Package ‘frbs’

July 2, 2014

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

License GPL (>= 2)

Title Fuzzy Rule-based Systems for Classification and Regression Tasks

Author Lala Septem Riza, Christoph Bergmeir, Francisco Herrera, and Jose Manuel Benitez

Description This package implements functionality and various algorithms to build and use fuzzy rule-based systems (FRBSs). FRBSs are based on the concept of fuzzy sets, proposed by Zadeh in 1965, which aims at representing the reasoning of human experts in a set of IF-THEN rules, to handle real-life problems in, e.g., control, prediction and inference, data mining, bioinformatics data processing, and robotics. FRBSs are also known as fuzzy inference systems and fuzzy models. During the modeling of an FRBS, there are two important steps that need to be conducted: structure identification and parameter estimation. Nowadays, there exists a wide variety of algorithms to generate fuzzy IF-THEN rules automatically from numerical data, covering both steps. Approaches that have been used in the past are, e.g., heuristic procedures, neuro-fuzzy techniques, clustering methods, genetic algorithms, squares methods, etc. This package aims to implement the most widely used standard procedures, thus offering a standard package for FRBS modeling to the R community.

Version 2.2-0

URL http://sci2s.ugr.es/dicits/software/FRBS

Date 2014-02-03

Suggests class, e1071

Collate 'FRBS.MainFunction.R' 'FSpacePartition.Method.R' 'FSpacePartition.FunctionCollection.R' 'FSpacePartition.Predict.R' 'FNN.FunctionCollection.R' 'FNN.Methods.R' 'FCluster.Methods.R' 'FCluster.FunctionCollection.R' 'FCluster.Predict.R' 'FGradDescent.Methods.R' 'FGradDescent.FunctionCollection.R' 'GFS.Methods.R' 'GFS.FunctionCollection.R' 'GFS.Predict.R' 'docData.R' 'frbs-package.R'

NeedsCompilation no

Repository CRAN

Date/Publication 2014-02-03 12:49:34

R topics documented:

`frbs-package` *Getting started with the frbs package*

Description

Fuzzy rule-based systems (FRBSs) are based on the fuzzy concept proposed by Zadeh in 1965, which represents the reasoning of human experts in production rules (a set of IF-THEN rules) to handle real-life problems from domains such as control, prediction and inference, data mining, bioinformatics data processing, robotics, and speech recognition. FRBSs are also known as fuzzy inference systems and fuzzy models. When applied to specific tasks, they may also be known under specific names such as fuzzy associative memories or fuzzy controllers. In this package, we consider systems with multi-inputs and single-output (MISO), with real-valued data.

Details

FRBSs are a competitive alternative to other classic models and algorithms in order to solve classification and regression problems. Generally, an FRBS consists of four functional parts:

- a fuzzification interface which transforms the crisp inputs into degrees of membership function of the linguistic term of each variable. See `fuzzifier`.
- a knowledge base consisting of a database (DB) and a rulebase (RB). While the database includes the fuzzy set definitions, the rulebase contains the fuzzy IF-THEN rules. We will represent the knowledge as a set of rules. Each one has the following structure.

  `IF premise (antecedent), THEN conclusion   (consequent)`

  See `rulebase`.
- an inference engine which performs the inference operations on the fuzzy IF-THEN rules. There are two kinds of inference for fuzzy systems based on linguistic rules: The Mamdani and the Takagi Sugeno Kang model. See `inference`.
- a defuzzification process to obtain the crisp values from linguistic values. There are several methods for defuzzification such as the weighted average, centroid, etc. See `defuzzifier`.

Since it may be difficult to obtain information from human experts in the form required, an alternative and effective way to acquire the knowledge is to generate the fuzzy IF-THEN rules automatically from the numerical training data. In general, when modeling an FRBS, there are two important processes which should be conducted, namely structure identification and parameter estimation. Structure identification is a process to find appropriate fuzzy IF-THEN rules and to determine the overall number of rules. Parameter estimation is applied to tune the parameters on the consequent and/or antecedent parts of the fuzzy IF-THEN rules. Many approaches have been proposed in order to perform this modeling such as a table-lookup scheme, heuristic procedures, neuro-fuzzy techniques, clustering methods, genetic algorithms, least squares methods, gradient descent, etc. In this package, the following approaches to generate fuzzy IF-THEN rules have been implemented:

1. FRBS based on space partition

- Wang and Mendel's technique (`WM`): It is used to solve regression tasks. See `WM`.
- Chi's technique (`FRBCS.CHI`): It is used to solve classification tasks. See `FRBCS.CHI`.
- Ishibuchi's technique using weight factor (`FRBCS.W`): It is used to solve classification tasks. See `FRBCS.W`.

