upper approximations as follows.

$$(R_\alpha \downarrow A)(y) = inf_{A(y) \leq \alpha} \mathcal{I}(R(x,y), \alpha) \wedge inf_{A(y) > \alpha} \mathcal{I}(R(x,y), A(y))$$

$$(R_\alpha \uparrow A)(y) = sup_{A(y) \geq N(\alpha)} \mathcal{T}(R(x,y), N(\alpha)) \vee sup_{A(y) < N(\alpha)} \mathcal{T}(R(x,y), A(y))$$

where α, $\mathcal{I}$ and $\mathcal{T}$ are the variable precision parameter, implicator and t-norm operators, respectively.

- "rfrs": It refers to robust fuzzy rough sets (RFRS) proposed by (Hu et al, 2012). This package provides six types of RFRS which are k-trimmed minimum, k-mean minimum, k-median minimum, k-trimmed maximum, k-mean maximum, and k-median maximum. Basically, these methods are a special case of ordered weighted average (OWA) where they consider the weight vectors as follows.

  - "k.trimmed.min": $w_i^l = 1$ for $i = n - k$ and $w_i^l = 0$ otherwise.
  - "k.mean.min": $w_i^l = 1/k$ for $i > n - k$ and $w_i^l = 0$ otherwise.
  - "k.median.min": $w_i^l = 1$ if k odd, $i = n - (k-1)/2$ and $w_i^l = 1/2$ if k even, $i = n - k/2$ and $w_i^l = 0$ otherwise.
  - "k.trimmed.max": $w_i^u = 1$ for $i = k + 1$ and $w_i^u = 0$ otherwise.
  - "k.mean.max": $w_i^u = 1/k$ for $i < k + 1$ and $w_i^u = 0$ otherwise.
  - "k.median.max": $w_i^u = 1$ if k odd, $i = (k+1)/2$ and $w_i^u = 1/2$ if k even, $i = k/2 + 1$ or $w_i^u = 0$ otherwise.

- "beta.pfrs": It refers to β-precision fuzzy rough sets (β-PFRS) proposed by (Salido and Murakami, 2003). This algorithm uses β-precision quasi-$\mathcal{T}$-norm and β-precision quasi-$\mathcal{T}$-conorm. The following are the β-precision versions of fuzzy lower and upper approximations of a fuzzy set A in U

  $(R_B \downarrow A)(y) = T_{\beta_{x \in U}} \mathcal{I}(R_B(x,y), A(x))$

  $(R_B \uparrow A)(y) = S_{\beta_{x \in U}} \mathcal{T}(R_B(x,y), A(x))$

  where T_β and S_β are β-precision quasi-$\mathcal{T}$-norm and β-precision quasi-$\mathcal{T}$-conorm. Given a t-norm $\mathcal{T}$, a t-conorm S, $\beta \in [0,1]$ and $n \in N \setminus \{0,1\}$, the corresponding β-precision quasi-t-norm T_β and β-precision-$\mathcal{T}$-conorm S_β of order n are $[0,1]^n \rightarrow [0,1]$ mappings such that for all $x = (x_1, ..., x_n)$ in $[0,1]^n$,

  $T_\beta(x) = \mathcal{T}(y_1, ..., y_{n-m})$,

  $S_\beta(x) = \mathcal{T}(z_1, ..., z_{n-p})$,

  where y_i is the ith greatest element of x and z_i is the ith smallest element of x, and

  $m = max(i \in \{0, ..., n\} | i \leq (1 - \beta) \sum_{j=1}^{n} x_j)$,

  $p = max(i \in \{0, ..., n\} | i \leq (1 - \beta) \sum_{j=1}^{n} (a - x_j))$.

  In this package we use min and max for $\mathcal{T}$-norm and $\mathcal{T}$-conorm, respectively.

- "custom": It refers to user-defined lower and upper approximations. An example can be seen in Section Examples.

The parameter type.LU, which is explained above, is related with parameter control. In other words, when choosing a specific value of type.LU, we should take into account to set values of related components in control. The components that are considered depend on what kind of lower and upper approximations are used. So, we do not need to assign all components for a particular approach but only components related with type.LU. The following is a list showing the components of each approaches.

- type.LU = "implicator.tnorm":

  control <- list(t.implicator, t.tnorm)

- type.LU = "vqrs":

  control <- list(q.some, q.most, type.aggregation, t.tnorm)

- type.LU = "owa":

  control <- list(t.implicator, t.tnorm, m.owa)

  or

  control <- list(t.implicator, t.tnorm, w.owa)

- type.LU = "fvprs":

  control <- list(t.implicator, t.tnorm, alpha)

- type.LU = "beta.pfrs":

  control <- list(t.implicator, t.tnorm, beta.quasi)

- type.LU = "rfrs":

  control <- list(t.implicator, t.tnorm, type.rfrs, k.rfrs)

- type.LU = "custom":

  control <- list(t.implicator, t.tnorm, FUN.lower, FUN.upper)

The description of the components can be seen in the `control` parameter. In Section Examples, we provide two examples showing different cases which are when we have to handle a nominal decision attribute and a continuous one.

It should be noted that this function depends on `BC.IND.relation.FRST` which is a function used to calculate the fuzzy indiscernibility relation as input data. So, it is obvious that before performing this function, users must execute `BC.IND.relation.FRST` first.

Value

A class `"LowerUpperApproximation"` representing fuzzy rough set (fuzzy lower and upper approximations). It contains the following components:

- `fuzzy.lower`: a list showing the lower approximation classified based on decision concepts for each index of objects. The value refers to the degree of objects included in the lower approximation. In case the decision attribute is continuous, the result is in a data frame with dimension (number of objects x number of objects) and the value on position (i, j) shows the membership of object i to the lower approximation of the similarity class of object j.

- `fuzzy.upper`: a list showing the upper approximation classified based on decision concepts for each index of objects. The value refers to the degree of objects included in the upper approximation. In case the decision attribute is continuous values, the result is in data frame with dimension (number of objects x number of objects) and the value on position (i, j) shows the membership of object i to the upper approximation of the similarity class of object j.

- `type.LU`: a string representing the type of lower and upper approximation approaches.

- `type.model`: a string showing the type of model which is used. In this case, it is `"FRST"` which means fuzzy rough set theory.

Author(s)

Lala Septem Riza

References

A. M. Radzikowska and E. E. Kerre, "A Comparative Study of Fuzzy Rough Sets", Fuzzy Sets and Systems, vol. 126, p. 137 - 156 (2002).

C. Cornelis, M. De Cock, and A. Radzikowska, "Vaguely Quantified Rough Sets", Proceedings of 11th International Conference on Rough Sets, Fuzzy Sets, Data Mining and Granular Computing (RSFDGrC2007), Lecture Notes in Artificial Intelligence 4482, p. 87 - 94 (2007).

C. Cornelis, N. Verbiest, and R. Jensen, "Ordered Weighted Average Based Fuzzy Rough Sets", Proceedings of the 5th International Conference on Rough Sets and Knowledge Technology (RSKT 2010), p. 78 - 85 (2010).

J. M. F. Salido and S. Murakami, "Rough Set Analysis of a General Type of Fuzzy Data Using Transitive Aggregations of Fuzzy Similarity Relations", Fuzzy Sets Syst., vol. 139, p. 635 - 660 (2003).

Q. Hu, L. Zhang, S. An, D. Zhang, and D. Yu, "On Robust Fuzzy Rough Set Models", IEEE Trans. on Fuzzy Systems, vol. 20, no. 4, p. 636 - 651 (2012).

R. Jensen and Q. Shen, "New Approaches to Fuzzy-Rough Feature Selection", IEEE Trans. on Fuzzy Systems, vol. 19, no. 4, p. 824 - 838 (2009).

R. R. Yager, "On Ordered Weighted Averaging Aggregation Operators in Multicriteria Decision Making", IEEE Transactions on Systems, Man, and Cybernetics, vol. 18, p. 183 - 190 (1988).

S. Y. Zhao, E. C. C. Tsang, and D. G. Chen, "The Model of Fuzzy Variable Precision Rough Sets", IEEE Trans. Fuzzy Systems, vol. 17, no. 2, p. 451 - 467 (2009).

See Also

`BC.IND.relation.RST`, `BC.LU.approximation.RST`, and `BC.positive.reg.FRST`

Examples

```
############################################################
## 1. Example: Decision table contains nominal decision attribute
## we are using the same dataset and indiscernibility
## relation along this example.
############################################################
dt.ex1 <- data.frame(c(-0.4, -0.4, -0.3, 0.3, 0.2, 0.2),
                     c(-0.3, 0.2, -0.4, -0.3, -0.3, 0),
        c(-0.5, -0.1, -0.3, 0, 0, 0),
        c("no", "yes", "no", "yes", "yes", "no"))
colnames(dt.ex1) <- c("a", "b", "c", "d")
decision.table <- SF.asDecisionTable(dataset = dt.ex1, decision.attr = 4)

## let us consider the first and second attributes
## only as conditional attributes
condAttr <- c(1, 2)

## let us consider the fourth attribute as decision attribute
decAttr <- c(4)

#### calculate fuzzy indiscernibility relation ####
control.ind <- list(type.aggregation = c("t.tnorm", "lukasiewicz"),
                    type.relation = c("tolerance", "eq.1"))
control.dec <- list(type.aggregation = c("crisp"), type.relation = "crisp")

## fuzzy indiscernibility relation of conditional attribute
IND.condAttr <- BC.IND.relation.FRST(decision.table, attributes = condAttr,
                                     control = control.ind)

## fuzzy indiscernibility relation of decision attribute
IND.decAttr <- BC.IND.relation.FRST(decision.table, attributes = decAttr,
                                    control = control.dec)

#### Calculate fuzzy lower and upper approximation using type.LU : "implicator.tnorm" ####

control <- list(t.implicator = "lukasiewicz", t.tnorm = "lukasiewicz")
FRST.LU <- BC.LU.approximation.FRST(decision.table, IND.condAttr, IND.decAttr,
              type.LU = "implicator.tnorm", control = control)

#### Calculate fuzzy lower and upper approximation using type.LU : "vqrs" ####
control <- list(q.some = c(0.1, 0.6), q.most = c(0.2, 1), t.tnorm = "lukasiewicz")
FRST.VQRS <- BC.LU.approximation.FRST(decision.table, IND.condAttr, IND.decAttr,
```

```
                  type.LU = "vqrs", control = control)

#### Calculate fuzzy lower and upper approximation using type.LU : "owa" ####
control <- list(t.implicator = "lukasiewicz", t.tnorm = "lukasiewicz", m.owa = 3)
FRST.OWA.1 <- BC.LU.approximation.FRST(decision.table, IND.condAttr, IND.decAttr,
                 type.LU = "owa", control = control)

#### Calculate fuzzy lower and upper approximation using type.LU :
#### "owa" with customized function
#### In this case, we are using the same weight vector as
#### previous one with m.owa = 3
control <- list(t.implicator = "lukasiewicz", t.tnorm = "lukasiewicz",
                  w.owa =  c(0, 0, 0, 0.14, 0.29, 0.57))
FRST.OWA.2 <- BC.LU.approximation.FRST(decision.table, IND.condAttr, IND.decAttr,
                 type.LU = "owa", control = control)

#### Calculate fuzzy lower and upper approximation using type.LU : "fvprs" ####
control <- list(t.implicator = "lukasiewicz", t.tnorm = "lukasiewicz", alpha = 0.05)
FRST.fvprs <- BC.LU.approximation.FRST(decision.table, IND.condAttr, IND.decAttr,
                 type.LU = "fvprs", control = control)


#### Calculate fuzzy lower and upper approximation using type.LU : "rfrs" ####
control <- list(t.implicator = "lukasiewicz", t.tnorm = "lukasiewicz",
                   type.rfrs = "k.trimmed.min", k.rfrs = 0)
FRST.rfrs <- BC.LU.approximation.FRST(decision.table, IND.condAttr, IND.decAttr,
                 type.LU = "rfrs", control = control)

#### Calculate fuzzy lower and upper approximation using type.LU : "beta.pfrs" ####
control <- list(t.implicator = "lukasiewicz", t.tnorm = "lukasiewicz", beta.quasi = 0.05)
FRST.beta.pfrs <- BC.LU.approximation.FRST(decision.table, IND.condAttr, IND.decAttr,
                 type.LU = "beta.pfrs", control = control)

#### Calculate fuzzy lower and upper approximation using type.LU : "custom" ####
## In this case, we calculate approximations randomly.
f.lower <- function(x){
        return(min(runif(1, min = 0, max = 1) * x))
}
f.upper <- function(x){
        return(max(runif(1, min = 0, max = 1) * x))
}
control <- list(t.implicator = "lukasiewicz", t.tnorm = "lukasiewicz", FUN.lower = f.lower,
                  FUN.upper = f.upper)
FRST.custom <- BC.LU.approximation.FRST(decision.table, IND.condAttr, IND.decAttr,
                 type.LU = "custom", control = control)

#### In this case, we use custom function for triangular norm and implicator operator
## For example, let us define our implicator and t-norm operator as follows.
imp.lower <- function(antecedent, consequent){
                    return(max(1 - antecedent, consequent))
                 }
tnorm.upper <- function(x, y){
                  return (x * y)
```

```
            }
control.custom <- list(t.implicator = imp.lower, t.tnorm = tnorm.upper)
FRST.LU.custom <- BC.LU.approximation.FRST(decision.table, IND.condAttr, IND.decAttr,
               type.LU = "implicator.tnorm", control = control.custom)

#############################################################
## 2. Example: Decision table contains a continuous decision attribute.
## It should be noted that in this example, we are using
## the same dataset and indiscernibility relation.
## We only show one method but for other approaches
## the procedure is analogous to the previous example
#############################################################
## In this case, we are using housing dataset containing 7 objects
data(RoughSetData)
decision.table <- RoughSetData$housing7.dt

## let us consider the first and second conditional attributes only,
## and the decision attribute at 14.
cond.attributes <- c(1, 2)
dec.attributes <- c(14)
control.ind <- list(type.aggregation = c("t.tnorm", "lukasiewicz"),
               type.relation = c("tolerance", "eq.1"))
IND.condAttr <- BC.IND.relation.FRST(decision.table, attributes = cond.attributes,
                                     control = control.ind)
IND.decAttr <- BC.IND.relation.FRST(decision.table, attributes = dec.attributes,
                                     control = control.ind)

#### Calculate fuzzy lower and upper approximation using type.LU : "implicator.tnorm" ####

control <- list(t.implicator = "lukasiewicz", t.tnorm = "lukasiewicz")
FRST.LU <- BC.LU.approximation.FRST(decision.table, IND.condAttr, IND.decAttr,
               type.LU = "implicator.tnorm", control = control)
```

BC.LU.approximation.RST

Computation of lower and upper approximations of decision classes

---

Description

This function implements a fundamental part of RST: computation of lower and upper approxi-
mations. The lower and upper approximations determine whether the objects can be certainty or
possibly classified to a particular decision class on the basis of available knowledge.

Usage

```
BC.LU.approximation.RST(decision.table, IND)
```

Arguments

decision.table an object inheriting from the ″DecisionTable″ class, which represents a deci-
sion system. See `SF.asDecisionTable`.

IND an object inheriting from the ″IndiscernibilityRelation″ class, which rep-
resents indiscernibility clasees in the data.

Details

This function can be used as a basic building block for development of other RST-based methods.
A more detailed explanation of this notion can be found in `A.Introduction-RoughSets`.

Value

An object of a class ″LowerUpperApproximation″ which is a list with the following components:

- lower.approximation: a list with indices of data instances included in lower approximations
  of decision classes.

- upper.approximation: a list with indices of data instances included in upper approximations
  of decision classes.

- type.model: a character vector identifying the type of model which was used. In this case, it
  is ″RST″ which means the rough set theory.

Author(s)

Andrzej Janusz

References

Z. Pawlak, "Rough Sets", International Journal of Computer and Information Sciences, vol. 11, no.
5, p. 341 - 356 (1982).

See Also

`BC.IND.relation.RST`, `BC.LU.approximation.FRST`

Examples

```
#########################################
data(RoughSetData)
hiring.data <- RoughSetData$hiring.dt

## We select a single attribute for computation of indiscernibility classes:
A <- c(2)

## Compute the indiscernibility classes:
IND.A <- BC.IND.relation.RST(hiring.data, feature.set = A)

## Compute the lower and upper approximations:
roughset <- BC.LU.approximation.RST(hiring.data, IND.A)
roughset
```

BC.negative.reg.RST   *Computation of a negative region*

Description

This function implements a fundamental part of RST: computation of a negative region and the degree of dependency. This function can be used as a basic building block for development of other RST-based methods. A more detailed explanation of this notion can be found in A.Introduction-RoughSets.

Usage

```
BC.negative.reg.RST(decision.table, roughset)
```

Arguments

decision.table   an object inheriting from the ″DecisionTable″ class, which represents a decision system. See SF.asDecisionTable.

roughset   an object inheriting from the ″LowerUpperApproximation″ class, which represents lower and upper approximations of decision classes in the data. Such objects are typically produced by calling the BC.LU.approximation.RST function.

Value

An object of a class ″NegativeRegion″ which is a list with the following components:

- negative.reg: an integer vector containing indices of data instances belonging to the boundary region,
- degree.dependency: a numeric value giving the degree of dependency,
- type.model: a varacter vector identifying the utilized model. In this case, it is ″RST″ which means the rough set theory.

Author(s)

Dariusz Jankowski, Andrzej Janusz

References

Z. Pawlak, "Rough Sets", International Journal of Computer and Information Sciences, vol. 11, no. 5, p. 341 - 356 (1982).

See Also

BC.IND.relation.RST, BC.LU.approximation.RST, BC.LU.approximation.FRST

Examples

```
##########################################################
data(RoughSetData)
hiring.data <- RoughSetData$hiring.dt

## We select a single attribute for computation of indiscernibility classes:
A <- c(2)

## compute the indiscernibility classes:
IND.A <- BC.IND.relation.RST(hiring.data, feature.set = A)

## compute the lower and upper approximation:
roughset <- BC.LU.approximation.RST(hiring.data, IND.A)

## get the boundary region:
pos.negative = BC.negative.reg.RST(hiring.data, roughset)
pos.negative
```

BC.positive.reg.FRST   *Positive region based on fuzzy rough set*

Description

This is a function that implements a fundamental concept of fuzzy rough set theory which is the positive region and the corresponding degree of dependency. The explanation about this concept can be seen in `B.Introduction-FuzzyRoughSets`.

Usage

```
BC.positive.reg.FRST(decision.table, fuzzyroughset)
```

Arguments

decision.table   a "DecisionTable" class representing the decision table. See `SF.asDecisionTable`.

fuzzyroughset   a "LowerUpperApproximation" class representing a fuzzy rough set that is produced by `BC.LU.approximation.FRST`.

Details

In order to compute the function, we need to calculate the indiscernibility relation by executing `BC.IND.relation.FRST` and the lower and upper approximations by calling `BC.LU.approximation.FRST`.

Value

A class ″PositiveRegion″ containing the following components:

- positive.freg: a vector representing membership degrees to the fuzzy positive region for each index of objects.

- degree.dependency: a value expressing the degree of dependency.

- type.model: a string representing type of models. In this case, it is ″FRST″ which means fuzzy rough set theory.

Author(s)

Lala Septem Riza

References

R. Jensen and Q. Shen, "New Approaches to Fuzzy-Rough Feature Selection", IEEE Trans. on Fuzzy Systems, vol. 19, no. 4, p. 824 - 838 (2009).

See Also

BC.LU.approximation.FRST, BC.IND.relation.FRST, BC.IND.relation.RST,

BC.LU.approximation.RST, and BC.positive.reg.FRST.

Examples

```
#############################################################
##### 1. Example: Using a simple decision table containing
#####             nominal values for the decision attribute
#############################################################
dt.ex1 <- data.frame(c(-0.4, -0.4, -0.3, 0.3, 0.2, 0.2),
                     c(-0.3, 0.2, -0.4, -0.3, -0.3, 0),
       c(-0.5, -0.1, -0.3, 0, 0, 0),
       c("no", "yes", "no", "yes", "yes", "no"))
colnames(dt.ex1) <- c("a", "b", "c", "d")
decision.table <- SF.asDecisionTable(dataset = dt.ex1, decision.attr = 4)

## let us consider the first and second attributes only as conditional attribute
condAttr <- c(1, 2)

## let us consider the fourth attribute as decision attribute
decAttr <- c(4)

#### Calculate fuzzy indiscernibility relation ####
control.ind <- list(type.aggregation = c("t.tnorm", "lukasiewicz"),
                    type.relation = c("tolerance", "eq.1"))
control.dec <- list(type.aggregation = c("crisp"), type.relation = "crisp")

IND.condAttr <- BC.IND.relation.FRST(decision.table, attributes = condAttr,
                                     control = control.ind)
IND.decAttr <- BC.IND.relation.FRST(decision.table, attributes = decAttr,
```

```
                                                control = control.dec)

#### Calculate fuzzy lower and upper approximation using type.LU :
#### "implicator.tnorm"
control <- list(t.implicator = "lukasiewicz")
FRST.LU <- BC.LU.approximation.FRST(decision.table, IND.condAttr, IND.decAttr,
                type.LU = "implicator.tnorm", control = control)

#### Determine positive regions ####
res.1 <- BC.positive.reg.FRST(decision.table, FRST.LU)

###############################################################
##### 2. Example: Using the housing decision table containing
#####             continuous values for the decision attribute
###############################################################

## In this case, we are using the housing dataset containing 7 objects
data(RoughSetData)
decision.table <- RoughSetData$housing7.dt

conditional.attr <- c(1, 2)
decision.attr = c(14)
control.ind <- list(type.aggregation = c("t.tnorm", "lukasiewicz"),
                     type.relation = c("tolerance", "eq.1"))

#### Calculate fuzzy indiscernibility relation ####
IND.condAttr <- BC.IND.relation.FRST(decision.table, attributes = conditional.attr,
                                     control = control.ind)
IND.decAttr <- BC.IND.relation.FRST(decision.table, attributes = decision.attr,
                                    control = control.ind)

#### Calculate fuzzy lower and upper approximation using type.LU :
#### "implicator.tnorm"
control <- list(t.implicator = "lukasiewicz", t.tnorm = "lukasiewicz")

FRST.LU <- BC.LU.approximation.FRST(decision.table, IND.condAttr, IND.decAttr,
                type.LU = "implicator.tnorm", control = control)

#### Determine fuzzy regions ####
res.2 <- BC.positive.reg.FRST(decision.table, FRST.LU)
```

BC.positive.reg.RST   *Computation of a positive region*

Description

This function implements a fundamental part of RST: computation of a positive region and the degree of dependency. This function can be used as a basic building block for development of other RST-based methods. A more detailed explanation of this notion can be found in `A.Introduction-RoughSets`.

Usage

```
BC.positive.reg.RST(decision.table, roughset)
```

Arguments

decision.table   an object inheriting from the ″DecisionTable″ class, which represents a decision system. See `SF.asDecisionTable`.

roughset   an object inheriting from the ″LowerUpperApproximation″ class, which represents lower and upper approximations of decision classes in the data. Such objects are typically produced by calling the `BC.LU.approximation.RST` function.

Value

An object of a class ″PositiveRegion″ which is a list with the following components:

- `positive.reg`: an integer vector containing indices of data instances belonging to the positive region,
- `degree.dependency`: a numeric value giving the degree of dependency,
- `type.model`: a varacter vector identifying the utilized model. In this case, it is ″RST″ which means the rough set theory.

Author(s)

Andrzej Janusz

References

Z. Pawlak, "Rough Sets", International Journal of Computer and Information Sciences, vol. 11, no. 5, p. 341 - 356 (1982).

See Also

`BC.IND.relation.RST`, `BC.LU.approximation.RST`, `BC.LU.approximation.FRST`

Examples

```
#############################################################
data(RoughSetData)
hiring.data <- RoughSetData$hiring.dt

## We select a single attribute for computation of indiscernibility classes:
A <- c(2)

## compute the indiscernibility classes:
IND.A <- BC.IND.relation.RST(hiring.data, feature.set = A)

## compute the lower and upper approximation:
roughset <- BC.LU.approximation.RST(hiring.data, IND.A)
```

```
## get the positive region:
pos.region = BC.positive.reg.RST(hiring.data, roughset)
pos.region
```

C.FRNN.FRST   *The fuzzy-rough nearest neighbor algorithm*

Description

It is used to predict new datasets/patterns based on the fuzzy-rough nearest neighbor algorithm (FRNN) proposed by (Jensen and Cornelis, 2011).

Usage

```
C.FRNN.FRST(decision.table, newdata, control = list())
```

Arguments

decision.table	a "DecisionTable" class representing the decision table. See SF.asDecisionTable. It should be noted that the data must be numeric values instead of string/char.
newdata	a "DecisionTable" class representing data for the test process. See SF.asDecisionTable.
control	a list of other parameters as follows.

- type.LU: a type of lower and upper approximations. See Section Details. The default value is type.LU = "implicator.tnorm".
- k: the number of neighbors. It should be taken into account that this value could affect the accuracy. The default value is 5.
- type.aggregation: the type of the aggregation operator. See BC.IND.relation.FRST. The default value is type.aggregation = c("t.tnorm", "lukasiewicz").
- type.relation: the type of relation. See BC.LU.approximation.FRST. The default value is c("tolerance", "eq.1").
- type.implicator: the type of implicator operator. See BC.LU.approximation.FRST. The default value is "lukasiewicz".
- q.some: a vector of values of alpha and beta parameters of VQRS. See BC.LU.approximation.FRST. The default value is c(0.1, 0.6).
- q.most: a vector of values of alpha and beta parameter of VQRS. See BC.LU.approximation.FRST. The default value is c(0.2, 1).

Details

This method uses the fuzzy lower and upper approximations to improve the fuzzy nearest neighbor (FNN) algorithm. This algorithm assigns a class to a target instance t as follows.

- Determine k nearest neighbors considering their similarity to new patterns.

- Assign new patterns to the class based on maximal value of fuzzy lower and upper approximations. If a value of fuzzy lower approximation is high, it shows that neighbors of newdata belong to a particular class, e.g. C. On the other hand, a high value of fuzzy upper approximation means that at least one neighbor belongs to that class.

In this function, we provide two approaches based on types of fuzzy lower and upper approximations. The following is a list of the considered approximations:

- ″implicator.tnorm″: It refers to lower and upper approximations based on implicator/t-norm approach. For more detail, it can be seen in `BC.LU.approximation.FRST`. When using this approach, we need to assign the control parameter as follows:

  control <- list(type.LU = ″implicator.tnorm″, k,

  type.aggregation, type.relation, t.implicator)

  The detailed description of the components in the control parameter can be seen in
  `BC.LU.approximation.FRST`.

- ″vqrs″: It refers to lower and upper approximations based on vaguely quantified rough sets. For more detail, it can be seen in `BC.LU.approximation.FRST`. When using this approach, we need to assign the control parameter as follows:

  control <- list(type.LU = ″vqrs″, k, q.some, q.most,

  type.relation, type.aggregation)

  The detailed description of the components in the control parameter can be seen in
  `BC.LU.approximation.FRST`.

Value

A matrix of predicted classes of newdata.

Author(s)

Lala Septem Riza

References

R. Jensen and C. Cornelis, "Fuzzy-rough Nearest Neighbour Classification and Prediction", Theoretical Computer Science, vol. 412, p. 5871 - 5884 (2011).

See Also

`C.FRNN.O.FRST`, `C.POSNN.FRST`

Examples

```
###############################################################
## In this example, we are using Iris dataset.
## It should be noted that since the values of the decision attribute are strings,
## they should be transformed into numeric values using unclass()
###############################################################
data(iris)
## shuffle the data
```

```
set.seed(2)
irisShuffled <- iris[sample(nrow(iris)),]

## transform values of the decision attribute into numerics
irisShuffled[,5] <- unclass(irisShuffled[,5])

## split the data into training and testing data
iris.training <- irisShuffled[1:105,]
iris.testing <- irisShuffled[106:nrow(irisShuffled),1:4]

colnames(iris.training) <- c("Sepal.Length", "Sepal.Width", "Petal.Length",
                        "Petal.Width", "Species")

## convert into a standard decision table
decision.table <- SF.asDecisionTable(dataset = iris.training, decision.attr = 5,
                                    indx.nominal = c(5))
tst.iris <- SF.asDecisionTable(dataset = iris.testing)

###### FRNN algorithm using lower/upper approximation:
###### Implicator/tnorm based approach
control <- list(type.LU = "implicator.tnorm", k = 20,
                type.aggregation = c("t.tnorm", "lukasiewicz"),
                type.relation = c("tolerance", "eq.1"), t.implicator = "lukasiewicz")
## Not run: res.1 <- C.FRNN.FRST(decision.table = decision.table, newdata = tst.iris,
                            control = control)
## End(Not run)

###### FRNN algorithm using VQRS
control <- list(type.LU = "vqrs", k = 20, q.some = c(0.1, 0.6), q.most = c(0.2, 1),
                type.relation = c("tolerance", "eq.1"),
                type.aggregation = c("t.tnorm","lukasiewicz"))
## Not run: res.2 <- C.FRNN.FRST(decision.table = decision.table, newdata = tst.iris,
                            control = control)
## End(Not run)
```

C.FRNN.O.FRST   *The fuzzy-rough ownership nearest neighbor algorithm*

Description

It is used to predict classes of new datasets/patterns based on the fuzzy-rough ownership nearest neighbor algorithm (FRNN.O) proposed by (Sarkar, 2007).

Usage

```
C.FRNN.O.FRST(decision.table, newdata, control = list())
```

Arguments

decision.table   a "DecisionTable" class representing the decision table. See `SF.asDecisionTable`.
                  It should be noted that the data must be numeric values instead of strings/characters.

newdata   a "DecisionTable" class representing data for the test process.
                  See `SF.asDecisionTable`.

control   a list of other parameters.

- m: the weight of distance. The default value is 2.

Details

This method improves fuzzy k-nearest neighbors (FNN) by introducing rough sets into it. To avoid determining k by trial and error procedure, this method uses all training data. Uncertainties in data are accommodated by introducing the rough ownership function. It is the following equation o_c of each class expressing a squared weighted distance between a test pattern and all training data d and constrained fuzzy membership μ_{C_c}.

$$o_c(y) = \frac{1}{|X|}\mu_{C_c}(x)\exp\left(-d^{1/(q-1)}\right)$$

where $d = \sum_{j=1}^{N} K_j(y_j - x_{ij})^2$

The predicted value of y is obtained by selecting class c where $o_c(y)$ is maximum.

Value

A matrix of predicted classes of newdata.

Author(s)

Lala Septem Riza

References

M. Sarkar, "Fuzzy-Rough Nearest-Neighbor Algorithm in Classification" Fuzzy Sets and Systems, vol. 158, no. 19, p. 2123 - 2152 (2007).

See Also

`C.FRNN.FRST`, `C.POSNN.FRST`

Examples

```
#############################################################
## In this example, we are using Iris dataset.
## It should be noted that since the values of the decision attribute are strings,
## they should be transformed into numeric values using unclass()
#############################################################
data(iris)
## shuffle the data
set.seed(2)
irisShuffled <- iris[sample(nrow(iris)),]
```

```
## transform values of the decision attribute into numerics
irisShuffled[,5] <- unclass(irisShuffled[,5])

## split the data into training and testing data
iris.training <- irisShuffled[1:105,]
iris.testing <- irisShuffled[106:nrow(irisShuffled),1:4]

## convert into the standard decision table
colnames(iris.training) <- c("Sepal.Length", "Sepal.Width", "Petal.Length",
                             "Petal.Width", "Species")
decision.table <- SF.asDecisionTable(dataset = iris.training, decision.attr = 5,
                                     indx.nominal = c(5))
tst.iris <- SF.asDecisionTable(dataset = iris.testing)

## in this case, we are using "gradual" for type of membership
control <- list(m = 2)

## Not run: res.test.FRNN.O <- C.FRNN.O.FRST(decision.table = decision.table, newdata = tst.iris,
                                control = control)
## End(Not run)
```

C.POSNN.FRST   *The positive region based fuzzy-rough nearest neighbor algorithm*

Description

It is a function used to implement the positive region based fuzzy-rough nearest neighbor algorithm (POSNN) which was proposed by (Verbiest et al, 2012) for predicting classes of new data.

Usage

```
C.POSNN.FRST(decision.table, newdata, control = list())
```

Arguments

decision.table	a "DecisionTable" class representing the decision table. See SF.asDecisionTable. It should be noted that the data must be numeric values instead of string/char.
newdata	a "DecisionTable" class representing data for the test process. See SF.asDecisionTable.
control	a list of other parameters which is the same as C.FRNN.FRST.

Details

This method is aimed to improve the fuzzy-rough nearest neighbor algorithm (C.FRNN.FRST) algorithm by considering the fuzzy positive region. Basically the following steps are used to classify an instance t:

- determine the set of k-nearest neighbor of t, NN.

- assign t to the class C for which

$$\frac{\displaystyle\sum_{x \in NN} R(x,t)C(x)POS(x)}{\displaystyle\sum_{x \in NN} R(x,t)}$$

  is maximal.

Value

A matrix of predicted classes of newdata.

Author(s)

Lala Septem Riza

References

N. Verbiest, C. Cornelis and R. Jensen, "Fuzzy-rough Positive Region Based Nearest Neighbour Classification", In Proceedings of the 20th International Conference on Fuzzy Systems (FUZZ-IEEE 2012), p. 1961 - 1967 (2012).

See Also

C.FRNN.FRST, C.FRNN.O.FRST

Examples

```
###############################################################
## In this example, we are using Iris dataset.
## It should be noted that since the values of the decision attribute are strings,
## they should be transformed into numeric values using unclass()
###############################################################
data(iris)
## shuffle the data
set.seed(2)
irisShuffled <- iris[sample(nrow(iris)),]

## transform values of the decision attribute into numerics
irisShuffled[,5] <- unclass(irisShuffled[,5])

## split the data into training and testing data
iris.training <- irisShuffled[1:105,]
iris.testing <- irisShuffled[106:nrow(irisShuffled),1:4]

colnames(iris.training) <- c("Sepal.Length", "Sepal.Width", "Petal.Length",
                        "Petal.Width", "Species")

## convert into the standard decision table
decision.table <- SF.asDecisionTable(dataset = iris.training, decision.attr = 5,
                                    indx.nominal = c(5))
tst.iris <- SF.asDecisionTable(dataset = iris.testing)
```

```
## FRNN algorithm using lower/upper approximation: Implicator/tnorm based approach
control <- list(type.LU = "implicator.tnorm", k = 20, t.tnorm = "lukasiewicz",
                type.relation = c("tolerance", "eq.1"), t.implicator = "lukasiewicz")

## Not run: res.test.POSNN <- C.POSNN.FRST(decision.table = decision.table,
                              newdata = tst.iris, control = control)
## End(Not run)
```

D.discretization.RST   *The wrapper function for discretization methods*

Description

It is a wrapper function for all discretization methods based on RST. It provides an interface that allows users to use the discretization methods easily.

Usage

```
D.discretization.RST(decision.table,
  type.method = "unsupervised.quantiles", ...)
```

Arguments

decision.table an object inheriting from the "DecisionTable" class, which represents a decision system. See `SF.asDecisionTable`.

type.method a character representing a discretization method to be used in the computations. Currently it can be one of the following methods:

- "global.discernibility": See `D.global.discernibility.heuristic.RST`.
- "local.discernibility": See `D.local.discernibility.heuristic.RST`.
- "unsupervised.intervals": See `D.discretize.equal.intervals.RST`.
- "unsupervised.quantiles": See `D.discretize.quantiles.RST`.

... parameters that are passed to the discretization methods. See the manual of particular functions.

Details

The discretization is used to convert numeric attributes into nominal ones in an information system. It is usually a preliminary step for the most of methods based on the rough set theory, which need nominal attributes, for exemple, to compute the indiscernibility relation.

Output of this function is an object of a class Discretization which contains cut values. The function `SF.applyDecTable` can be used to generate a new (discretized) decision table from the computed cuts. Type of all attributes in the resulting table will be changed into nominal (i.e. ordered factors).

All implemented supervised discretization methods need a nominal decision attribute. Furthermore, especially for the method type "global.discernibility", all conditional attributes must be numeric. A different method needs to be chosen in a case when a data set contains attributes of mixed types (numeric and nominal).

Value

An object of a class "Discretization" which stores cuts for each conditional attribute. It contains the following components:

- cut.values: a list representing cut values for each of numeric attributes. NULL value means that no cut was selected for a given attribute.

- type.method: the type of method which is used to define cut values.

- type.task: the type of task which is "discretization".

- model: the type of model which is "RST".

Author(s)

Andrzej Janusz

See Also

[BC.LU.approximation.RST](), [FS.reduct.computation](), [SF.applyDecTable]().

Examples

```
###################################################################
## Example: Determine cut values and generate new decision table
###################################################################
data(RoughSetData)
wine.data <- RoughSetData$wine.dt
cut.values1 <- D.discretization.RST(wine.data,
                                    type.method = "unsupervised.quantiles",
                                    nOfIntervals = 3)

## generate a new decision table
wine.discretized1 <- SF.applyDecTable(wine.data, cut.values1)
dim(wine.discretized1)
lapply(wine.discretized1, unique)

cut.values2 <- D.discretization.RST(wine.data,
                                    type.method = "global.discernibility")

wine.discretized2 <- SF.applyDecTable(wine.data, cut.values2)
dim(wine.discretized2)
lapply(wine.discretized2, unique)
```

D.discretize.equal.intervals.RST

Unsupervised discretization into intervals of equal length.

Description

This function implements unsupervised discretization into intervals of equal size.

Usage

```
D.discretize.equal.intervals.RST(decision.table, nOfIntervals = 4)
```

Arguments

decision.table   an object inheriting from the "DecisionTable" class, which represents a decision system. See `SF.asDecisionTable`.

nOfIntervals   a positive integer giving the number of intervals.

Details

This approach belongs to a class of unsupervised discretization methods since it does not consider the class labels. Each numeric attribute is divided in k intervals of equal length. Detailed information regarding this method can be found in (Dougherty et al, 1995).

It should be noted that the output of this function is an object of a class "Discretization" which contains the cut values. The function `SF.applyDecTable` has to be used in order to generate the new (discretized) decision table.

Value

An object of a class "Discretization" which stores cuts for each conditional attribute. See `D.discretization.RST`.

Author(s)

Andrzej Janusz

References

J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and Unsupervised Discretization of Continuous Features", In A. Prieditis & S. J. Russell, eds. Work. Morgan Kaufmann, p. 194-202 (1995).