2. FRBS based on neural networks
   - The adaptive-network-based fuzzy inference system (`ANFIS`): It is used to solve regression tasks. See `ANFIS`.
   - The hybrid neural fuzzy inference system (`HYFIS`): It is used to solve regression tasks. See `HyFIS`.
3. FRBS based on clustering approach
   - The subtractive clustering and fuzzy c-means (`SBC`): It is used to solve regression tasks. See `SBC`.
   - The dynamic evolving neural-fuzzy inference system (`DENFIS`): It is used to solve regression tasks. See `DENFIS`.
4. FRBS based on genetic algorithms
   - The Thrift's method (`GFS.THRIFT`): It is used to solve regression tasks. See `GFS.Thrift`.
   - The Genetic fuzzy systems for fuzzy rule learning based on the MOGUL methodology (`GFS.FR.MOGUL`): It is used to solve regression tasks. See `GFS.FR.MOGUL`.
   - The Ishibuchi's method based on genetic cooperative-competitive learning (`GFS.GCCL`): It is used to solve classification tasks. See `GFS.GCCL`.
   - The Ishibuchi's method based on hybridization of genetic cooperative-competitive learning (GCCL) and Pittsburgh (`FH.GBML`): It is used to solve classification tasks. See `FH.GBML`.
   - The structural learning algorithm on vague environtment (`SLAVE`): It is used to solve classification tasks. See `SLAVE`.
   - The genetic for lateral tuning and rule selection of linguistic fuzzy system (`GFS.LT.RS`): It is used to solve regression tasks. See `GFS.LT.RS`.
5. FRBS based on the gradient descent method
   - The FRBS using heuristics and gradient descent method (`FS.HGD`): It is used to solve regression tasks. See `FS.HGD`
   - The fuzzy inference rules by descent method (`FIR.DM`): It is used to solve regression tasks. See `FIR.DM`

The functions documented in the manual for the single methods are all called internally by `frbs.learn`, which is the central function of the package. However, in the documentation of each of the internal learning functions, we give some theoretical background and references to the original literature.

Usage of the package:

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.

The main functions of the package are the following:

- The function `frbs.learn` allows to generate the model by creating fuzzy IF-THEN rules or cluster centers from training data. The different algorithms mentioned above are all accessible through this function. The outcome of the function is an `frbs-object`.

- Even though the main purpose of this package is to generate the FRBS models automatically, we provide the function `frbs.gen`, which can be used to build a model manually without using a learning method.
- The purpose of the function `predict` is to obtain predicted values according to the testing data and the model (analogous to the `predict` function that is implemented in many other R packages).
- There exist functions `summary.frbs` and `plotMF` to show a summary about an `frbs-object`, and to plot the shapes of the membership functions.

To get started with the package, the user can have a look at the examples included in the documentation of the functions `frbs.learn` for generating models and `predict` for the prediction phase.

Also, 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 present as R scripts in the package sources in the `"demo"` subdirectory. Note that some of them may take quite a long time which depends on specification hardwares.

Currently, there are the following demos available:

Regression using the Gas Furnance dataset:

`demo(WM.GasFur)`, `demo(SBC.GasFur)`, `demo(ANFIS.GasFur)`,

`demo(FS.HGD.GasFur)`, `demo(DENFIS.GasFur)`, `demo(HyFIS.GasFur)`,

`demo(FIR.DM.GasFur)`, `demo(GFS.FR.MOGUL.GasFur)`,

`demo(GFS.THRIFT.GasFur)`, `demo(GFS.LT.RS.GasFur)`.

Regression using the Mackey-Glass dataset:

`demo(WM.MG1000)`, `demo(SBC.MG1000)`, `demo(ANFIS.MG1000)`,

`demo(FS.HGD.MG1000)`, `demo(DENFIS.MG1000)`, `demo(HyFIS.MG1000)`,

`demo(GFS.THRIFT.MG1000)`, `demo(FIR.DM.MG1000)`,

`demo(GFS.FR.MOGUL.MG1000)`, `demo(GFS.LT.RS.MG1000)`.

Classification using the Iris dataset:

`demo(FRBCS.W.Iris)`, `demo(FRBCS.CHI.Iris)`, `demo(GFS.GCCL.Iris)`,

`demo(FH.GBML.Iris)`, `demo(SLAVE.Iris)`.

Generating FRBS model without learning process:

`demo(FRBS.Mamdani.Manual)`, `demo(FRBS.TSK.Manual)` `demo(FRBS.Manual)`

The Gas Furnance data and Mackey-Glass data are included in the package, please see `frbsData`. The Iris data is the standard Iris dataset that ships with R.

Also have a look at the package webpage `http://sci2s.ugr.es/dicits/software/FRBS`, 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>,

Christoph Bergmeir <c.bergmeir@decsai.ugr.es>,

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

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

DiCITS Lab, SCI2S group, DECSAI, University of Granada.

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

References

C.C. Lee, "Fuzzy Logic in control systems: Fuzzy Logic controller part I", IEEE Trans. Syst., Man, Cybern., vol. 20, no.2, pp. 404 - 418 (1990).

C.C. Lee, "Fuzzy Logic in control systems: Fuzzy Logic controller part II", IEEE Trans. Syst., Man, Cybern., vol. 20, no.2, pp. 419 - 435 (1990).

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

Mamdani, E. H. and Assilian, S., "An experiment in linguistic synthesis with a fuzzy logic controller," International Journal of Man Machine Studies, vol. 7, no. 1, pp. 1 - 13 (1975).

M. Sugeno and G. T. Kang, "Structure identification of fuzzy model," Fuzzy Sets Syst., vol. 28, pp. 15 - 33 (1988).

O. Cordon, et. al., "Genetic fuzzy systems evolutionary tuning and learning of fuzzy knowledge bases", Advances in Fuzzy Systems - Applications and Theory, vol. 19, World Scientific (2001).

Takagi, T., Sugeno, M., "Fuzzy identification of systems and its application to modelling and control", IEEE Transactions on Systems, Man and Cybernetics, vol. 15, no. 1, pp. 116 - 132 (1985).

W. Pedrycz, "Fuzzy Control and Fuzzy Systems," New York: Wiley (1989).

See Also

`frbs.learn` and `predict` for the learning and predicting stage, respectively.

Examples

```
##################################
## I. Regression Problem
##################################

## Input data: Using the Gas Furnace dataset
## then split the data to be training and testing datasets
data(frbsData)
data.train <- frbsData$GasFurnance.dt[1 : 204, ]
data.tst <- frbsData$GasFurnance.dt[205 : 292, 1 : 2]
real.val <- matrix(frbsData$GasFurnance.dt[205 : 292, 3], ncol = 1)

## Define interval of data
range.data <-apply(data.train,2,range)

## Set the method and its parameters,
```

```
## for example, we use Wang and Mendel's algorithm
method.type <- "WM"
control <- list(num.labels = 15, type.mf = "GAUSSIAN", type.defuz = "WAM",
           type.tnorm = "MIN", type.snorm = "MAX", type.implication.func = "ZADEH",
               name="sim-0")

## Learning step: Generate an FRBS model
object.reg <- frbs.learn(data.train, range.data, method.type, control)

## Predicting step: Predict for newdata
res.test <- predict(object.reg, data.tst)

## Display the FRBS model
summary(object.reg)

## Plot the membership functions
plotMF(object.reg)

##################################
## II. Classification Problem
##################################

## Input data: Using the Iris dataset
data(iris)
set.seed(2)

## Shuffle the data
## then split the data to be training and testing datasets
irisShuffled <- iris[sample(nrow(iris)),]
irisShuffled[,5] <- unclass(irisShuffled[,5])
tra.iris <- irisShuffled[1:105,]
tst.iris <- irisShuffled[106:nrow(irisShuffled),1:4]
real.iris <- matrix(irisShuffled[106:nrow(irisShuffled),5], ncol = 1)

## Define range of input data. Note that it is only for the input variables.
range.data.input <- apply(iris[,-ncol(iris)], 2,range)

## Set the method and its parameters. In this case we use FRBCS.W algorithm
method.type <- "FRBCS.W"
control <- list(num.labels = 7, type.mf = "GAUSSIAN", type.tnorm = "MIN",
               type.snorm = "MAX", type.implication.func = "ZADEH")

## Learning step: Generate fuzzy model
object.cls <- frbs.learn(tra.iris, range.data.input, method.type, control)

## Predicting step: Predict newdata
res.test <- predict(object.cls, tst.iris)

## Display the FRBS model
summary(object.cls)

## Plot the membership functions
plotMF(object.cls)
```