See Also

`D.discretize.quantiles.RST`, `D.global.discernibility.heuristic.RST`, `D.local.discernibility.heuristic.RST`, `SF.applyDecTable`. A wrapper function for all available discretization methods: `D.discretization.RST`

Examples

```
####################################################################
## Example: Determine cut values and generate new decision table
####################################################################
data(RoughSetData)
wine.data <- RoughSetData$wine.dt
cut.values <- D.discretize.equal.intervals.RST(wine.data, nOfIntervals = 3)

## generate a new decision table
wine.discretized <- SF.applyDecTable(wine.data, cut.values)
dim(wine.discretized)
lapply(wine.discretized, unique)
```

D.discretize.quantiles.RST

                            The quantile-based discretization

---

Description

This function implements unsupervised discretization into intervals containing similar number of instances ("quantile-based").

Usage

```
D.discretize.quantiles.RST(decision.table, nOfIntervals = 4)
```

Arguments

decision.table   an object inheriting from the "DecisionTable" class, which represents a deci-
                 sion system. See SF.asDecisionTable.

nOfIntervals   a positive integer giving the number of intervals.

Details

This approach belongs to a class of unsupervised discretization methods since it does not consider the class labels. Each numeric attribute is divided in k intervals which contain approximately the same number of data instances (objects). Detailed information regarding this method can be found in (Dougherty et al, 1995).

It should be noted that the output of this function is an object of a class "Discretization" which contains the cut values. The function SF.applyDecTable has to be used in order to generate the new (discretized) decision table.

Value

An object of a class "Discretization" which stores cuts for each conditional attribute. See D.discretization.RST.

Author(s)

Andrzej Janusz

References

J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and Unsupervised Discretization of Continuous Features", In A. Prieditis & S. J. Russell, eds. Work. Morgan Kaufmann, p. 194-202 (1995).

See Also

D.discretize.equal.intervals.RST, D.global.discernibility.heuristic.RST, D.local.discernibility.heuris SF.applyDecTable. A wrapper function for all available discretization methods: D.discretization.RST

Examples

```
#####################################################################
## Example: Determine cut values and generate new decision table
#####################################################################
data(RoughSetData)
wine.data <- RoughSetData$wine.dt
cut.values <- D.discretize.quantiles.RST(wine.data, nOfIntervals = 5)

## generate a new decision table
wine.discretized <- SF.applyDecTable(wine.data, cut.values)
dim(wine.discretized)
lapply(wine.discretized, unique)
```

D.global.discernibility.heuristic.RST

Supervised discretization based on the maximum discernibility heuristic

---

Description

It is a function used for computing globally semi-optimal cuts using the maximum discernibility heuristic.

Usage

```
D.global.discernibility.heuristic.RST(decision.table, maxNOfCuts = 2 *
  ncol(decision.table), attrSampleSize = ncol(decision.table) - 1,
  cutCandidatesList = NULL, discFunction = global.discernibility, ...)
```

Arguments

decision.table an object inheriting from the "DecisionTable" class, which represents a deci-
                sion system. See `SF.asDecisionTable`. It should be noted that for this partic-
                ular method all conditional attributes must be numeric.

maxNOfCuts   a positive integer indicating the maximum number of allowed cuts.

attrSampleSize an integer between 1 and the number of conditional attributes (the default). It
                indicates the attribute sample size for the Monte Carlo selection of candidating
                cuts.

cutCandidatesList

                an optional list containing candidates for optimal cut values. By default the
                candidating cuts are determined automatically.

discFunction   a function used for computation of cuts. Currently only one implementation of
                maximu discernibility heuristic is available (the default). However, this param-
                eter can be used to integrate custom implementations of discretization functions
                with the `RoughSets` package.

...   additional parameters to the discFunction (currently unsupported).

Details

A complete description of the implemented algorithm can be found in (Nguyen, 2001).

It should be noted that the output of this function is an object of a class "Discretization" which
contains the cut values. The function `SF.applyDecTable` has to be used in order to generate the
new (discretized) decision table.

Value

An object of a class "Discretization" which stores cuts for each conditional attribute. See
`D.discretization.RST`.

Author(s)

Andrzej Janusz

References

S. H. Nguyen, "On Efficient Handling of Continuous Attributes in Large Data Bases", Fundamenta
Informaticae, vol. 48, p. 61 - 81 (2001).

Jan G. Bazan, Hung Son Nguyen, Sinh Hoa Nguyen, Piotr Synak, and Jakub Wroblewski, "Rough
Set Algorithms in Classification Problem", Chapter 2 In: L. Polkowski, S. Tsumoto and T.Y. Lin
(eds.): Rough Set Methods and Applications Physica-Verlag, Heidelberg, New York, p. 49 - 88 (
2000).

See Also

`D.discretize.quantiles.RST`, `D.discretize.equal.intervals.RST`, `D.local.discernibility.heuristic.RST`
and `SF.applyDecTable`. A wrapper function for all available discretization methods: `D.discretization.RST`

Examples

```
####################################################################
## Example: Determine cut values and generate new decision table
####################################################################
data(RoughSetData)
wine.data <- RoughSetData$wine.dt
cut.values <- D.global.discernibility.heuristic.RST(wine.data)

## generate a new decision table:
wine.discretized <- SF.applyDecTable(wine.data, cut.values)
dim(wine.discretized)
lapply(wine.discretized, unique)

## remove attributes with only one possible value:
to.rm.idx <- which(sapply(lapply(wine.discretized, unique), function(x) length(x) == 1))
to.rm.idx
wine.discretized.reduced <- wine.discretized[-to.rm.idx]
dim(wine.discretized.reduced)

## check whether the attributes in the reduced data are a super-reduct of the original data:
colnames(wine.discretized.reduced)
class.idx <- which(colnames(wine.discretized.reduced) == "class")
sum(duplicated(wine.discretized.reduced)) == sum(duplicated(wine.discretized.reduced[-class.idx]))
## yes it is
```

D.local.discernibility.heuristic.RST

Supervised discretization based on the local discernibility heuristic

---

Description

It is a function used for computing locally semi-optimal cuts using the local discernibility heuristic.

Usage

```
D.local.discernibility.heuristic.RST(decision.table, maxNOfCuts = 2,
  cutCandidatesList = NULL, discFunction = local.discernibility)
```

Arguments

decision.table an object inheriting from the "DecisionTable" class, which represents a deci-
                sion system. See SF.asDecisionTable. It should be noted that for this partic-
                ular method all conditional attributes must be numeric.

maxNOfCuts   a positive integer indicating the maximum number of allowed cuts on a single
                attribute.

cutCandidatesList

                an optional list containing candidates for optimal cut values. By default the
                candidating cuts are determined automatically.

discFunction   a function used for computation of cuts. Currently only one implementation of
                    the local discernibility heuristic is available (the default). However, this param-
                    eter can be used to integrate custom implementations of discretization functions
                    with the RoughSets package.

Details

A local (univariate) version of the algorithm described in (Nguyen, 2001) and (Bazan et al., 2000).

The output of this function is an object of a class "Discretization" which contains cut values.
The function SF.applyDecTable has to be used in order to generate the new (discretized) decision
table.

Value

An object of a class "Discretization" which stores cuts for each conditional attribute. See
D.discretization.RST.

Author(s)

Andrzej Janusz

References

S. H. Nguyen, "On Efficient Handling of Continuous Attributes in Large Data Bases", Fundamenta
Informaticae, vol. 48, p. 61 - 81 (2001).

Jan G. Bazan, Hung Son Nguyen, Sinh Hoa Nguyen, Piotr Synak, and Jakub Wroblewski, "Rough
Set Algorithms in Classification Problem", Chapter 2 In: L. Polkowski, S. Tsumoto and T.Y. Lin
(eds.): Rough Set Methods and Applications Physica-Verlag, Heidelberg, New York, p. 49 - 88 (
2000).

See Also

D.discretize.quantiles.RST, D.discretize.equal.intervals.RST, D.global.discernibility.heuristic.RST
and SF.applyDecTable. A wrapper function for all available discretization methods: D.discretization.RST

Examples

```
###################################################################
## Example: Determine cut values and generate new decision table
###################################################################
data(RoughSetData)
wine.data <- RoughSetData$wine.dt
cut.values <- D.local.discernibility.heuristic.RST(wine.data)

## generate a new decision table:
wine.discretized <- SF.applyDecTable(wine.data, cut.values)
dim(wine.discretized)
lapply(wine.discretized, unique)
```

FS.all.reducts.computation

A function for computing all decision reducts of a decision system

Description

A wrapper function used for generating all decision reducts of a decision system. The reducts are obtained from a discernibility matrix which can be computed using methods based on RST and FRST. Therefore, it should be noted that before calling the function, we need to compute a discernibility matrix using BC.discernibility.mat.RST or BC.discernibility.mat.FRST.

Usage

```
FS.all.reducts.computation(discernibilityMatrix)
```

Arguments

discernibilityMatrix

an "DiscernibilityMatrix" object representing a discernibility matrix of a decision system.

Value

An object of a class "ReductSet".

Author(s)

Andrzej Janusz

See Also

BC.discernibility.mat.RST, BC.discernibility.mat.FRST.

Examples

```
#########################################################
## Example 1: Generate all reducts and
##            a new decision table using RST
#########################################################
data(RoughSetData)
decision.table <- RoughSetData$hiring.dt

## build the decision-relation discernibility matrix
res.2 <- BC.discernibility.mat.RST(decision.table, range.object = NULL)

## generate all reducts
reduct <- FS.all.reducts.computation(res.2)

## generate new decision table
```

```
new.decTable <- SF.applyDecTable(decision.table, reduct, control = list(indx.reduct = 1))

###############################################################
## Example 2: Generate all reducts and
##             a new decision table using FRST
###############################################################
## Not run: data(RoughSetData)
decision.table <- RoughSetData$hiring.dt

## build the decision-relation discernibility matrix
control.1 <- list(type.relation = c("crisp"),
                  type.aggregation = c("crisp"),
                  t.implicator = "lukasiewicz", type.LU = "implicator.tnorm")
res.1 <- BC.discernibility.mat.FRST(decision.table, type.discernibility = "standard.red",
                                    control = control.1)

## generate single reduct
reduct <- FS.all.reducts.computation(res.1)

## generate new decision table
new.decTable <- SF.applyDecTable(decision.table, reduct, control = list(indx.reduct = 1))
## End(Not run)
```

FS.DAAR.heuristic.RST   *The DAAR heuristic for computation of decision reducts*

Description

This function implements the Dynamically Adjusted Approximate Reducts heuristic (DAAR) for feature selection based on RST. The algorithm modifies the greedy approach to selecting attributes by introducing an additional stop condition. The algorithm stops when a random probe (permutation) test fails to reject a hypothesis that the selected attribute introduces illusionary dependency in data (in a context of previously selected attributes).

Usage

```
FS.DAAR.heuristic.RST(decision.table,
  attrDescriptions = attr(decision.table, "desc.attrs"),
  decisionIdx = attr(decision.table, "decision.attr"),
  qualityF = X.gini, nAttrs = NULL,
  allowedRandomness = 1/ncol(decision.table),
  nOfProbes = max(ncol(decision.table), 100),
  permsWithinINDclasses = FALSE, semigreedy = FALSE,
  inconsistentDecisionTable = NULL)
```

Arguments

decision.table   an object of a "DecisionTable" class representing a decision table. See SF.asDecisionTable.

attrDescriptions

> a list containing possible values of attributes (columns) in \ codedecision.table. It usually corresponds to attr(decision.table, "desc.attrs").

decisionIdx   an integer value representing an index of the decision attribute.

qualityF   a function used for computation of the quality of attribute subsets. Currently, the following functions are included:

- X.entropy: See X.entropy.
- X.gini: See X.gini.
- X.nOfConflicts: See X.nOfConflicts.

nAttrs   an integer between 1 and the number of conditional attributes. It indicates the attribute sample size for the Monte Carlo selection of candidating attributes. If set to NULL (default) all attributes are used and the algorithm changes to a standard greedy method for computation of decision reducts.

allowedRandomness

> a threshold for attribute relevance. Computations will be terminated when the relevance of a selected attribute fall below this threshold.

nOfProbes   a number of random probes used for estimating the attribute relevance (see the references).

permsWithinINDclasses

> a logical value indicating whether the permutation test should be conducted within indescernibility classes.

semigreedy   a logical indicating whether the semigreedy heuristic should be used for selecting the best attribute in each iteration of the algorithm

inconsistentDecisionTable

> a logical indicating whether the decision table is suspected to be inconsistent or NULL (the default) which indicated that a test should be made to determine the data consistency.

Details

As in the case of FS.greedy.heuristic.reduct.RST the implementation can use different attribute subset quality functions (parameter qualityF) and Monte Carlo generation of candidating attributes (parameter nAttrs).

Value

A class "FeatureSubset" that contains the following components:

- reduct: a list representing a single reduct. In this case, it could be a superreduct or just a subset of features.
- type.method: a string representing the type of method which is "greedy.heuristic".
- type.task: a string showing the type of task which is "feature selection".
- model: a string representing the type of model. In this case, it is "RST" which means rough set theory.
- relevanceProbabilities: an intiger vector with estimated relevances of selected attributes.
- epsilon: a value between 0 and 1 representing the estimated approximation threshold.

Author(s)

Andrzej Janusz

References

A. Janusz and D. Ślęzak, "Random Probes in Computation and Assessment of Approximate Reducts", Proceedings of RSEISP 2014, Springer, LNCS vol. 8537: p. 53 - 64 (2014).

Andrzej Janusz and Dominik Slezak. "Computation of approximate reducts with dynamically adjusted approximation threshold". In Proceedings of ISMIS 2015, LNCS volume 9384, pages 19–28. Springer, 2015.

A. Janusz and S. Stawicki, "Applications of Approximate Reducts to the Feature Selection Problem", Proceedings of International Conference on Rough Sets and Knowledge Technology (RSKT), vol. 6954, p. 45 - 50 (2011).

See Also

`FS.greedy.heuristic.reduct.RST` and `FS.reduct.computation`.

Examples

```
####################################################
## Example 1: Evaluate reduct and generate
##            new decision table
####################################################
data(RoughSetData)
decision.table <- RoughSetData$hiring.dt

## evaluate a single reduct
res.1 <- FS.DAAR.heuristic.RST(decision.table)

## generate a new decision table corresponding to the reduct
new.decTable <- SF.applyDecTable(decision.table, res.1)
```

FS.feature.subset.computation

The superreduct computation based on RST and FRST

Description

This function is a wrapper for computing different types of decision superreducts (i.e. attribute subsets which do not lose any information regarding the decisions but are not require to be irreducable).

Usage

```
FS.feature.subset.computation(decision.table,
  method = "greedy.heuristic.superreduct", ...)
```

Arguments

decision.table	an object of a "DecisionTable" class representing a decision table. See SF.asDecisionTable.
method	a character representing the type of a method to use for computations. See in Section Details.
...	other parameters corresponding to the chosen method.

Details

Currently, there are implemented three methods that can be used with this function:

- "greedy.heuristic.superreduct": it is a greedy heuristic method which employs several quality measures from RST. See FS.greedy.heuristic.superreduct.RST.

- "quickreduct.frst": it is a feature selection function based on the fuzzy QuickReduct algorithm on FRST. See FS.quickreduct.FRST.

- "quickreduct.rst": it is a feature selection function based on the RST QuickReduct algorithm. See FS.quickreduct.RST.

These methods can be selected by assigning an appropriate value of the parameter method. Additionally, SF.applyDecTable is provided to generate the new decision table.

Value

A class "FeatureSubset".

Author(s)

Andrzej Janusz

See Also

FS.greedy.heuristic.superreduct.RST, FS.quickreduct.RST, FS.quickreduct.FRST.

Examples

```
###############################################################
## Example 1: generate reduct and new decision table using RST
###############################################################
data(RoughSetData)
decision.table <- RoughSetData$hiring.dt

## generate single superreduct
res.1 <- FS.feature.subset.computation(decision.table,
                                       method = "quickreduct.rst")

## generate new decision table according to the reduct
new.decTable <- SF.applyDecTable(decision.table, res.1)


###############################################################
## Example 2: generate reduct and new decision table using FRST
###############################################################
```

```
data(RoughSetData)
decision.table <- RoughSetData$housing7.dt

## generate single superreduct
res.2 <- FS.feature.subset.computation(decision.table,
                                       method = "quickreduct.frst")

## generate new decision table according to the reduct
new.decTable <- SF.applyDecTable(decision.table, res.2)
```

FS.greedy.heuristic.reduct.RST

The greedy heuristic algorithm for computing decision reducts and approximate decision reducts

---

Description

This function implements a greedy heuristic algorithm for computing decision reducts (or approximate decision reducts) based on RST.

Usage

```
FS.greedy.heuristic.reduct.RST(decision.table,
  attrDescriptions = attr(decision.table, "desc.attrs"),
  decisionIdx = attr(decision.table, "decision.attr"),
  qualityF = X.gini, nAttrs = NULL, epsilon = 0,
  inconsistentDecisionTable = NULL)
```

Arguments

decision.table   an object of a "DecisionTable" class representing a decision table. See `SF.asDecisionTable`.

attrDescriptions

                a list containing possible values of attributes (columns) in `decision.table`. It usually corresponds to `attr(decision.table, "desc.attrs")`.

decisionIdx   an integer value representing an index of the decision attribute.

qualityF   a function used for computation of the quality of attribute subsets. Currently, the following functions are included:

          • X.entropy: See `X.entropy`.

          • X.gini: See `X.gini`.

          • X.nOfConflicts: See `X.nOfConflicts`.

nAttrs   an integer between 1 and the number of conditional attributes. It indicates the attribute sample size for the Monte Carlo selection of candidating attributes. If set to NULL (default) all attributes are used and the algorithm changes to a standard greedy method for computation of decision reducts.

epsilon   a numeric value between [0, 1) representing an approximate threshold. It indi-
                    cates whether to compute approximate reducts or not. If it equals 0 (the default)
                    a standard decision reduct is computed.

inconsistentDecisionTable
                    logical indicating whether the decision table is suspected to be inconsistent or
                    NULL (the default) which indicated that a test should be made to determine the
                    data consistency.

Details

In this implementation, we provided some attribute subset quality measures which can be passed
to the algorithm by the parameter `qualityF`. Those measures guide the computations in the search
for a decision/approximated reduct. They are used to assess amount of information gained after
addition of an attribute. For example, `X.entropy` corresponds to the information gain measure.

Additionally, this function can use the value of `epsilon` parameter in order to compute ϵ-approximate
reducts. The ϵ-approximate can be defined as an irreducable subset of attributes B, such that:

$$Quality_{\mathcal{A}}(B) \geq (1 - \epsilon)Quality_{\mathcal{A}}(A),$$

where $Quality_{\mathcal{A}}(B)$ is the value of a quality measure (see possible values of the parameter `qualityF`)
for an attribute subset B in decision table $\mathcal{A}$ and ϵ is a numeric value between 0 and 1 expressing
the approximation threshold. A lot of monographs provide comprehensive explanations about this
topics, for example (Janusz and Stawicki, 2011; Slezak, 2002; Wroblewski, 2001) which are used
as the references of this function.

Finally, this implementation allows to restrain the computational complexity of greedy searching for
decision reducts by setting the value of the parameter `nAttrs`. If this parameter is set to a positive
integer, the Monte Carlo method of selecting candidating attributes will be used in each iteration of
the algorithm.

Value

A class `"FeatureSubset"` that contains the following components:

- reduct: a list representing a single reduct. In this case, it could be a superreduct or just a
  subset of features.

- type.method: a string representing the type of method which is `"greedy.heuristic"`.