ANFIS *ANFIS model building*

Description

This is the internal function that implements the adaptive-network-based fuzzy inference system (ANFIS). It is used to solve regression tasks. Users do not need to call it directly, but just use `frbs.learn` and `predict`.

Usage

```
ANFIS(data.train, num.labels, max.iter = 10,
  step.size = 0.01, type.tnorm = "MIN",
  type.snorm = "MAX", type.implication.func = "ZADEH")
```

Arguments

`data.train` a matrix $(m \times n)$ of normalized data for the training process, where m is the number of instances and n is the number of variables; the last column is the output variable. Note the data must be normalized between 0 and 1.

`num.labels` a matrix $(1 \times n)$, whose elements represent the number of labels (linguistic terms); n is the number of variables.

`max.iter` the maximal number of iterations.

`step.size` a real number between 0 and 1 representing the step size of the gradient descent.

`type.tnorm` the type of t-norm. For more detail, please have a look at `inference`.

`type.snorm` the type of s-norm. For more detail, please have a look at `inference`.

`type.implication.func`
a value representing type of implication function. For more detail, please have a look at `WM`.

Details

This method was proposed by J. S. R. Jang. It uses the Takagi Sugeno Kang model on the consequent part of the fuzzy IF-THEN rules. The ANFIS architecture consists of two processes, the forward and the backward stage. The forward stage has five layers as follows:

- Layer 1: The fuzzification process which transforms crisp values into linguistic terms using the Gaussian function as the shape of the membership function.
- Layer 2: The inference stage using the t-norm operator (the AND operator).
- Layer 3: Calculating the ratio of the strengths of the rules.
- Layer 4: Calculating the consequent parameters.
- Layer 5: Calculating the overall output as the sum of all incoming signals.

The backward stage is a process of parameter learning. In this step, the least squares method is used in order to obtain the parameters, which are coefficients of linear equations on the consequent part, and mean and variance on the antecedent part.

References

J. S. R. Jang, "ANFIS: adaptive-network-based fuzzy inference system", IEEE Transactions on Systems, Man, and Cybernetics, vol. 23, no. 3, pp. 665 - 685 (1993).

J. S. R. Jang, C. T. Sun, and E. Mizutani., "Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence", Prentice-Hall, Inc (1997).

See Also

`ANFIS.update`, `frbs.learn`, and `predict`

`ANFIS.update` *ANFIS updating function*

Description

The role of this function is to update parameters in the ANFIS method. This function is called by the main function of the ANFIS method, `ANFIS`.

Usage

```
ANFIS.update(data.train, def, rule.data.num, miu.rule,
  func.tsk, varinp.mf, step.size = 0.01)
```

Arguments

`data.train`	a matrix ($m \times n$) of normalized data for the training process, where m is the number of instances and n is the number of variables; the last column is the output variable.
`def`	a predicted value
`rule.data.num`	a matrix containing the rule base in integer form.
`miu.rule`	a matrix with the degrees of rules. See `inference`.
`func.tsk`	a matrix of parameters of the function on the consequent part using the Takagi Sugeno Kang model.
`varinp.mf`	a matrix of parameters of membership functions of the input variables.
`step.size`	a real number between 0 and 1 representing the step size of the gradient descent.