- type.task: a string showing the type of task which is `"feature selection"`.

- model: a string representing the type of model. In this case, it is `"RST"` which means rough
  set theory.

- epsilon: the approximation threshold.

Author(s)

Andrzej Janusz

References

Andrzej Janusz and Dominik Slezak. "Rough Set Methods for Attribute Clustering and Selection". Applied Artificial Intelligence, 28(3):220–242, 2014.

A. Janusz and S. Stawicki, "Applications of Approximate Reducts to the Feature Selection Problem", Proceedings of International Conference on Rough Sets and Knowledge Technology (RSKT), vol. 6954, p. 45 - 50 (2011).

D. Ślęzak, "Approximate Entropy Reducts", Fundamenta Informaticae, vol. 53, no. 3 - 4, p. 365 - 390 (2002).

J. Wróblewski, "Ensembles of Classifiers Based on Approximate Reducts", Fundamenta Informaticae, vol. 47, no. 3 - 4, p. 351 - 360 (2001).

See Also

`FS.DAAR.heuristic.RST` and `FS.reduct.computation`.

Examples

```
#####################################################
## Example 1: Evaluate reduct and generate
##            new decision table
#####################################################
data(RoughSetData)
decision.table <- RoughSetData$hiring.dt

## evaluate a single reduct
res.1 <- FS.greedy.heuristic.reduct.RST(decision.table, qualityF = X.entropy,
                                        epsilon = 0.0)

## generate a new decision table corresponding to the reduct
new.decTable <- SF.applyDecTable(decision.table, res.1)
```

FS.greedy.heuristic.superreduct.RST

The greedy heuristic method for determining superreduct based on RST

Description

It is used to get a feature subset (superreduct) based on the greedy heuristic algorithm employing some quality measurements. Regarding the quality measurements, the detailed description can be seen in `FS.greedy.heuristic.reduct.RST`.

Usage

```
FS.greedy.heuristic.superreduct.RST(decision.table,
  attrDescriptions = attr(decision.table, "desc.attrs"),
  decisionIdx = attr(decision.table, "decision.attr"),
  qualityF = X.gini, nAttrs = NULL, inconsistentDecisionTable = NULL)
```

Arguments

decision.table   an object of a "DecisionTable" class representing a decision table. See `SF.asDecisionTable`.

attrDescriptions

        a list containing possible values of attributes (columns) in decision.table. It usually corresponds to attr(decision.table, "desc.attrs").

decisionIdx   a integer value representing an index of decision attribute.

qualityF   a function for calculating a quality of an attribute subset. See `FS.greedy.heuristic.reduct.RST`.

nAttrs   an integer between 1 and the number of conditional attributes. It indicates the attribute sample size for the Monte Carlo selection of candidating attributes. If set to NULL (default) all attributes are used and the algorithm changes to a standard greedy method for computation of decision reducts.

inconsistentDecisionTable

       logical indicating whether the decision table is suspected to be inconsistent or NULL (the default) which indicated that a test should be made to determine the data consistency.

Value

A class "FeatureSubset" that contains the following components:

- reduct: a list representing a single reduct. In this case, it could be a superreduct or just a subset of features.
- type.method: a string representing the type of method which is "greedy.heuristic.superreduct".
- type.task: a string showing the type of task which is "feature selection".
- model: a string representing the type of model. In this case, it is "RST" which means rough set theory.

Author(s)

Andrzej Janusz

References

Andrzej Janusz and Dominik Slezak. "Rough Set Methods for Attribute Clustering and Selection". Applied Artificial Intelligence, 28(3):220–242, 2014.

A. Janusz and S. Stawicki, "Applications of Approximate Reducts to the Feature Selection Problem", Proceedings of International Conference on Rough Sets and Knowledge Technology (RSKT), vol. 6954, p. 45 - 50 (2011).

D. Ślęzak, "Approximate Entropy Reducts", Fundamenta Informaticae, vol. 53, no. 3 - 4, p. 365 - 390 (2002).

J. Wroblewski, "Ensembles of Classifiers Based on Approximate Reducts", Fundamenta Informaticae, vol. 47, no. 3 - 4, p. 351 - 360 (2001).

See Also

`FS.quickreduct.RST` and `FS.feature.subset.computation`.

Examples

```
#####################################################
## Example 1: Evaluate reduct and generate
##             new decision table
#####################################################
data(RoughSetData)
decision.table <- RoughSetData$hiring.dt

## evaluate single reduct
res.1 <- FS.greedy.heuristic.superreduct.RST(decision.table, qualityF = X.nOfConflicts)
print(res.1)

## generate new decision table according to the reduct
new.decTable <- SF.applyDecTable(decision.table, res.1)
```

FS.nearOpt.fvprs.FRST   *The near-optimal reduction algorithm based on fuzzy rough set theory*

Description

This is a function implementing the near-optimal reduction algorithm by employing fuzzy variable precision rough sets (FVPRS) for feature selection based on FRST proposed by (Zhao et al, 2009).

Usage

```
FS.nearOpt.fvprs.FRST(decision.table, alpha.precision = 0.05)
```

Arguments

decision.table an object of a "DecisionTable" class representing a decision table. See `SF.asDecisionTable`. In this case, the decision attribute must be nominal.

alpha.precision

a numeric value representing variable precision of FVPRS.

See `BC.LU.approximation.FRST`.

Details

The near-optimal algorithm is an algorithm to find one reduct only rather than all reducts. It modifies the α-reduction based on discernibility matrix by using a heuristic algorithm. To get basic knowledge about discernibility matrix, users can refers to the "alpha.red" discernibility type in `BC.discernibility.mat.FRST`.

It should be noted that this function does not give the new decision table directly. The other additional function called `SF.applyDecTable` is used to produce the new decision table based on information about the reduct from this function.

Value

A class ″FeatureSubset″ that contains the following components:

- reduct: a list representing a single reduct. In this case, it could be a superreduct or just a subset of features.
- type.method: a string representing the type of method which is ″near.optimal.fvprs″.
- type.task: a string showing the type of task which is ″feature selection″.
- model: a string representing the type of model. In this case, it is ″FRST″ which means fuzzy rough set theory.

Author(s)

Lala Septem Riza

References

S. Zhao, E. C. C. Tsang, and D. Chen, "The Model of Fuzzy Variable Precision Rough Sets", IEEE Trans. on Fuzzy Systems, vol. 17, no. 2, p. 451 - 467 (2009).

See Also

BC.discernibility.mat.FRST

Examples

```
#########################################################
## Example 1: Hiring dataset containing 8 objects with 5 attributes
#########################################################
## Not run: data(RoughSetData)
decision.table <- RoughSetData$hiring.dt

## get reduct as FeatureSubset class
reduct.1 <- FS.nearOpt.fvprs.FRST(decision.table)

## get new decision table according to the reduct
new.decTable <- SF.applyDecTable(decision.table, reduct.1)
## End(Not run)

#########################################################
## Example 2: Pima dataset containing 7 objects with 9 attributes
#########################################################
data(RoughSetData)
decision.table <- RoughSetData$pima7.dt

## get reduct
reduct.2 <- FS.nearOpt.fvprs.FRST(decision.table)

## get new decision table according to the reduct
new.decTable <- SF.applyDecTable(decision.table, reduct.2)
```

FS.one.reduct.computation

Computing one reduct from a discernibility matrix

Description

It is a function for computing one reduct from a discernibility matrix - it can use the greedy heuristic or a randomized (Monte Carlo) search.

Usage

```
FS.one.reduct.computation(discernibilityMatrix, greedy = TRUE,
  power = 1)
```

Arguments

discernibilityMatrix

a "DiscernibilityMatrix" class representing the discernibility matrix of RST and FRST.

greedy   a boolean value indicating whether the greedy heuristic or a randomized search should be used in computations.

power   a numeric representing a parameter of the randomized search heuristic.

Value

An object of a class "ReductSet".

Author(s)

Andrzej Janusz

References

Jan G. Bazan, Hung Son Nguyen, Sinh Hoa Nguyen, Piotr Synak, and Jakub Wroblewski, "Rough Set Algorithms in Classification Problem", Chapter 2 In: L. Polkowski, S. Tsumoto and T.Y. Lin (eds.): Rough Set Methods and Applications Physica-Verlag, Heidelberg, New York, p. 49 - 88 (2000).

See Also

BC.discernibility.mat.RST and BC.discernibility.mat.FRST.

Examples

```
########################################################
## Example 1: Generate one reduct and
##             a new decision table using RST
########################################################
data(RoughSetData)
decision.table <- RoughSetData$hiring.dt

## build the decision-relation discernibility matrix
res.1 <- BC.discernibility.mat.RST(decision.table)

## generate all reducts
reduct <- FS.one.reduct.computation(res.1)

## generate new decision table
new.decTable <- SF.applyDecTable(decision.table, reduct, control = list(indx.reduct = 1))


##########################################################
## Example 2: Generate one reduct and
##             a new decision table using FRST
##########################################################
data(RoughSetData)
decision.table <- RoughSetData$hiring.dt

## build the decision-relative discernibility matrix
control <- list(type.relation = c("crisp"),
                type.aggregation = c("crisp"),
                t.implicator = "lukasiewicz", type.LU = "implicator.tnorm")
res.2 <- BC.discernibility.mat.FRST(decision.table, type.discernibility = "standard.red",
                                    control = control)

## generate a single reduct
reduct <- FS.one.reduct.computation(res.2)

## generate new decision table
new.decTable <- SF.applyDecTable(decision.table, reduct, control = list(indx.reduct = 1))
```

FS.permutation.heuristic.reduct.RST

The permutation heuristic algorithm for computation of a decision reduct

---

Description

It is a function implementing the permutation heuristic approach based on RST.

Usage

```
FS.permutation.heuristic.reduct.RST(decision.table, permutation = NULL,
  decisionIdx = ncol(decision.table))
```

Arguments

decision.table   an object of a "DecisionTable" class representing a decision table. See SF.asDecisionTable.

permutation   a logical value, an integer vector orNULL (the default). If an integer vector with a length equal the cardinality of the conditional attribute set of the decision table is given (it must contain a permutation of integers from 1:(ncol(decision.table) - 1)), then it will define the elimination order. Otherwise, if permutation is NULL or TRUE a random permutation will be generated. In the case when permutation is FALSE, the elimination will be performed in the order of attributes in the decision system.

decisionIdx   an index of the decision attribute. The default value is the last column of a decision table.

Details

Basically there are two steps in this algorithm which are

- generating feature subset as a superreduct: In this step, we choose a subset of attributes that discern all object from different decision classes. It is done by adding consecutive attributes in an order defined by a permutation of attribute indices. The permutation can be random or it can be explicitly given (by the parameter permutation).

- iterative elimination of attributes from the set obtained in the previous step. It is done in the reverse order to that, defined by the permutation.

More details regarding this algorithm can be found in (Janusz and Slezak, 2012).

Additionally, SF.applyDecTable has been provided to generate new decision table.

Value

A class "FeatureSubset" that contains the following components:

- reduct: a list representing a single reduct. In this case, it could be a superreduct or just a subset of features.
- type.method: a string representing the type of method which is "permutation.heuristic".
- type.task: a string showing the type of task which is "feature selection".
- model: a string representing the type of model. In this case, it is "RST" which means rough set theory.
- epsilon: the approximation threshold.

Author(s)

Andrzej Janusz

References

A. Janusz and D. Ślęzak, "Utilization of Attribute Clustering Methods for Scalable Computation of Reducts from High-Dimensional Data". Proceedings of Federated Conference on Computer Science and Information Systems - FedCSIS, p. 295 - 302 (2012).

Andrzej Janusz and Dominik Slezak. "Rough Set Methods for Attribute Clustering and Selection". Applied Artificial Intelligence, 28(3):220–242, 2014.

See Also

FS.quickreduct.RST and FS.reduct.computation.

Examples

```
#####################################################
## Example 1: Generate reduct and new decision table
#####################################################
data(RoughSetData)
decision.table <- RoughSetData$hiring.dt

## generate single reduct
res.1 <- FS.permutation.heuristic.reduct.RST(decision.table,
                                             permutation = NULL,
                                             decisionIdx = 5)
print(res.1)

res.2 <- FS.permutation.heuristic.reduct.RST(decision.table,
                                             permutation = 4:1,
                                             decisionIdx = 5)
print(res.2)

## generate new decision table according to the reduct
new.decTable <- SF.applyDecTable(decision.table, res.1)
```

FS.quickreduct.FRST   *The fuzzy QuickReduct algorithm based on FRST*

Description

It is a function implementing the fuzzy QuickReduct algorithm for feature selection based on FRST.
The fuzzy QuickReduct is a modification of QuickReduct based on RST (see FS.quickreduct.RST).

Usage

```
FS.quickreduct.FRST(decision.table, type.method = "fuzzy.dependency",
  type.QR = "fuzzy.QR", control = list())
```

Arguments

decision.table an object of a "DecisionTable" class representing a decision table. See SF.asDecisionTable.

type.method   a string representing the type of methods. The complete description can be found in Section Details.

type.QR   a string expressing the type of QuickReduct algorithm which is one of the two following algorithms:

- "fuzzy.QR": it is the original fuzzy rough QuickReduct algorithm based on (Jensen and Shen, 2002).

- "modified.QR": it is the modified QuickReduct algorithm based on (Bhatt and Gopal, 2005).

control   a list of other parameters as follows.

- type.aggregation: a type of aggregation operator. See BC.IND.relation.FRST.
- t.implicator: a type of implicator function. See BC.LU.approximation.FRST. The default value is "lukasiewicz".
- type.relation: a type of indiscernibility relation. See BC.IND.relation.FRST. The default value is type.relation = c("tolerance", "eq.3").
- alpha: a real number between 0 and 1 expressing a threshold value or stopping criterion. The following methods use the parameter: "vqrs", "min.positive.reg", and "fuzzy.discernibility". The default value is 0.95.
- alpha.precision: a real number between 0 and 1 expressing variable precision (α) for "fvprs". See BC.LU.approximation.FRST. The default value is 0.05.
- q.some: a pair of numeric values for the alpha and beta parameter of VQRS for the quantifier some. The default value is q.some = c(0.1, 0.6). See BC.LU.approximation.FRST.
- q.most: a pair of numeric values for the alpha and beta parameter of VQRS for the quantifier most. The default value is q.most = c(0.2, 1). See BC.LU.approximation.FRST.
- m.owa: a numeric value to define the parameter in OWA. The default value is the mean number of objects.
- type.rfrs: a type of robust fuzzy rough sets. The default is type.rfrs = "k.trimmed.min"). See BC.LU.approximation.FRST.
- k.rfrs: a value between 0 and length of data representing index of considered data. The default is k.rfrs = round(0.5*nrow(decision.table)). See BC.LU.approximation.FRST.
- beta.quasi: a number between 0 and 1 representing β-precision t-norms and t-conorms. The default value is 0.05.
- randomize: a boolean value to define whether selecting attributes randomly or not. For more detail, see in Section Details. The default value is FALSE.

It should be noted that instead of supplying all the above parameters, we only set those parameters needed by the considered method. See in Section Details. Also, we provide some examples to illustrate how the parameters are used.

Details

In this function, we provide an algorithm proposed by (Jensen and Shen, 2002) which is fuzzy QuickReduct. Then, the algorithm has been modified by (Bhatt and Gopal, 2005) to improve stopping criteria. This function is aimed to implement both algorithms. These algorithms can be executed by assigning the parameter type.QR with "fuzzy.QR" and "modified.QR" for fuzzy quickreduct and modified fuzzy quickreduct algorithms, respectively. Additionally, in the control parameter, we provide one component which is randomize having boolean values: TRUE or FALSE. randomize = TRUE means that we evaluate some (or not all) attributes randomly along iteration. It will be useful if we have a large number of attributes in a decision table.

In this function, we have considered many approaches of the lower and upper approximations. The following list shows considered methods and their descriptions. Additionally, those approaches can be executed by assigning the following value to the parameter `type.method`.

- `"fuzzy.dependency"`: It is based on the degree of dependency using the implication/t-norm model approximation (Jensen and Shen, 2009). The detailed concepts about this approximation have been explained in `B.Introduction-FuzzyRoughSets` and `BC.LU.approximation.FRST`.

- `"fuzzy.boundary.reg"`: It is based on the fuzzy boundary region proposed by (Jensen and Shen, 2009). This algorithm introduced the usage of the total uncertainty degree $\lambda_B(Q)$ for all concepts of feature subset B and decision attribute Q. The total uncertainty degree is used as a parameter to select appropriate features.

- `"vqrs"`: It is based on vaquely quantified rough set (VQRS) proposed by (Cornelis and Jensen, 2008). See also `BC.LU.approximation.FRST`.

- `"owa"`: Based on ordered weighted average (OWA) based fuzzy rough set, (Cornelis et al, 2010) proposed the degree of dependency as a parameter employed in the algorithm to select appropriate features. The explanation about lower and upper approximations based on OWA can be found in `BC.LU.approximation.FRST`.

- `"rfrs"`: It is based on degree of dependency that is obtained by performing the robust fuzzy rough sets proposed by (Hu et al, 2012). The detailed concepts about this approximation have been explained in `BC.LU.approximation.FRST`.

- `"min.positive.reg"`: Based on measure introduced in (Cornelis et al, 2010) which considers the most problematic element in the positive region, defined using the implicator/t-norm model.

- `"fvprs"`: It is based on degree of dependency proposed by (Zhao et al, 2009). The degree is obtained by using fuzzy lower approximation based on fuzzy variable precision rough set model.

- `"fuzzy.discernibility"`: This approach attempts to combine the the decision-relative discernibility matrix and the fuzzy QuickReduct algorithm. (Jensen and Shen, 2009) introduced a measurement which is the degree of satisfaction to select the attributes.

- `"beta.pfrs"`: Based on β-precision fuzzy rough sets (β-PFRS) proposed by (Salido and Murakami, 2003), the degree of dependency as a parameter employed in the algorithm to select appropriate features. The explanation about lower and upper approximations based on β-PFRS can be found in `BC.LU.approximation.FRST`.

It should be noted that the parameter `type.method` is related to parameter `control`. In other words, we only set the components in the `control` parameter that related to the chosen type of method. The following is a list showing the components of `control` needed by each type of methods.

- type.method = `"fuzzy.dependency"`:
  control <- list(t.implicator, type.relation, type.aggregation)

- type.method = `"fuzzy.boundary.reg"`:
  control <- list(t.implicator, type.relation, type.aggregation)

- type.method = `"vqrs"`:
  control <- list(alpha, q.some, q.most, type.aggregation)

- `type.method = "owa"`:

  `control <- list(t.implicator, type.relation, m.owa, type.aggregation)`

- `type.method = "rfrs"`:

  `control <- list(t.implicator, type.relation, type.rfrs,`
  `k.rfrs, type.aggregation)`

- `type.method = "min.positive.reg"`:

  `control <- list(alpha, t.implicator, type.relation, type.aggregation)`

- `type.method = "fuzzy.discernibility"`:

  `control <- list(alpha, t.implicator, type.relation, type.aggregation)`

- `type.method = "fvprs"`:

  `control <- list(alpha.precision, t.implicator, type.relation, type.aggregation)`

- `type.method = "beta.pfrs"`:

  `control <- list(t.implicator, type.relation, beta.quasi, type.aggregation)`

The descriptions of each component can be seen in the documentation of the `control` parameter.

It should be noted that this function does not give the new decision table directly. An additional function called `SF.applyDecTable` is used to produce new decision table based on information about the reduct from this function. See Section `Examples`.

Value

A class `"FeatureSubset"` that contains the following components:

- `reduct`: a list representing a single reduct. In this case, it could be a superreduct or just a subset of feature.
- `type.method`: a string representing the type of method.
- `type.task`: a string showing the type of task which is `"feature selection"`.
- `model`: a string representing the type of model. In this case, it is `"FRST"` which means fuzzy rough set theory.

Author(s)

Lala Septem Riza

References

C. Cornelis, G. Hurtado Martin, R. Jensen, and D. Slezak, "Feature Selection with Fuzzy Decision Reducts", Information Sciences, vol. 180, no. 2, p. 209 - 224 (2010).

C. Cornelis and R. Jensen, "A Noise-tolerant Approach to Fuzzy-rough Feature Selection", Proceedings of the 2008 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2008), p. 1598 - 1605 (2008).

Q. Hu, L. Zhang, S. An, D. Zhang, and D. Yu, "On Robust Fuzzy Rough Set Models", IEEE Trans. on Fuzzy Systems, vol. 20, no. 4, p. 636 - 651 (2012).

See Also

FS.quickreduct.RST and FS.feature.subset.computation.

Examples

```
############################################################
## Example 1: Dataset containing nominal values on all attributes
############################################################

data(RoughSetData)
decision.table <- RoughSetData$housing7.dt

########## using fuzzy lower approximation ##############
control <- list(t.implicator = "lukasiewicz", type.relation = c("tolerance", "eq.1"),
                type.aggregation = c("t.norm", "lukasiewicz"))
reduct.1 <- FS.quickreduct.FRST(decision.table, type.method = "fuzzy.dependency",
                            type.QR = "fuzzy.QR", control = control)


########## using fuzzy boundary region ##############
## Not run: control <- list(t.implicator = "lukasiewicz", type.relation = c("tolerance", "eq.1"),
                type.aggregation = c("t.norm", "lukasiewicz"))
reduct.2 <- FS.quickreduct.FRST(decision.table, type.method = "fuzzy.boundary.reg",
                            type.QR = "fuzzy.QR", control = control)


########## using vaguely quantified rough sets (VQRS) #########
control <- list(alpha = 0.9, q.some = c(0.1, 0.6), q.most = c(0.2, 1),
                type.aggregation = c("t.norm", "lukasiewicz"))
reduct.3 <- FS.quickreduct.FRST(decision.table, type.method = "vqrs",
                            type.QR = "fuzzy.QR", control = control)


########## ordered weighted average (OWA) #########
control <- list(t.implicator = "lukasiewicz", type.relation = c("tolerance", "eq.1"),
                m.owa = 3, type.aggregation = c("t.norm","lukasiewicz"))
reduct.4 <- FS.quickreduct.FRST(decision.table, type.method = "owa",
                            type.QR = "fuzzy.QR", control = control)


########## robust fuzzy rough sets (RFRS) #########
control <- list(t.implicator = "lukasiewicz", type.relation = c("tolerance", "eq.1"),
             type.rfrs = "k.trimmed.min", type.aggregation = c("t.norm", "lukasiewicz"),
                k.rfrs = 0)
reduct.5 <- FS.quickreduct.FRST(decision.table, type.method = "rfrs",
                            type.QR = "fuzzy.QR", control = control)


########## using min positive region (delta) ###########
control <- list(alpha = 1, t.implicator = "lukasiewicz",
                type.relation = c("tolerance", "eq.1"), type.aggregation =
                                c("t.norm", "lukasiewicz"))
reduct.6 <- FS.quickreduct.FRST(decision.table, type.method = "min.positive.reg",
                            type.QR = "fuzzy.QR", control = control)


########## using FVPRS approximation ##############
control <- list(alpha.precision = 0.05, t.implicator = "lukasiewicz",
```

```
                 type.aggregation = c("t.tnorm", "lukasiewicz"),
                 type.relation = c("tolerance", "eq.1"))
reduct.7 <- FS.quickreduct.FRST(decision.table, type.method = "fvprs",
                          type.QR = "fuzzy.QR", control = control)


########## using beta.PFRS approximation ##############
control <- list(t.implicator = "lukasiewicz", type.relation = c("tolerance", "eq.1"),
                 beta.quasi = 0.05, type.aggregation = c("t.tnorm", "lukasiewicz"))
reduct.8 <- FS.quickreduct.FRST(decision.table, type.method = "beta.pfrs",
                          type.QR = "fuzzy.QR", control = control)


########## using fuzzy discernibility matrix ##############
control <- list(alpha = 1, type.relation = c("tolerance", "eq.1"),
                 type.aggregation = c("t.tnorm", "lukasiewicz"),
                 t.implicator = "lukasiewicz")
reduct.9 <- FS.quickreduct.FRST(decision.table, type.method = "fuzzy.discernibility",
                          type.QR = "fuzzy.QR", control = control)
## End(Not run)


############################################################
## Example 2: Dataset containing nominal and continuous values
## In this case, we only provide one method but others work in
## the same way.
## In this example, we will show how to get the
## new decision table as well
############################################################
data(RoughSetData)
decision.table <- RoughSetData$hiring.dt


########## using fuzzy lower approximation ##############
control <- list(type.aggregation = c("t.tnorm", "lukasiewicz"),
                 t.implicator = "lukasiewicz", type.relation = c("tolerance", "eq.1"))
reduct.1 <- FS.quickreduct.FRST(decision.table, type.method = "fuzzy.dependency",
                          type.QR = "fuzzy.QR", control = control)


## get new decision table based on reduct
new.decTable <- SF.applyDecTable(decision.table, reduct.1)
```

FS.quickreduct.RST   *QuickReduct algorithm based on RST*

Description

This is a function for implementing the QuickReduct algorithm for feature selection based on RST
proposed by (Shen and Chouchoulas, 2000). The algorithm produces only one feature subset that
could be a superreduct.

Usage

```
FS.quickreduct.RST(decision.table, control = list())
```

Arguments

decision.table   an object of a ″DecisionTable″ class representing a decision table. See SF.asDecisionTable.

control   other parameters. It contains the following component:

- randomize: it has a boolean value. For the detailed description, see in Section Details. The default value is FALSE.

Details

This algorithm considers the dependency degree (see A.Introduction-RoughSets) of the addition of each attribute to the current reduct candidate. Then the best candidate will be chosen. This process continues until the dependency of the subset equals to the dependency of the full dataset.

Additionally, in control parameter, we provide one component which is randomize. It has a boolean value: TRUE or FALSE that means we want to perform quickreduct by evaluating attributes randomly or all attributes in decision table.

It should be noted that this function does not give the new decision table directly. The other additional function called SF.applyDecTable is used to produce new decision table based on information about the reduct from this function.

Value

A class ″FeatureSubset″ that contains the following components:

- reduct: a list representing single reduct. In this case, it could be super reduct or just subset of feature.
- type.method: a string representing a type of method which is ″quickreduct″.
- type.task: a string showing type of task which is ″feature selection″.
- model: a string representing a type of model. In this case, it is ″RST″ which means rough set theory.

Author(s)

Lala Septem Riza

References

Q. Shen and A. Chouchoulas, "A Modular Approach to Generating Fuzzy Rules with Reduced Attributes for the Monitoring of Complex Systems", Engineering Applications of Artificial Intelligence, vol. 13, p. 263 - 278 (2000).

See Also

FS.quickreduct.FRST

Examples

```
#####################################################
## Example 1: Evaluate reduct and generate
##             new decision table
#####################################################
data(RoughSetData)
decision.table <- RoughSetData$hiring.dt

## evaluate single reduct
res.1 <- FS.quickreduct.RST(decision.table)

## generate new decision table according to the reduct
new.decTable <- SF.applyDecTable(decision.table, res.1)
```

FS.reduct.computation   *The reduct computation methods based on RST and FRST*

Description

This function is a wrapper for computing different types of decision reducts and approximate decision reducts.

Usage

```
FS.reduct.computation(decision.table, method = "greedy.heuristic", ...)
```

Arguments

decision.table an object of a "DecisionTable" class representing a decision table. See SF.asDecisionTable.

method   a character representing the type of computation method to use. See in Section
                Details.

...   other parameters. See the parameters of FS.greedy.heuristic.reduct.RST,
                FS.DAAR.heuristic.RST, FS.nearOpt.fvprs.FRST and FS.permutation.heuristic.reduct.RST.

Details

The implemented methods include the following approaches:

- "greedy.heuristic": a greedy heuristic method for computation of decision reducts (or approximate decision reducts) based on RST. See FS.greedy.heuristic.reduct.RST.

- "DAAR.heuristic": Dynamically Adapted Approximate Reduct heuristic, which is a modification of the greedy heuristic with a random probe test to avoid inclusion of irrelevant attributes to the reduct. See FS.DAAR.heuristic.RST.

- "nearOpt.fvprs": the near-optimal reduction algorithm based on FRST. See FS.nearOpt.fvprs.FRST.

- "permutation.heuristic": a permutation-based elimination heuristic for computation of decision reducts based on RST. See FS.permutation.heuristic.reduct.RST.

Those methods can be selected by setting the parameter `method`. Additionally, `SF.applyDecTable` has been provided to generate a new decision table.

Value

An object of a class ″FeatureSubset″. See `FS.greedy.heuristic.reduct.RST`, `FS.DAAR.heuristic.RST`, `FS.permutation.heuristic.reduct.RST` or `FS.nearOpt.fvprs.FRST` for more details.

Author(s)

Andrzej Janusz

See Also

`D.discretization.RST`, `BC.LU.approximation.RST`

Examples

```
###############################################################
## Example 1: generate reduct and new decision table
## using RST and FRST
###############################################################
data(RoughSetData)
decision.table <- RoughSetData$hiring.dt

## generate a single reduct using RST
reduct.1 <- FS.reduct.computation(decision.table, method = "greedy.heuristic")

## generate a single reduct using FRST
reduct.2 <- FS.reduct.computation(decision.table, method = "nearOpt.fvprs")

## generate a new decision table using reduct.1
new.decTable.1 <- SF.applyDecTable(decision.table, reduct.1)

## generate new decision table using reduct.2
new.decTable.2 <- SF.applyDecTable(decision.table, reduct.2)
```

IS.FRIS.FRST *The fuzzy rough instance selection algorithm*

Description

This is a function that implements the fuzzy-rough instance selection (FRIS) proposed by (Jensen and Cornelis, 2010) which is used to perform instance selection.

Usage

```
IS.FRIS.FRST(decision.table, control = list())
```

Arguments

decision.table a "DecisionTable" class representing the decision table. See `SF.asDecisionTable`.

control   a list of other parameters which are

- threshold.tau: a value determining whether an object can be removed or not. The object can be removed if it is less than the threshold. The default value is 0.95.
- alpha: a parameter determining the granularity of the fuzzy similarity measure, which has positive values (>= 0). The default value is 1.
- type.aggregation: a list representing the type of aggregation and its value. The default value is type.aggregation = c("t.tnorm", "lukasiewicz"). See `BC.IND.relation.FRST`.
- t.implicator: a string representing the value of implicator function. The default value is "lukasiewicz". See `BC.LU.approximation.FRST`.

Details

FRIS is used to remove training instances that cause conflicts with other instances as determined by the fuzzy positive region.

This algorithm evaluates the degree of membership of each instance to the fuzzy positive region. If there is a instance less than the threshold, then the instance can be removed. Additionally, it uses a fuzzy indiscernibility relation R_a to express the approximate equality between objects x and y on attribute a in the training set:

$$R_a^\alpha(x, y) = max(0, 1 - \alpha \frac{|a(x) - a(y)|}{l(a)})$$

where parameter α ($\alpha \geq 0$) determines the granularity of R_a^α. Then, the fuzzy B-indiscernibility relation, fuzzy lower approximation, positive region and degree of dependency are calculated based on the concept in `B.Introduction-FuzzyRoughSets`.

It should be noted that this function does not give the new decision table directly. The other additional function called `SF.applyDecTable` is used to produce the new decision table based on the output of this function.

Value

A class "InstanceSelection" that contains the following components:

- indx.objects: it contains all indexes of objects that are selected.
- type.method: a string representing the type of method. In this case, it is "IS.FRIS".
- type.task: a string showing the type of task which is "instance selection".
- type.model: a string representing the type of model which is "FRST".

Author(s)

Lala Septem Riza

References

R. Jensen and C. Cornelis, "Fuzzy-rough Instance Selection", Proceedings of the 19th International Conference on Fuzzy Systems (FUZZ-IEEE 2010), p. 1776 - 1782 (2010).

See Also

IS.FRPS.FRST.

Examples

```
###############################################
## Example: Evaluate instances/objects and
##          generate new decision table
###############################################
dt.ex1 <- data.frame(c(0.1, 0.5, 0.2, 0.3, 0.2, 0.2, 0.8),
                 c(0.1, 0.1, 0.4, 0.2, 0.4, 0.4, 0.5), c(0, 0, 0, 0, 1, 1, 1))
colnames(dt.ex1) <- c("a1", "a2", "d")
decision.table <- SF.asDecisionTable(dataset = dt.ex1, decision.attr = 3, indx.nominal = c(3))

## evaluate index of objects
res.1 <- IS.FRIS.FRST(decision.table = decision.table, control =
                       list(threshold.tau = 0.5, alpha = 0.8,
                       type.aggregation = c("t.tnorm", "lukasiewicz"),
                       t.implicator = "lukasiewicz"))

## generate new decision table
new.decTable <- SF.applyDecTable(decision.table, res.1)
```

IS.FRPS.FRST   *The fuzzy rough prototype selection method*

Description

This is a function for implementing instance selection using prototype selection method (FRPS) proposed by (Verbiest et al, 2013).

Usage

```
IS.FRPS.FRST(decision.table, type.alpha = "FRPS.1")
```

Arguments

decision.table   a "DecisionTable" class representing the decision table. See SF.asDecisionTable.

type.alpha   type of FRPS expressing the equations of α. The default value is "FRPS.1". See Section Details.

Details

This algorithm uses prototype selection (PS) to improve the accuracy of the k-nearest neighbour (kNN) method. It selects a subset of instances $S \subseteq X$ and then classifies a new instance t using the kNN rule acting over S instead of over X. Based on fuzzy rough set theory, S is built. Basically, two steps are performed in the algorithm. First, the instances are ordered according to a measure

called α which is based on fuzzy rough set theory that evaluates the lack of predictive ability of the instances, and the instances for which the values exceeds a certain threshold are removed from training set. To determine this threshold, we consider several equations which are applied on all instances. These equations are constructed by considering fuzzy positive region membership degree on several implication and t-norm operators. The following is the equations of α:

- "FRPS.1": $max_{y \notin [x]_d} \frac{1}{max_{i=1}^m \delta_{a_i}(x,y)}$

- "FRPS.2": $OWA_w \left(\frac{1}{OWA_w \delta_{a_i}(x,y)} \right)$

- "FRPS.3": $max_{y \notin [x]_d} \dfrac{1}{\displaystyle\sum_{i=1}^m \delta_{a_i}(x,y)}$

- "FRPS.4": $OWA_w \left(\dfrac{1}{\displaystyle\sum_{i=1}^m \delta_{a_i}(x,y)} \right)$

where $[x]_d$ and OWA_w are equivalence class on the decision attribute and ordered weighted average operator, respectively. And $\delta_{a_i}(x,y)$ is a distance measure of the attribute a_i between x and y. After that, `1knn` will be performed in order to select and get prototype S which produces the highest training accuracy.

It should be noted that this function does not give the new decision table directly. The other additional function called `SF.applyDecTable` is used to produce new decision table based on the output of this function.

Value

A class "InstanceSelection" that contains the following components:

- `indx.objects`: it contains all indexes of objects that are selected.

- `type.method`: a string representing a type of method. In this case, it is "IS.FRPS".

- `type.task`: a string showing the type of task which is "instance selection".

- `type.model`: a string representing the type of model which is "FRST". It means fuzzy rough set theory.

Author(s)

Lala Septem Riza

References

N. Verbiest, C. Cornelis, and F. Herrera, "A Fuzzy Rough Prototype Selection Method", Pattern Recognition, Vol. 46, no. 10, p. 2770 - 2782 (2013).

See Also

IS.FRIS.FRST

Examples

```
################################################
## Example: Evaluate instances/objects and
## generate new decision table
################################################
dt.ex1 <- data.frame(c(0.5, 0.2, 0.3, 0.7, 0.2, 0.2), c(0.1, 0.4, 0.2, 0.8, 0.4, 0.4),
                     c(0, 0, 0, 1, 1, 1))
colnames(dt.ex1) <- c("a1", "a2", "d")
decision.table <- SF.asDecisionTable(dataset = dt.ex1, decision.attr = 3, indx.nominal = c(3))

## evaluate instances
res.1 <- IS.FRPS.FRST(decision.table, type.alpha = "FRPS.3")

## generate new decision table
new.decTable <- SF.applyDecTable(decision.table, res.1)
```

MV.conceptClosestFit   *Concept Closest Fit*

Description

It is used for handling missing values based on the concept closest fit.

Usage

```
MV.conceptClosestFit(decision.table)
```

Arguments

decision.table   a "DecisionTable" class representing a decision table. See SF.asDecisionTable.
                 Note: missing values are recognized as NA.

Details

This method is similar to the global closest fit method. The difference is that the original data set,
containing missing attribute values, is first split into smaller data sets, each smaller data set corre-
sponds to a concept from the original data set. More precisely, every smaller data set is constructed
from one of the original concepts, by restricting cases to the concept.

Value

A class "MissingValue". See MV.missingValueCompletion.

Author(s)

Lala Septem Riza

References

J. Grzymala-Busse and W. Grzymala-Busse, "Handling Missing Attribute Values," in Data Mining and Knowledge Discovery Handbook, O. Maimon and L. Rokach, Eds. New York : Springer, 2010, pp. 33-51

See Also

MV.missingValueCompletion

Examples

```
##############################################
## Example: Concept Closest Fit
##############################################
dt.ex1 <- data.frame(
    c(100.2, 102.6, NA, 99.6, 99.8, 96.4, 96.6, NA),
    c(NA, "yes", "no", "yes", NA, "yes", "no", "yes"),
    c("no", "yes", "no", "yes", "yes", "no", "yes", NA),
    c("yes", "yes", "no", "yes", "no", "no", "no", "yes"))
colnames(dt.ex1) <- c("Temp", "Headache", "Nausea", "Flu")
decision.table <- SF.asDecisionTable(dataset = dt.ex1, decision.attr = 4,
                                    indx.nominal = c(2:4))
indx = MV.conceptClosestFit(decision.table)
```

MV.deletionCases   *Missing value completion by deleting instances*

Description

It is used for handling missing values by deleting instances. It should be noted that the output of the function is val.NA which contains indices of missing values and their values (i.e., NA). Therefore, in order to generate a new decision table (dataset) the user need to execute SF.applyDecTable.

Usage

```
MV.deletionCases(decision.table)
```

Arguments

decision.table   a "DecisionTable" class representing a decision table. See SF.asDecisionTable.
                 Note: missing values are recognized as NA.

Value

A class "MissingValue". See MV.missingValueCompletion.