`data.gen3d` *A data generator*

Description

The purpose of this function is to generate data, which contains two input variables and one output variable, automatically for all values on a plane.

Usage

```
data.gen3d(range.input, num.grid = 10)
```

Arguments

`range.input`	the range of the input variables, as a matrix $(2 \times n)$.
`num.grid`	a number representing the size of the grid on the plane.

Value

the data

Examples

```
range.input <- matrix(c(0, 100, 0, 100), nrow=2)
num.grid <- 10
data.test <- data.gen3d(range.input, num.grid)
```

`defuzzifier` *Defuzzifier to transform from linguistic terms to crisp values*

Description

Defuzzification is a transformation that extracts the crisp values from the linguistic terms.

Usage

```
defuzzifier(data, rule = NULL, range.output = NULL,
  names.varoutput = NULL, varout.mf = NULL, miu.rule,
  type.defuz = NULL, type.model = "TSK", func.tsk = NULL)
```

Arguments

`data` a matrix ($m \times n$) of data, where m is the number of instances and n is the number of variables.

`rule` a list or matrix of fuzzy IF-THEN rules, as discussed in `rulebase`.

`range.output` a matrix ($2 \times n$) containing the range of the output data.

`names.varoutput`
a list for giving names to the linguistic terms. See `rulebase`.

`varout.mf` a matrix constructing the membership function of the output variable. See `fuzzifier`.

`miu.rule` the results of the inference module. See `inference`.

`type.defuz` the type of defuzzification to be used as follows.

- 1 or `WAM` means weighted average method,
- 2 or `FIRST.MAX` means first maxima,
- 3 or `LAST.MAX` means last maxima,
- 4 or `MEAN.MAX` means mean maxima,
- 5 or `COG` means modified center of gravity (COG).

`type.model` the type of the model that will be used in the simulation. Here, `1` or `MAMDANI` and `2` or `TSK` means we use Mamdani or Takagi Sugeno Kang model, respectively.

`func.tsk` a matrix used to build the linear equation for the consequent part if we are using Takagi Sugeno Kang. See also `rulebase`.

Details

In this function, there exist two kinds of models which are based on the Mamdani and Takagi Sugeno Kang model. For the Mamdani model there are five methods for defuzzifying a linguistic term A of a universe of discourse Z. They are as follows:

1. weighted average method (`WAM`).
2. first of maxima (`FIRST.MAX`).
3. last of maxima (`LAST.MAX`)
4. mean of maxima (`MEAN.MAX`).
5. modified center of gravity (`COG`).

Value

A matrix of crisp values

See Also

`fuzzifier`, `rulebase`, and `inference`

DENFIS *DENFIS model building*

Description

This is the internal function that implements the dynamic evolving neural-fuzzy inference system (DENFIS). It is used to handle regression tasks. Users do not need to call it directly, but just use `frbs.learn` and `predict`.

Usage

```
DENFIS(data.train, range.data.ori, Dthr = 0.1,
  max.iter = 100, step.size = 0.01, d = 2)
```

Arguments

`data.train`	a matrix ($m \times n$) of data for the training process, where m is the number of instances and n is the number of variables (input and output variables).
`range.data.ori`	a matrix ($2 \times n$) containing the range of the data, where n is the number of variables, and first and second rows are the minimum and maximum values, respectively.
`Dthr`	the threshold value for the evolving clustering method (ECM), between 0 and 1.
`max.iter`	the maximal number of iterations.
`step.size`	the step size of the least squares method, between 0 and 1.
`d`	a parameter for the width of the triangular membership function.

Details

This method was proposed by Nikola K. Kasabov and Q. Song. There are several steps in this method that are to determine the cluster centers using the evolving clustering method (ECM), to partition the input space and to find optimal parameters on the consequent part (Takagi Sugeno Kang model) for the IF-THEN rule using a least squares estimator.

ECM is a distance-based clustering method which is determined by a threshold value, `Dthr`. This parameter influences how many clusters are created. In the beginning of the clustering process, the first instance from the training data is chosen to be a cluster center, and the determining radius is set to zero. Afterwards, using the next instance, cluster centers and radius are changed based on certain mechanisms of ECM (please see `ECM`). All of the cluster centers are then obtained after evaluating all the training data. The next step is to update the parameters on the consequent part with the assumption that the antecedent part which we got from ECM is fixed. Actually, ECM can perform well as an online clustering method, but in this package it is used in an offline mode.