Author(s)

Lala Septem Riza

References

J. Grzymala-Busse and W. Grzymala-Busse, "Handling Missing Attribute Values," in Data Mining and Knowledge Discovery Handbook, O. Maimon and L. Rokach, Eds. New York : Springer, 2010, pp. 33-51

See Also

[MV.missingValueCompletion](#)

Examples

```
################################################
## Example : Deletion Cases
################################################
dt.ex1 <- data.frame(
    c("high", "very_high", NA, "high", "high", "normal", "normal", NA),
    c(NA, "yes", "no", "yes", NA, "yes", "no", "yes"),
    c("no", "yes", "no", "yes", "yes", "no", "yes", NA),
    c("yes", "yes", "no", "yes", "no", "no", "no", "yes"))
colnames(dt.ex1) <- c("Temp", "Headache", "Nausea", "Flu")
decision.table <- SF.asDecisionTable(dataset = dt.ex1, decision.attr = 4,
                                     indx.nominal = c(1:4))
indx = MV.deletionCases(decision.table)
```

MV.globalClosestFit   *Global Closest Fit*

Description

It is used for handling missing values based on the global closest fit.

Usage

```
MV.globalClosestFit(decision.table)
```

Arguments

decision.table a "DecisionTable" class representing a decision table. See [SF.asDecisionTable](#).
Note: missing values are recognized as NA.

Details

The global closes fit method is based on replacing a missing attribute value by the known value in another case that resembles as much as possible the case with the missing attribute value. In searching for the closest fit case we compare two vectors of attribute values, one vector corresponds to the case with a missing attribute value, the other vector is a candidate for the closest fit. The search is conducted for all cases, hence the name global closest fit. For each case a distance is computed, the case for which the distance is the smallest is the closest fitting case that is used to determine the missing attribute value.

Value

A class ″MissingValue″. See `MV.missingValueCompletion`.

Author(s)

Lala Septem Riza

References

J. Grzymala-Busse and W. Grzymala-Busse, "Handling Missing Attribute Values," in Data Mining and Knowledge Discovery Handbook, O. Maimon and L. Rokach, Eds. New York : Springer, 2010, pp. 33-51

See Also

`MV.missingValueCompletion`

Examples

```
#############################################
## Example: Global Closest Fit
#############################################
dt.ex1 <- data.frame(
     c(100.2, 102.6, NA, 99.6, 99.8, 96.4, 96.6, NA),
     c(NA, "yes", "no", "yes", NA, "yes", "no", "yes"),
     c("no", "yes", "no", "yes", "yes", "no", "yes", NA),
     c("yes", "yes", "no", "yes", "no", "no", "no", "yes"))
colnames(dt.ex1) <- c("Temp", "Headache", "Nausea", "Flu")
decision.table <- SF.asDecisionTable(dataset = dt.ex1, decision.attr = 4,
                                     indx.nominal = c(2:4))
indx = MV.globalClosestFit(decision.table)
```

MV.missingValueCompletion

Wrapper function of missing value completion

Description

It is a wrapper function for missing value completion.

Usage

```
MV.missingValueCompletion(decision.table, type.method = "deletionCases")
```

Arguments

decision.table   a ″DecisionTable″ class representing a decision table. See `SF.asDecisionTable`. Note: missing values are recognized as NA.

type.method   one of the following methods:

- ″deletionCases″: See `MV.deletionCases`.
- ″mostCommonValResConcept″: See `MV.mostCommonValResConcept`.
- ″mostCommonVal″: See `MV.mostCommonVal`.
- ″globalClosestFit″: See `MV.globalClosestFit`.
- ″conceptClosestFit″: See `MV.conceptClosestFit`.

Value

A class ″MissingValue″ which contains

- val.NA: a matrix containing indices of missing value (i.e., unknown values) positions and their values.
- type.method: a string showing the type of used method. In this case, it is ″deleteCases″.

Author(s)

Lala Septem Riza

References

J. Grzymala-Busse and W. Grzymala-Busse, "Handling Missing Attribute Values," in Data Mining and Knowledge Discovery Handbook, O. Maimon and L. Rokach, Eds. New York : Springer, 2010, pp. 33-51

See Also

`MV.deletionCases`, `MV.mostCommonValResConcept`, `MV.mostCommonVal`, `MV.globalClosestFit`, and `MV.conceptClosestFit`.

Examples

```
#############################################
## Example :
#############################################
dt.ex1 <- data.frame(
    c(100.2, 102.6, NA, 99.6, 99.8, 96.4, 96.6, NA),
    c(NA, "yes", "no", "yes", NA, "yes", "no", "yes"),
    c("no", "yes", "no", "yes", "yes", "no", "yes", NA),
    c("yes", "yes", "no", "yes", "no", "no", "no", "yes"))
colnames(dt.ex1) <- c("Temp", "Headache", "Nausea", "Flu")
decision.table <- SF.asDecisionTable(dataset = dt.ex1, decision.attr = 4,
                                     indx.nominal = c(2:4))
indx = MV.missingValueCompletion(decision.table, type.method = "deletionCases")

## generate new decision table
new.decTable <- SF.applyDecTable(decision.table, indx)
```

MV.mostCommonVal   *Replacing missing attribute values by the attribute mean or common values*

Description

It is used for handling missing values by replacing the attribute mean or common values. If an attributes containing missing values is continuous/real, the method uses mean of the attribute instead of the most common value. In order to generate a new decision table, we need to execute `SF.applyDecTable`.

Usage

```
MV.mostCommonVal(decision.table)
```

Arguments

decision.table a ″DecisionTable″ class representing a decision table. See `SF.asDecisionTable`. Note: missing values are recognized as NA.

Value

A class ″MissingValue″. See `MV.missingValueCompletion`.

Author(s)

Lala Septem Riza

References

J. Grzymala-Busse and W. Grzymala-Busse, "Handling Missing Attribute Values," in Data Mining and Knowledge Discovery Handbook, O. Maimon and L. Rokach, Eds. New York : Springer, 2010, pp. 33-51

See Also

`MV.missingValueCompletion`

Examples

```
##############################################
## Example: Replacing missing attribute values
##           by the attribute mean/common values
##############################################
dt.ex1 <- data.frame(
    c(100.2, 102.6, NA, 99.6, 99.8, 96.4, 96.6, NA),
    c(NA, "yes", "no", "yes", NA, "yes", "no", "yes"),
    c("no", "yes", "no", "yes", "yes", "no", "yes", NA),
    c("yes", "yes", "no", "yes", "no", "no", "no", "yes"))
```

```
colnames(dt.ex1) <- c("Temp", "Headache", "Nausea", "Flu")
decision.table <- SF.asDecisionTable(dataset = dt.ex1, decision.attr = 4,
                                     indx.nominal = c(2:4))
indx = MV.mostCommonVal(decision.table)
```

MV.mostCommonValResConcept

The most common value or mean of an attribute restricted to a concept

---

Description

It is used for handling missing values by assigning the most common value of an attribute restricted to a concept. If an attributes containing missing values is continuous/real, the method uses mean of the attribute instead of the most common value. In order to generate a new decision table, we need to execute `SF.applyDecTable`.

Usage

```
MV.mostCommonValResConcept(decision.table)
```

Arguments

decision.table   a `"DecisionTable"` class representing a decision table. See `SF.asDecisionTable`. Note: missing values are recognized as NA.

Value

A class `"MissingValue"`. See `MV.missingValueCompletion`.

Author(s)

Lala Septem Riza

References

J. Grzymala-Busse and W. Grzymala-Busse, "Handling Missing Attribute Values," in Data Mining and Knowledge Discovery Handbook, O. Maimon and L. Rokach, Eds. New York : Springer, 2010, pp. 33-51

See Also

`MV.missingValueCompletion`

Examples

```
#############################################
## Example: The most common value
#############################################
dt.ex1 <- data.frame(
    c(100.2, 102.6, NA, 99.6, 99.8, 96.4, 96.6, NA),
    c(NA, "yes", "no", "yes", NA, "yes", "no", "yes"),
    c("no", "yes", "no", "yes", "yes", "no", "yes", NA),
    c("yes", "yes", "no", "yes", "no", "no", "no", "yes"))
colnames(dt.ex1) <- c("Temp", "Headache", "Nausea", "Flu")
decision.table <- SF.asDecisionTable(dataset = dt.ex1, decision.attr = 4,
                                     indx.nominal = c(2:4))
indx = MV.mostCommonValResConcept(decision.table)
```

---

predict.RuleSetFRST   *The predicting function for rule induction methods based on FRST*

Description

It is a function used to obtain predicted values after obtaining rules by using rule induction methods. We have provided the functions [RI.GFRS.FRST](#) and [RI.hybridFS.FRST](#) to generate rules based on FRST.

Usage

```
## S3 method for class 'RuleSetFRST'
predict(object, newdata, ...)
```

Arguments

object	a "RuleSetFRST" class resulted by [RI.GFRS.FRST](#) and [RI.hybridFS.FRST.](#)
newdata	a "DecisionTable" class containing a data frame or matrix (m x n) of data for the prediction process, where m is the number of instances and n is the number of input attributes. It should be noted that this data must have colnames on each attributes.
...	the other parameters.

Value

The predicted values.

Author(s)

Lala Septem Riza

See Also

[RI.indiscernibilityBasedRules.RST](#), [RI.GFRS.FRST](#) and [RI.hybridFS.FRST](#)

Examples

```
###############################################
## Example: Classification Task
###############################################
data(RoughSetData)
decision.table <- RoughSetData$pima7.dt

## using RI.hybrid.FRST for generating rules
control <- list(type.aggregation = c("t.tnorm", "lukasiewicz"),
          type.relation = c("tolerance", "eq.1"), t.implicator = "lukasiewicz")
rules.hybrid <- RI.hybridFS.FRST(decision.table, control)

## in this case, we are using the same data set as the training data
res.1 <- predict(rules.hybrid, decision.table[, -ncol(decision.table)])

## using RI.GFRS.FRST for generating rules
control <- list(alpha.precision = 0.01, type.aggregation = c("t.tnorm", "lukasiewicz"),
              type.relation = c("tolerance", "eq.3"), t.implicator = "lukasiewicz")
rules.gfrs <- RI.GFRS.FRST(decision.table, control)

## in this case, we are using the same data set as the training data
res.2 <- predict(rules.gfrs, decision.table[, -ncol(decision.table)])

###############################################
## Example: Regression Task
###############################################
data(RoughSetData)
decision.table <- RoughSetData$housing7.dt

## using RI.hybrid.FRST for generating rules
control <- list(type.aggregation = c("t.tnorm", "lukasiewicz"),
          type.relation = c("tolerance", "eq.1"), t.implicator = "lukasiewicz")
rules <- RI.hybridFS.FRST(decision.table, control)

## in this case, we are using the same data set as the training data
res.1 <- predict(rules, decision.table[, -ncol(decision.table)])
```

---

predict.RuleSetRST   *Prediction of decision classes using rule-based classifiers.*

Description

The prediction method for objects inheriting from the RuleSetRST class.

Usage

```
## S3 method for class 'RuleSetRST'
predict(object, newdata, ...)
```

Arguments

object   an object inheriting from the "RuleSetRST" class. Such objects are typically
                  produced by implementations of rule induction methods, which derives from
                  the rough set theory (RST), such as RI.indiscernibilityBasedRules.RST,
                  RI.CN2Rules.RST, RI.LEM2Rules.RST or RI.AQRules.RST.

newdata   an object inheriting from the "DecisionTable" class, which represents the data
                  for which predictions are to be made. See SF.asDecisionTable. Columns in
                  newdata should correspond to columns of a data set used for the rule induction.

...   additional parameters for a rule voting strategy. It can be applied only to the
                  methods which classify new objects by voting. Currently, those methods in-
                  clude RI.LEM2Rules.RST and RI.AQRules.RST which accept a named param-
                  eter votingMethod. This parameter can be used to pass a custom function for
                  computing a weight of a voting rule. There are three such functions already
                  available in the package:

                  • X.ruleStrength is the default voting method. It is defined as a product
                    of a cardinality of a support of a rule and the length of this rule. See
                    X.ruleStrength.
                  • X.laplace corresponds to a voting weighted by the Laplace estimates of
                    rules' confidence. See X.laplace.
                  • X.rulesCounting corresponds to voting by counting the matching rules
                    for different decision classes. See X.rulesCounting.

                  A custom function passed using the votingMethod can get additional parame-
                  ters using the ... interface.

Value

A data.frame with a single column containing predictions for objects from newdata.

Author(s)

Andrzej Janusz

See Also

Rule induction methods implemented within RST include: RI.indiscernibilityBasedRules.RST,
RI.CN2Rules.RST, RI.LEM2Rules.RST and RI.AQRules.RST. For details on rule induction meth-
ods based on FRST see RI.GFRS.FRST and RI.hybridFS.FRST.

Examples

```
###############################################
## Example: Classification Task
###############################################
data(RoughSetData)
wine.data <- RoughSetData$wine.dt
set.seed(13)
wine.data <- wine.data[sample(nrow(wine.data)),]
```

```
## Split the data into a training set and a test set,
## 60% for training and 40% for testing:
idx <- round(0.6 * nrow(wine.data))
wine.tra <-SF.asDecisionTable(wine.data[1:idx,],
                               decision.attr = 14,
                               indx.nominal = 14)
wine.tst <- SF.asDecisionTable(wine.data[(idx+1):nrow(wine.data), -ncol(wine.data)])

true.classes <- wine.data[(idx+1):nrow(wine.data), ncol(wine.data)]

## discretization:
cut.values <- D.discretization.RST(wine.tra,
                                   type.method = "unsupervised.quantiles",
                                   nOfIntervals = 3)
data.tra <- SF.applyDecTable(wine.tra, cut.values)
data.tst <- SF.applyDecTable(wine.tst, cut.values)

## rule induction from the training set:
rules <- RI.LEM2Rules.RST(data.tra)

## predicitons for the test set:
pred.vals1 <- predict(rules, data.tst)
pred.vals2 <- predict(rules, data.tst,
                       votingMethod = X.laplace)
pred.vals3 <- predict(rules, data.tst,
                       votingMethod = X.rulesCounting)

## checking the accuracy of predictions:
mean(pred.vals1 == true.classes)
mean(pred.vals2 == true.classes)
mean(pred.vals3 == true.classes)
```

---

print.FeatureSubset   *The print method of FeatureSubset objects*

Description

This is a print method for FeatureSubset objects.

Usage

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

Arguments

x	an object inheriting from "FeatureSubset" class. See FS.reduct.computation.
...	parameters passes to other functions (currently omitted).

Value

Prints its argument and returns it invisibly.

Author(s)

Andrzej Janusz

Examples

```
#############################################################
## Example : Computation of a decision reduct
#############################################################
data(RoughSetData)
decision.table <- RoughSetData$hiring.dt

res.1 <- FS.reduct.computation(decision.table)
print(res.1)
```

---

print.RuleSetRST   *The print function for RST rule sets*

Description

A print method for RuleSetRST objects.

Usage

```
## S3 method for class 'RuleSetRST'
print(x, howMany = min(10, length(x)), ...)
```

Arguments

x	a "RuleSetRST" object. See RI.LEM2Rules.RST.
howMany	an integer giving the number of rules to be printed. The default is minimum from 10 and the total number of rules in the set.
...	the other parameters.

Value

prints its argument and returns it invisibly

Author(s)

Andrzej Janusz

Examples

```
###########################################################
## Example : Printing of a decision rule set problem
###########################################################
data(RoughSetData)
hiring.data <- RoughSetData$hiring.dt

rules <- RI.LEM2Rules.RST(hiring.data)

rules             # all rules are printed
print(rules, 2)   # only the first two rules are printed

# printing a subset of rules
rules[2:3]
```

RI.AQRules.RST   *Rule induction using the AQ algorithm*

Description

A version of the AQ algorithm which was originally proposed by R.S. Michalski. This implamentation is based on a concept of a local (object-relative) decision reduct from RST.

Usage

```
RI.AQRules.RST(decision.table, confidence = 1, timesCovered = 1)
```

Arguments

decision.table an object inheriting from the "DecisionTable" class, which represents a decision system. See `SF.asDecisionTable`.

confidence   a numeric value giving the minimal confidence of computed rules.

timesCovered   a positive integer. The algorithm will try to find a coverage of training examples with rules, such that each example is covered by at least `timesCovered` rules and no rule can be removed from the coverage without breaking this property. This is not always possible - there is a chance that some rules are duplicated if the value of `timesCovered` is larger than 1.

Value

An object of a class "RuleSetRST". For details see `RI.indiscernibilityBasedRules.RST`.

Author(s)

Andrzej Janusz

References

R.S. Michalski, K. Kaufman, J. Wnek: "The AQ Family of Learning Programs: A Review of Recent Developments and an Exemplary Application", Reports of Machine Learning and Inference Laboratory, George Mason University (1991)

See Also

predict.RuleSetFRST, RI.indiscernibilityBasedRules.RST, RI.CN2Rules.RST, RI.LEM2Rules.RST.

Examples

```
#############################################################
## Example
#############################################################
data(RoughSetData)
wine.data <- RoughSetData$wine.dt
set.seed(13)
wine.data <- wine.data[sample(nrow(wine.data)),]

## Split the data into a training set and a test set,
## 60% for training and 40% for testing:
idx <- round(0.6 * nrow(wine.data))
wine.tra <-SF.asDecisionTable(wine.data[1:idx,],
                              decision.attr = 14,
                              indx.nominal = 14)
wine.tst <- SF.asDecisionTable(wine.data[(idx+1):nrow(wine.data), -ncol(wine.data)])

true.classes <- wine.data[(idx+1):nrow(wine.data), ncol(wine.data)]

## discretization:
cut.values <- D.discretization.RST(wine.tra,
                                    type.method = "unsupervised.quantiles",
                                    nOfIntervals = 3)
data.tra <- SF.applyDecTable(wine.tra, cut.values)
data.tst <- SF.applyDecTable(wine.tst, cut.values)

## rule induction from the training set:
rules <- RI.AQRules.RST(data.tra, confidence = 0.9, timesCovered = 3)
rules

## predicitons for the test set:
pred.vals <- predict(rules, data.tst)

## checking the accuracy of predictions:
mean(pred.vals == true.classes)
```

RI.CN2Rules.RST   *Rule induction using a version of CN2 algorithm*

Description

An implementation of verions of the famous CN2 algorithm for induction of decision rules, proposed by P.E. Clark and T. Niblett.

Usage

```
RI.CN2Rules.RST(decision.table, K = 3)
```

Arguments

decision.table	an object inheriting from the ″DecisionTable″ class, which represents a decision system. See `SF.asDecisionTable`.
K	a positive integer that controls a complexity of the algorithm. In each iteration K best rule predicates are extended by all possible descriptors.

Value

An object of a class ″RuleSetRST″. For details see `RI.indiscernibilityBasedRules.RST`.

Author(s)

Andrzej Janusz

References

P.E. Clark and T. Niblett, "The CN2 Induction algorithm", Machine Learning, 3, p. 261 - 284 (1986).

See Also

`predict.RuleSetFRST`, `RI.indiscernibilityBasedRules.RST`, `RI.LEM2Rules.RST`, `RI.AQRules.RST`.

Examples

```
#############################################################
## Example
#############################################################
data(RoughSetData)
wine.data <- RoughSetData$wine.dt
set.seed(13)
wine.data <- wine.data[sample(nrow(wine.data)),]

## Split the data into a training set and a test set,
## 60% for training and 40% for testing:
idx <- round(0.6 * nrow(wine.data))
```

```
wine.tra <-SF.asDecisionTable(wine.data[1:idx,],
                             decision.attr = 14,
                             indx.nominal = 14)
wine.tst <- SF.asDecisionTable(wine.data[(idx+1):nrow(wine.data), -ncol(wine.data)])

true.classes <- wine.data[(idx+1):nrow(wine.data), ncol(wine.data)]

## discretization:
cut.values <- D.discretization.RST(wine.tra,
                                   type.method = "unsupervised.quantiles",
                                   nOfIntervals = 3)
data.tra <- SF.applyDecTable(wine.tra, cut.values)
data.tst <- SF.applyDecTable(wine.tst, cut.values)

## rule induction from the training set:
rules <- RI.CN2Rules.RST(data.tra, K = 5)
rules

## predicitons for the test set:
pred.vals <- predict(rules, data.tst)

## checking the accuracy of predictions:
mean(pred.vals == true.classes)
```

RI.GFRS.FRST   *Generalized fuzzy rough set rule induction based on FRST*

Description

It is a function generating rules in classification tasks using the fuzzy variable precision rough sets (FVPRS) approach (see BC.LU.approximation.FRST).

Usage

```
RI.GFRS.FRST(decision.table, control = list())
```

Arguments

decision.table   a "DecisionTable" class representing the decision table. See SF.asDecisionTable.

control   a list of other parameters which consist of

- alpha.precision: a numeric value representing variable precision of FVPRS. The default value is 0.05. See BC.LU.approximation.FRST.
- type.aggregation: a list of a type of aggregations. The default value is type.aggregation = c("t.tnorm", "lukasiewicz"). See BC.IND.relation.FRST.
- type.relation: a type of indiscernibility relation. The default value is type.relation = c("tolerance", "eq.1"). See BC.IND.relation.FRST.
- t.implicator: a type of implication function. The default value is "lukasiewicz". See BC.LU.approximation.FRST.

Details

The method proposed by (Zhao, 2010) consists of three steps as follows. First, it builds a general lower approximation that is able to deal with misclassification and perturbation. In this case, the fuzzy variable precision rough sets (FVPRS) is used to calculate the lower approximation (see `BC.LU.approximation.FRST`). Secondly, a discernibility matrix considering a consistence degree is constructed for obtaining rules. The details about the matrix can be seen in `BC.discernibility.mat.FRST`. Then, we calculate attribute value reduction of every object and perform near-minimal rule set. The final step is to construct rules considering the consistence degree of associated objects.

It should be noted that this function only allows classification problems. After obtaining the rules, predicting can be done by calling `predict` or `predict.RuleSetFRST`. Additionally, to get better representation we can execute `summary`.

Value

A class `"RuleSetFRST"` which has following components:

- `rules`: It is a list containing two elements which are `rules` and `threshold`. The `rules` represent knowledge in data set that can be expressed as an IF-THEN form. For example, we got the rule as follows: `90 8 2` and its colnames: `pres`, `preg`, and `class`. It refers to the following rule: `IF pres is about 90 and preg is about 8 THEN class is 2`. In other words, while the last column represents the consequent part, the rest expresses the antecedent part. The second part of this object is `threshold` representing a value used to predict new data. In order to change IF-THEN form, we can use `summary`.

- `type.model`: It is the type of the theory whether `"FRST"` or `"RST"`. In this case, it is FRST.

- `type.method`: It is the considered method. In this case, it is RI.GFRS.FRST.

- `type.task`: It is the type of task. In this case, it is `"classification"`.

- `t.similariy`: It is the type of similarity equation. See `BC.IND.relation.FRST`.

- `t.tnorm`: It is the type of triangular operator. See `BC.IND.relation.FRST`.

- `variance.data`: It represents the variance of the data set. It has NA values when the associated attributes are nominal values.

- `range.data`: It represents the range of the data set. It has NA values when the associated attributes are nominal values.

- `antecedent.attr`: It is a list of attribute names involved in the antecedent part.

- `consequent.attr`: It is the attribute in the consequent part.

- `nominal.att`: It is a list of boolean that represent whether a attribute is nominal or not.

Author(s)

Lala Septem Riza

References

S. Y. Zhao, E. C. C. Tsang, D. G. Chen, and X. Z. Wang, "Building a Rule-based Classifier – A Fuzzy-rough Set Approach", IEEE Trans. on Knowledge and Data Engineering, vol. 22, no. 5, p. 624 - 638 (2010).

See Also

`RI.indiscernibilityBasedRules.RST`, `predict.RuleSetFRST`, and `RI.hybridFS.FRST`.

Examples

```
##############################################################
## Example
##############################################################
data(RoughSetData)
decision.table <- RoughSetData$pima7.dt

control <- list(alpha.precision = 0.01, type.aggregation = c("t.tnorm", "lukasiewicz"),
                type.relation = c("tolerance", "eq.3"), t.implicator = "lukasiewicz")
rules <- RI.GFRS.FRST(decision.table, control)
```

RI.hybridFS.FRST   *Hybrid fuzzy-rough rule and induction and feature selection*

Description

It is a function for generating rules based on hybrid fuzzy-rough rule induction and feature selection. It allows for classification and regression tasks.

Usage

```
RI.hybridFS.FRST(decision.table, control = list())
```

Arguments

decision.table a "DecisionTable" class representing the decision table. See `SF.asDecisionTable`.

control   a list of other parameters which consist of

- type.aggregation a list representing the type of aggregation. The default value is type.aggregation = c("t.tnorm", "lukasiewicz").
  See `BC.IND.relation.FRST`.
- type.relation the type of indiscernibility relation. The default value is type.relation = c("tolerance", "eq.3"). See `BC.IND.relation.FRST`.
- t.implicator the type of implication function. The default value is "lukasiewicz". See BC.LU.approximation.FRST.

Details

It was proposed by (Jensen et al, 2009) attempting to combine rule induction and feature selection at the same time. Basically this algorithm inserts some steps to generate rules into the fuzzy QuickReduct algorithm (see `FS.quickreduct.FRST`. Furthermore, by introducing the degree of coverage, this algorithm selects proper rules.

This function allows not only for classification but also for regression problems. After obtaining the rules, predicting can be done by calling `predict` or `predict.RuleSetFRST`. Additionally, to get better representation we can execute `summary`.

Value

A class *"RuleSetFRST"* which has similar components as `RI.GFRS.FRST` but in this case the `threshold` component is not included.

Author(s)

Lala Septem Riza

References

R. Jensen, C. Cornelis, and Q. Shen, "Hybrid Fuzzy-rough Rule Induction and Feature Selection", in: IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), p. 1151 - 1156 (2009).

See Also

`RI.indiscernibilityBasedRules.RST`, `predict.RuleSetFRST`, and `RI.GFRS.FRST`.

Examples

```
#############################################################
## Example 1: Regression problem
#############################################################
data(RoughSetData)
decision.table <- RoughSetData$housing7.dt

control <- list(type.aggregation = c("t.tnorm", "lukasiewicz"), type.relation =
                c("tolerance", "eq.3"), t.implicator = "lukasiewicz")
res.1 <- RI.hybridFS.FRST(decision.table, control)

#############################################################
## Example 2: Classification problem
#############################################################
data(RoughSetData)
decision.table <- RoughSetData$pima7.dt

control <- list(type.aggregation = c("t.tnorm", "lukasiewicz"), type.relation =
                c("tolerance", "eq.3"), t.implicator = "lukasiewicz")
res.2 <- RI.hybridFS.FRST(decision.table, control)
```

RI.indiscernibilityBasedRules.RST

Rule induction from indiscernibility classes.

Description

Rule induction from indiscernibility classes.

Usage

```
RI.indiscernibilityBasedRules.RST(decision.table, feature.set)
```

Arguments

decision.table   an object inheriting from the ″DecisionTable″ class, which represents a decision system. See `SF.asDecisionTable`.

feature.set   an object inheriting from the ″FeatureSubset″ class which is a typical output of feature selection methods based on RST e.g. `FS.greedy.heuristic.reduct.RST`. See also `FS.reduct.computation`, `FS.feature.subset.computation` and `FS.all.reducts.computa...` based on RST.

Details

This function generates "if-then" decision rules from indiscernibility classes defined by a given subset of conditional attributes.

After obtaining the rules, decision classes of new objects can be predicted using the `predict` method or by a direct call to `predict.RuleSetRST`.

Value

An object of a class ″RuleSetRST″, which is a list with additional attributes:

- uniqueCls: a vector of possible decision classes,
- supportDenominator: an integer giving the number of objects in the data,
- clsProbs: a vector giving the a priori probability of the decision classes,
- majorityCls: a class label representing the majority class in the data,
- method: the type a rule induction method used for computations,
- dec.attr: a name of the decision attribute in the data,
- colnames: a vector of conditional attribute names.

Each rule is a list with the following elements:

- idx: a vector of indexes of attribute that are used in antecedent of a rule,
- values: a vector of values of attributes indicated by idx,
- consequent: a value of the consequent of a rule,
- support: a vactor of integers indicating objects from the data, which support a given rule,
- laplace: ia numeric value representing the Laplace estimate of the rule's confidence.

Author(s)

Andrzej Janusz

See Also

`predict.RuleSetRST`, `RI.CN2Rules.RST`, `RI.LEM2Rules.RST`, `RI.AQRules.RST`.

Examples

```
#############################################################
## Example
#############################################################
data(RoughSetData)
hiring.data <- RoughSetData$hiring.dt

## determine feature subset/reduct
reduct <- FS.reduct.computation(hiring.data,
                                method = "permutation.heuristic",
                                permutation = FALSE)

rules <- RI.indiscernibilityBasedRules.RST(hiring.data, reduct)
rules
```

RI.laplace   *Quality indicators of RST decision rules*

Description

Functions for extracting quality indices of rules.

Usage

```
RI.laplace(rules, ...)
RI.support(rules, ...)
RI.confidence(rules, ...)
RI.lift(rules, ...)
```

Arguments

rules	a "RuleSetRST" object containing a set of decision rules. See `RI.LEM2Rules.RST`.
...	the other parameters (currently omitted).

Value

A numeric vector with values of the corresponding quality measure.

Author(s)

Andrzej Janusz

Examples

```
############################################################
## Example : Filtering a set of decision rules
############################################################
data(RoughSetData)
hiring.data <- RoughSetData$hiring.dt

rules <- RI.LEM2Rules.RST(hiring.data)

rules

# a vector of rules' Laplace estimate of the confidence:
RI.laplace(rules)
# a vector of rules' confidence values:
RI.confidence(rules)

# subsetting a set of rules:
rules[RI.support(rules) > 0.2]
rules[RI.lift(rules) < 1.5]
```

RI.LEM2Rules.RST   *Rule induction using the LEM2 algorithm*

Description

An implementation of LEM2 (Learning from Examples Module, version 2) for induction of decision rules, originally proposed by J.W. Grzymala-Busse.

Usage

```
RI.LEM2Rules.RST(decision.table)
```

Arguments

decision.table   an object inheriting from the ″DecisionTable″ class, which represents a decision system. See SF.asDecisionTable.

Value

An object of a class ″RuleSetRST″. For details see RI.indiscernibilityBasedRules.RST.

Author(s)

Andrzej Janusz

References

J.W. Grzymala-Busse, "A New Version of the Rule Induction System LERS", Fundamenta Informaticae, 31, p. 27 - 39 (1997).

See Also

predict.RuleSetFRST, RI.indiscernibilityBasedRules.RST, RI.CN2Rules.RST, RI.AQRules.RST.

Examples

```
#############################################################
## Example
##############################################################
data(RoughSetData)
wine.data <- RoughSetData$wine.dt
set.seed(13)
wine.data <- wine.data[sample(nrow(wine.data)),]

## Split the data into a training set and a test set,
## 60% for training and 40% for testing:
idx <- round(0.6 * nrow(wine.data))
wine.tra <-SF.asDecisionTable(wine.data[1:idx,],
                              decision.attr = 14,
                              indx.nominal = 14)
wine.tst <- SF.asDecisionTable(wine.data[(idx+1):nrow(wine.data), -ncol(wine.data)])

true.classes <- wine.data[(idx+1):nrow(wine.data), ncol(wine.data)]

## discretization:
cut.values <- D.discretization.RST(wine.tra,
                                   type.method = "local.discernibility",
                                   maxNOfCuts = 1)
data.tra <- SF.applyDecTable(wine.tra, cut.values)
data.tst <- SF.applyDecTable(wine.tst, cut.values)

## rule induction from the training set:
rules <- RI.LEM2Rules.RST(data.tra)
rules

## predicitons for the test set:
pred.vals <- predict(rules, data.tst)

## checking the accuracy of predictions:
mean(pred.vals == true.classes)
```

RoughSetData   *Data set of the package*

Description

Several datasets have been embedded in this package to be used as decision table of examples. They can be accessed by typing data(RoughSetData). The following is a description of each datasets.

Details

The hiring dataset

It is simple data taken from (Komorowski et al, 1999) where all the attributes have nominal values. It consists of eight objects with four conditional attributes and one decision attribute. The detailed description of each attribute is as follows:

- Diploma: it has the following values: "MBA", "MSc", "MCE".
- Exprience: it has the following values: "High", "Low", "Medium".
- French: it has the following values: "Yes", "No".
- Reference: it has the following values: "Excellent", "Good", "Neutral".
- Decision: it is a decision attribute that contains the following values: "Accept", "Reject".

The housing dataset

This data was taken from the Boston housing dataset located at the UCI Machine Learning repository, available at http://www.ics.uci.edu. It was first created by (Harrison and Rubinfeld, 1978). It contains 506 objects with 13 conditional attributes and one decision attribute. Furthermore, it should be noted that the housing dataset is a regression dataset which means that the decision attribute has continuous values. The conditional attributes contain both continuous and nominal attributes. The following is a description of each attribute:

- CRIM: it is a continuous attribute that expresses per capita crime rate by town. It has values in: [0.0062, 88.9762].
- ZN: it is a continuous attribute that represents the proportion of residential land zoned for lots over 25,000 sq.ft. It has values in: [0, 100].
- INDUS: it is a continuous attribute that shows the proportion of non-retail business acres per town. It has values in: [0.46, 27.74].
- CHAS: it is a nominal attribute that represents Charles River dummy variable. It has two values which are 1 if tract bounds river and 0 otherwise.
- NOX: it is a continuous attribute that shows the nitric oxides concentration (parts per 10 million). It has values in: [0.385, 0.871].
- RM: it is a continuous attribute that explains the average number of rooms per dwelling. It has values in: [3.561, 8.78].
- AGE: it is a continuous attribute that expresses proportion of owner-occupied units built prior to 1940. It has values in: [2.9, 100].
- DIS: it is a continuous attribute that shows weighted distances to five Boston employment centres. It has values in: [1.1296, 12.1265].
- RAD: it is a nominal attribute that shows the index of accessibility to radial highways. it has the integer value from 1 to 24.
- TAX: it is a continuous attribute that shows the full-value property-tax rate per \$10,000. It has values in: [187, 711].
- PTRATIO: it is a continuous attribute that shows the pupil-teacher ratio by town. It has values in: [12.6, 22].
- B: it is a continuous attribute that can be expressed by $1000(Bk - 0.63)^2$ where Bk is the proportion of blacks by town. It has values in: [0.32, 396.9].

- LSTAT: it is a continuous attribute that illustrates the percentage of lower status of the population. It has values in: [1.73, 37.97].
- MEDV: it is a continuous attribute that shows the median value of owner-occupied homes in $1000's. It has values in: [5, 50].

The wine dataset

This dataset is a classification dataset introduced first by (Forina, et al) which is commonly used as benchmark for simulation in the machine learning area. Additionally, it is available at the KEEL dataset repository (Alcala-Fdez, 2009), available at http://www.keel.es/. It consists of 178 instances with 13 conditional attributes and one decision attribute where all conditional attributes have continuous values. The description of each attribute is as follows:

- alcohol: it has a range in: [11, 14.9].
- malid_acid: it has a range in: [0.7, 5.8].
- ash: it has a range in: [1.3, 3.3].
- alcalinity_of_ash: it has a range in: [10.6, 30.0].
- magnesium: it has a range in: [70, 162].
- total_phenols: it has a range in: [0.9, 3.9].
- flavanoids: it has a range in: [0.3 5.1].
- nonflavanoid_phenols: it has a range in: [0.4 3.6].
- proanthocyanins: it has a range in: [0.4 3.6].
- color_intensity: it has a range in: [1.2 13.0].
- hue: it has a range in: [0.4 1.8].
- od: it has a range in: [1.2 4.0].
- proline: it has a range in: [278 1680].
- class: it is nominal decision attribute that has values: 1, 2, 3.

The pima dataset

It was taken from the pima Indians diabetes dataset which is available at the KEEL dataset repository (Alcala-Fdez, 2009), available at http://www.keel.es/. It was first created by National Institute of Diabetes and Digestive and Kidney Diseases. It contains 768 objects with 8 continuous conditional attributes. The description of each attribute is as follows:

- preg: it represents number of times pregnant and has values in: [1, 17].
- plas: it represents plasma glucose concentration a 2 hours in an oral glucose tolerance test and has values in: [0.0, 199.0].
- pres: it represents diastolic blood pressure (mm Hg) and has values in: [0.0, 122.0].
- skin: it represents triceps skin fold thickness (mm) and has values in: [0.0, 99.0].
- insu: it represents 2-hour serum insulin (mu U/ml) and has values in: [0.0, 846.0].
- mass: it represents body mass index (weight in kg/(height in m)^2) and has values in: [0.0, 67.1].
- pedi: it represents diabetes pedigree function and has values in: [0.078, 2.42].
- age: it represents age (years) and has values in: [21.0, 81.0].
- class: it is a decision attribute and has values in: [1, 2].

References

M. Forina, E. Leardi, C. Armanino, and S. Lanteri, "PARVUS - An Extendible Package for Data Exploration, Classification and Correlation", Journal of Chemonetrics, vol. 4, no. 2, p. 191 - 193 (1988).

D. Harrison, and D. L. Rubinfeld, "Hedonic Prices and the Demand for Clean Air", J. Environ. Economics & Management, vol.5, 81-102 (1978).

J. Alcala-Fdez, L. Sanchez, S. Garcia, M. J. del Jesus, S. Ventura, J. M. Garrell, J. Otero, C. Romero, J. Bacardit, V. M. Rivas, J. C. Fernandez, and F. Herrera, "KEEL: A Software Tool to Assess Evolutionary Algorithms to Data Mining Problems", Soft Computing vol. 13, no. 3, p. 307 - 318 (2009).

J. Komorowski, Z. Pawlak, L. Polwski, and A. Skowron, "Rough Sets: A Tutorial", In S. K. Pal and A. Skowron, editors, Rough Fuzzy Hybridization, A New Trend in Decision Making, pp. 3 - 98, Singopore, Springer (1999).

SF.applyDecTable   *Apply for obtaining a new decision table*

Description

It is used to apply a particular object/model for obtaining a new decision table. In other words, in order to use the function, the models, which are objects of missing value completion, feature selection, instance selection, or discretization, have been calculated previously .

Usage

```
SF.applyDecTable(decision.table, object, control = list())
```

Arguments

decision.table   a "DecisionTable" class representing a decision table. See SF.asDecisionTable.

object   a class resulting from feature selection (e.g., FS.reduct.computation), discretization (e.g., D.discretization.RST), instance selection functions (e.g., IS.FRIS.FRST), and missing value completion (e.g., MV.missingValueCompletion).

control   a list of other parameters which are indx.reduct representing an index of the chosen decision reduct. It is only considered when we calculate all reducts using FS.all.reducts.computation. The default value is that the first reduct will be chosen.

Value

A new decision table. Especially for the new decision table resulting from discretization, we obtain a different representation. Values are expressed in intervals instead of labels. For example, $a_1 = [-Inf, 1.35]$ refers to the value a_1 has a value in that range.

Author(s)

Lala Septem Riza and Andrzej Janusz

Examples

```
###############################################################
## Example 1: The feature selection in RST
## using quickreduct
###############################################################
data(RoughSetData)
decision.table <- RoughSetData$hiring.dt

## generate reducts
red.1 <- FS.quickreduct.RST(decision.table)

new.decTable <- SF.applyDecTable(decision.table, red.1)

###############################################################
## Example 2: The feature selection in FRST
## using fuzzy.QR (fuzzy quickreduct)
###############################################################
data(RoughSetData)
decision.table <- RoughSetData$hiring.dt

## fuzzy quickreduct using fuzzy lower approximation
control <- list(decision.attr = c(5), t.implicator = "lukasiewicz",
                type.relation = c("tolerance", "eq.1"), type.aggregation =
                c("t.tnorm", "lukasiewicz"))
red.2 <- FS.quickreduct.FRST(decision.table, type.method = "fuzzy.dependency",
                             type.QR = "fuzzy.QR", control = control)

## generate new decision table
new.decTable <- SF.applyDecTable(decision.table, red.2)

####################################################
## Example 3: The Instance selection by IS.FRPS and
## generate new decision table
####################################################
dt.ex1 <- data.frame(c(0.5, 0.2, 0.3, 0.7, 0.2, 0.2),
                     c(0.1, 0.4, 0.2, 0.8, 0.4, 0.4), c(0, 0, 0, 1, 1, 1))
colnames(dt.ex1) <- c("a1", "a2", "d")
decision.table <- SF.asDecisionTable(dataset = dt.ex1, decision.attr = 3)

## evaluate and select instances
res.1 <- IS.FRPS.FRST(decision.table, type.alpha = "FRPS.3")

## generate new decision table
new.decTable <- SF.applyDecTable(decision.table, res.1)

###############################################################
## Example 4: Discretization by determining cut values and
## then generate new decision table
```

```
#####################################################################
dt.ex2 <- data.frame(c(1, 1.2, 1.3, 1.4, 1.4, 1.6, 1.3), c(2, 0.5, 3, 1, 2, 3, 1),
                            c(1, 0, 0, 1, 0, 1, 1))
colnames(dt.ex2) <- c("a", "b", "d")
decision.table <- SF.asDecisionTable(dataset = dt.ex2, decision.attr = 3,
                    indx.nominal = 3)

## get cut values using the local strategy algorithm
cut.values <- D.discretization.RST(decision.table, type.method = "global.discernibility")

## generate new decision table
new.decTable <- SF.applyDecTable(decision.table, cut.values)

#####################################################################
## Example 5: Missing value completion
#####################################################################
dt.ex1 <- data.frame(
     c(100.2, 102.6, NA, 99.6, 99.8, 96.4, 96.6, NA),
     c(NA, "yes", "no", "yes", NA, "yes", "no", "yes"),
     c("no", "yes", "no", "yes", "yes", "no", "yes", NA),
     c("yes", "yes", "no", "yes", "no", "no", "no", "yes"))
colnames(dt.ex1) <- c("Temp", "Headache", "Nausea", "Flu")
decision.table <- SF.asDecisionTable(dataset = dt.ex1, decision.attr = 4,
                                     indx.nominal = c(2:4))

## missing value completion
val.NA = MV.missingValueCompletion(decision.table, type.method = "globalClosestFit")

## generate new decision table
new.decTable <- SF.applyDecTable(decision.table, val.NA)
new.decTable
```

SF.asDecisionTable   *Converting a data.frame into a* DecisionTable *object*

Description

This function converts data.frames into DecisionTable objects. This is a standard data representation in the RoughSets package.

Usage

```
SF.asDecisionTable(dataset, decision.attr = NULL, indx.nominal = NULL)
```

Arguments

dataset   data.frame that contains objects/instances and attributes/features in its rows and
                 columns, respectively. See in Section Details.

decision.attr an integer value representing the index position of the decision attribute. If this parameter is ignored, then the function will treat the data as an information system or newdata/test data. In other words, it is necessary to define the index of the decision attribute in order to construct a decision table (e.g. a training data set).

indx.nominal a logical vector indicating nominal attributes in the data. If this parameter is not given, then the function will use a heuristic to guess which of the attributes are nominal. The following rules will be applied used:

- an attribute contains character values or factors: it will be recognized as a nominal attribute.
- an attribute contains integer or numeric values: it will be recognized as a numeric attribute.
- indx.nominal: the indicated attributes will be considered as nominal.

Details

An object of the `"DecisionTable"` class adds a few attributes to a standard data.frame:

- `desc.attrs`: a list containing the names of attributes and their range/possible symbolic values. There are two kinds of representation in this parameters which depend on whether the attributes are nominal or numeric, for example:
  - nominal attribute: `a = c(1,2,3)` means that the attribute a has values 1, 2, and 3.
  - numeric attribute: `a = c(10, 20)` means that the attribute a has values between 10 and 20.
- `nominal.attrs`: a logical vector whose length equals the number of columns in the data. In this vector TRUE values indicate that the corresponding attribute is a nominal. For example: `nominal.attrs = c(FALSE, TRUE, FALSE)` means that the first and third attributes are numeric and the second one is nominal.
- `decision.attr`: a numeric value representing the index of the decision attribute. It is necessary to define the index of the decision attribute in order to construct a proper decision system. If the value of `decision.attr` is NULL, the constructed object will correspond to an information system. It is strongly recommended to place the decision attribute as the last data column.

`"DecisionTable"` objects allow to use all methods of standard data.frame objects. The function `SF.read.DecisionTable` can be used to import data from a file and then construct DecisionTable object.

Value

An object of the `"DecisionTable"` class.

Author(s)

Andrzej Janusz

See Also

`SF.read.DecisionTable`, `SF.applyDecTable`.

Examples

```
#################################################################
## Example : converting from datasets in data.frame
##           into decision table
#################################################################
## Let use iris which is available in R be dataset
decision.table <- SF.asDecisionTable(dataset = iris, decision.attr = 5,
                   indx.nominal = 5)
```

SF.asFeatureSubset   *Converting custom attribute name sets into a FeatureSubset object*

Description

The function can be used to change a custom set of attribute names from a decision table into an object of the FeatureSubset class. It can be useful for converting results of discernibility matrix-based attribute selection methods (i.e. functions FS.all.reducts.computation and FS.one.reduct.computation).

Usage

```
SF.asFeatureSubset(colNames, decisionTable = NULL,
  attributeNames = NULL, type.method = "custom subset",
  model = "custom")
```

Arguments

colNames	a character vector containing names of attributes from a decision table
decisionTable	a decision table which contains attributes from colNames, can be NULL and in that case a non-NULL value of attributeNames must be given
attributeNames	a character vector of names of decision table's attributes, can be NULL and in that case a non-NULL value of decisionTable must be given
type.method	an indicator of the method used for selecting the attributes
model	an indicator of the model used for selecting the attributes

Value

an object of a class FeatureSubset

Author(s)

Andrzej Janusz

Examples

```
##############################################################
## Example 1:
##############################################################
data(RoughSetData)
wine.data <- RoughSetData$wine.dt
dim(wine.data)

## selection of an arbitrary attribute subset
attrNames = colnames(wine.data)[1:3]
attrNames
class(attrNames)

## convertion into a FeatureSubset object
reduct <- SF.asFeatureSubset(attrNames, wine.data,
                              type.method = "greedy reduct from a discernibility matrix")

class(reduct)
reduct
```

SF.read.DecisionTable   *Reading tabular data from files.*

Description

This function can be used to import data sets from files and then construct a DecisionTable object. It uses read.table function from base R.

Usage

```
SF.read.DecisionTable(filename, decision.attr = NULL,
  indx.nominal = NULL, ...)
```

Arguments

filename	a path with a file name.
decision.attr	an integer indicating an index of the decision attribute. See SF.asDecisionTable.
indx.nominal	an integer vector with indices of attributes which should be considered as nominal. See SF.asDecisionTable.
...	additional parameters which are passed to the read.table function. See read.table.

Details

The data should be in a tabular format containing rows and columns, where every row represents an object/instance, while columns represent attributes of the objects.

Value

An object of the ″DecisionTable″ class. See `SF.asDecisionTable`.

Author(s)

Andrzej Janusz

Examples

```
#############################################################
## Example 1: data set saved in a file
#############################################################
## Let us assume we have the following data which has been already saved to the file ″tes.dat″
data <- data.frame(c(0.12, 0.23, 0.24), c(1,3,2), c(10, 12, 18), c("a", "a", "b"), c(1, 1, 2))
## Not run: write.table(data, file = ″tes.dat″, row.names = FALSE, col.names = FALSE,
                        fileEncoding =″″)
## End(Not run)

## Then we would generate decision table from tes.dat file.
## in this case, we want to define that second and third attributes are nominal and continuous,
## respectively.
## Not run: decision.table <- SF.read.DecisionTable(filename = ″tes.dat″, decision.attr = 5,
              indx.nominal = c(2, 5), sep= ″ ″, col.names = c(″v1″, ″v2″, ″v3″, ″v4″, ″o1″))
## End(Not run)
```

---

summary.IndiscernibilityRelation

The summary function for an indiscernibility relation

Description

This function enables the output of a summary of the indiscernibility relation functions.

Usage

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

Arguments

object   a ″IndiscernibilityRelation″ object. See `BC.IND.relation.FRST`
                and `BC.IND.relation.RST`.

...   the other parameters.

Value

a description that contains the following information. For FRST model:

Author(s)

Lala Septem Riza

Examples

```
#############################################################
## Example 1: Dataset containing nominal values for
## all attributes.
#############################################################
## Decision table is represented as data frame
dt.ex1 <- data.frame(c(1,0,2,1,1,2,2,0), c(0, 1,0, 1,0,2,1,1),
                        c(2,1,0,0,2,0,1,1), c(2,1,1,2,0,1,1,0), c(0,2,1,2,1,1,2,1))
colnames(dt.ex1) <- c("aa", "bb", "cc", "dd", "ee")
decision.table <- SF.asDecisionTable(dataset = dt.ex1, decision.attr = 5)

## In this case, we only consider the second and third attributes.
attributes <- c(2, 3)

#### calculate fuzzy indiscernibility relation ####
## in this case, we are using "crisp" as a type of relation and type of aggregation
control.ind <- list(type.relation = c("crisp"), type.aggregation = c("crisp"))
IND <- BC.IND.relation.FRST(decision.table, attributes = attributes, control = control.ind)

summary(IND)
```

---

summary.LowerUpperApproximation

The summary function of lower and upper approximations based on RST and FRST

Description

This function enables the output of a summary of the lower and upper approximations.

Usage

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

Arguments

object	a "LowerUpperApproximation" object. See `BC.LU.approximation.FRST` and `BC.LU.approximation.RST`.
...	the other parameters.

Author(s)

Lala Septem Riza

Examples

```
#########################################
## Example: Using simple data set
#########################################
dt.ex1 <- data.frame(c(1,0,2,1,1,2,2,0), c(0, 1,0, 1,0,2,1,1),
                       c(2,1,0,0,2,0,1,1), c(2,1,1,2,0,1,1,0), c(0,2,1,2,1,1,2,1))
colnames(dt.ex1) <- c("aa", "bb", "cc", "dd", "ee")
decision.table <- SF.asDecisionTable(dataset = dt.ex1, decision.attr = 5,
                                     indx.nominal = c(1:5))

P <- c(2,3)

####### Compute indiscernibility relation #######
IND <- BC.IND.relation.RST(decision.table, feature.set = P)

####### Compute lower and upper approximation #####
roughset <- BC.LU.approximation.RST(decision.table, IND)

summary(roughset)
```

---

summary.PositiveRegion

The summary function of positive region based on RST and FRST

Description

This function enables the output of a summary of the positive region and degree of dependency.

Usage

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

Arguments

object	a "PositiveRegion" object. See BC.positive.reg.FRST and BC.positive.reg.RST.
...	the other parameters.

Author(s)

Lala Septem Riza

Examples

```
dt.ex1 <- data.frame(c(1,0,2,1,1,2,2,0), c(0, 1,0, 1,0,2,1,1),
                       c(2,1,0,0,2,0,1,1), c(2,1,1,2,0,1,1,0), c(0,2,1,2,1,1,2,1))
colnames(dt.ex1) <- c("aa", "bb", "cc", "dd", "ee")
decision.table <- SF.asDecisionTable(dataset = dt.ex1, decision.attr = 5,
                                     indx.nominal = c(1:5))
```

```
## in this case, we consider second and third attributes only
P <- c(2,3)

####### Perform indiscernibility relation #######
IND <- BC.IND.relation.RST(decision.table, feature.set = P)

####### Perform lower and upper approximations #####
roughset <- BC.LU.approximation.RST(decision.table, IND)

####### Determine the positive region ######
region <- BC.positive.reg.RST(decision.table, roughset)

summary(region)
```

---

summary.RuleSetFRST   *The summary function of rules based on FRST*

Description

This function enables the output of a summary of the rule induction methods.

Usage

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

Arguments

object	a "RuleSetFRST" object. See RI.hybridFS.FRST and RI.GFRS.FRST.
...	the other parameters.

Value

a description that contains the following information:

- The type of the considered model.
- The type of the considered method.
- The type of the considered task.
- The type of similarity.
- The type of triangular norm.
- The names of attributes and their type (whether nominal or not).
- The interval of the data.
- the variance values of the data.
- The rules. Every rule constitutes two parts which are IF and THEN parts. For example, "IF pres is around 90 and preg is around 8 THEN class is 2". See RI.GFRS.FRST.

Author(s)

Lala Septem Riza

Examples

```
############################################################
## Example 1: Regression problem
############################################################
data(RoughSetData)
decision.table <- RoughSetData$housing7.dt

control <- list(type.aggregation = c("t.tnorm", "lukasiewicz"), type.relation =
                c("tolerance", "eq.3"), t.implicator = "lukasiewicz")
res.1 <- RI.hybridFS.FRST(decision.table, control)

summary(res.1)
############################################################
## Example 2: Classification problem
#############################################################
data(RoughSetData)
decision.table <- RoughSetData$pima7.dt

control <- list(type.aggregation = c("t.tnorm", "lukasiewicz"), type.relation =
                c("tolerance", "eq.3"), t.implicator = "lukasiewicz")
res.2 <- RI.hybridFS.FRST(decision.table, control)

summary(res.2)
```

---

summary.RuleSetRST   *The summary function of rules based on RST*

Description

This function enables the output of a summary of the rule induction methods.

Usage

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

Arguments

object	a "RuleSetRST" object. See RI.indiscernibilityBasedRules.RST.
...	the other parameters.

Value

a description that contains the following information:

- The type of the considered model.
- The type of the considered method.
- The type of the considered task.
- The rules. Every rule constitutes two parts which are IF and THEN parts. For example, `"IF pres is around 90 and preg is around 8 THEN class is 2; (support=4;laplace=0.67)"`.

Author(s)

Lala Septem Riza and Andrzej Janusz

Examples

```
#############################################################
## Example : Classification problem
#############################################################
data(RoughSetData)
decision.table <- RoughSetData$hiring.dt

## determine feature subset/reduct
reduct <- FS.permutation.heuristic.reduct.RST(decision.table,  permutation = NULL)

rules <- RI.indiscernibilityBasedRules.RST(decision.table, reduct)

summary(rules)
```

X.entropy   *The entropy measure*

Description

An auxiliary function for the `qualityF` parameter in the `FS.greedy.heuristic.reduct.RST`, `FS.DAAR.heuristic.RST` and `FS.greedy.heuristic.superreduct.RST` functions. It is based on *entropy* as a measure of information(Shannon, 1948).

Usage

```
X.entropy(decisionDistrib)
```

Arguments

decisionDistrib

an integer vector corresponding to a distribution of attribute values

Value

a numeric value indicating entropy of an attribute.

Author(s)

Andrzej Janusz

References

C. E. Shannon, "A Mathematical Theory of Communication", Bell System Technical Journal, vol. 27, p. 379 - 423, 623 - 656 (1948).

X.gini   *The gini-index measure*

Description

An auxiliary function for the qualityF parameter in the `FS.greedy.heuristic.reduct.RST`, `FS.DAAR.heuristic.RST` and `FS.greedy.heuristic.superreduct.RST` functions. It is based on the *gini* index as a measure of information (Stoffel and Raileanu, 2000).

Usage

```
X.gini(decisionDistrib)
```

Arguments

decisionDistrib

an integer vector corresponding to a distribution of attribute values.

Value

a numeric value indicating the gini index of an attribute.

Author(s)

Andrzej Janusz

References

K. Stoffel and L. E. Raileanu, "Selecting Optimal Split-Functions with Linear Threshold Unit Trees and Madaline-Style Networks", in: Research and Development in Intelligent Systems XVII, BCS Conference Series (2000).

X.laplace   *Rule voting by the Laplace estimate*

Description

A function returning a weight of rule's vote understood as the Laplace estimate of its confidence.

Usage

```
X.laplace(rule)
```

Arguments

rule   a decision rule, i.e. element of an "RuleSetRST" object

Value

a numerical weight of the vote

Author(s)

Andrzej Janusz

See Also

Other currently available voting methods are: `X.ruleStrength`, `X.rulesCounting`.

X.nOfConflicts   *The discernibility measure*

Description

It is an auxiliary function for the qualityF parameter in the `FS.greedy.heuristic.reduct.RST` and `FS.greedy.heuristic.superreduct.RST` functions.

Usage

```
X.nOfConflicts(decisionDistrib)
```

Arguments

decisionDistrib

               an integer vector corresponding to a distribution of decision attribute values

Value

a numeric value indicating a number of conflicts in a decision attribute

Author(s)

Andrzej Janusz

X.rulesCounting   *Rule voting by counting matching rules*

Description

A function returning an equal vote's weight for every rule. It corresponds to voting by counting the matching rules.

Usage

```
X.rulesCounting(rule)
```

Arguments

rule   a decision rule, i.e. element of an "RuleSetRST" object

Value

a numerical weight of the vote

Author(s)

Andrzej Janusz

See Also

Other currently available voting methods are: `X.ruleStrength`, `X.laplace`.

X.ruleStrength   *Rule voting by strength of the rule*

Description

A function returning a weight of rule's vote understood as strength of the rule. It is defined as a product of a cardinality of a support of a rule and the length of this rule.

Usage

```
X.ruleStrength(rule)
```

Arguments

rule   a decision rule, i.e. element of a "RuleSetRST" object

Value

a numerical weight of the vote

Author(s)

Andrzej Janusz

See Also

Other currently available voting methods are: `X.laplace`, `X.rulesCounting`.

[.RuleSetRST   *The* [. *method for* "RuleSetRST" *objects*

Description

Subsetting a set of decision rules.

Usage

```
## S3 method for class 'RuleSetRST'
x[i, ...]
```

Arguments

x	a "RuleSetRST" object from which to extract rules(s) or in which to replace rules(s). See `RI.LEM2Rules.RST`.
i	integer indices specifying elements to be extracted or replaced.
...	the other parameters.

Value

A subset of rules.

Author(s)

Andrzej Janusz

Examples

```
##############################################################
## Example : Subsetting a set of decision rules
##############################################################
data(RoughSetData)
hiring.data <- RoughSetData$hiring.dt

rules <- RI.LEM2Rules.RST(hiring.data)
```

```
rules

# taking a subset of rules
rules[1:3]
rules[c(TRUE,FALSE,TRUE,FALSE)]

# replacing a subset of rules
rules2 <- rules
rules2[c(2,4)] <- rules[c(1,3)]
rules2
```

Index