References

N. K. Kasabov and Q. Song, "DENFIS: Dynamic evolving neural-fuzzy inference system and its Application for time-series prediction", IEEE Transactions on Fuzzy Systems, vol. 10, no. 2, pp. 144 - 154 (2002).

See Also

DENFIS.eng, frbs.learn, and predict

DENFIS.eng *DENFIS prediction function*

Description

This function is an internal function for the prediction phase using the DENFIS method. The user should use this function not directly, but with calling predict.

Usage

```
DENFIS.eng(object, newdata)
```

Arguments

object	the frbs model. See frbs-object.
newdata	a matrix ($m \times n$) of data for the prediction process, where m is the number of instances and n is the number of input variables.

Value

a matrix of predicted values

See Also

DENFIS

---

denorm.data *The data de-normalization*

Description

This function is to transform from normalized data into real-valued data.

Usage

```
denorm.data(dt.norm, range.data, min.scale = 0,
  max.scale = 1)
```

Arguments

`dt.norm`	a matrix ($n \times m$) of the normalized data.
`range.data`	a matrix ($2 \times n$) containing the range of the data, where n is the number of variables, and first and second rows are the minimum and maximum value, respectively.
`min.scale`	the minimum value within normalization.
`max.scale`	the maximum value within normalization.

Value

the real-valued data

See Also

`norm.data`

`DM.update` *FIR.DM updating function*

Description

The role of this function is to update the parameters of the fuzzy inference rules by descent method (FIR.DM). This function is called by the main function of the FIR.DM method, `FIR.DM`.

Usage

```
DM.update(data.train, rule.data.num, miu.rule, func.tsk,
  varinp.mf, step.size = 0.01, def)
```

Arguments

`data.train`	a matrix ($m \times n$) of normalized data, where m is the number of instances and n is the number of variables; the last column is the output variable.
`rule.data.num`	a matrix containing the rulebase. Its elements are integers, see `rulebase`.
`miu.rule`	a matrix with the degrees of rules which is a result of the `inference`.
`func.tsk`	a matrix of parameters of the functions on the consequent part of the Takagi Sugeno Kang model.
`varinp.mf`	a matrix of parameters of the membership functions of the input variables.
`step.size`	the step size of the descent method, between 0 and 1.
`def`	a matrix which is obtained from the defuzzification. Please have a look at `defuzzifier`.

See Also

`frbs.learn`, `predict`, and `FIR.DM`.

ECM *Evolving Clustering Method*

Description

This function is a part of the DENFIS method to generate cluster centers.

Usage

```
ECM(data.train, Dthr)
```

Arguments

`data.train`	a matrix ($m \times n$) of data for training, where m is the number of instances and n is the number of variables where the last column is the output variable.
`Dthr`	the threshold value for the evolving clustering method (ECM), between 0 and 1.

Value

a matrix of cluster centers

See Also

`DENFIS` and `DENFIS.eng`

FH.GBML *FH.GBML model building*

Description

This is the internal function that implements the Ishibuchi's method based on hybridization of genetic cooperative-competitive learning (GCCL) and Pittsburgh (FH.GBML). It is used to solve classification tasks. Users do not need to call it directly, but just use `frbs.learn` and `predict`.

Usage

```
FH.GBML(data.train, popu.size = 10, max.num.rule = 5,
  persen_cross = 0.6, persen_mutant = 0.3, max.gen = 10,
  num.class, range.data.input, p.dcare = 0.5,
  p.gccl = 0.5)
```

Arguments

- `data.train` a matrix ($m \times n$) of normalized data for the training process, where m is the number of instances and n is the number of variables; the last column is the output variable. Note the data must be normalized between 0 and 1.
- `popu.size` the size of the population which is generated in each generation.
- `max.num.rule` the maximum number of rules.
- `persen_cross` a real number between 0 and 1 determining the probability of crossover.
- `persen_mutant` a real number between 0 and 1 determining the probability of mutation.
- `max.gen` the maximal number of generations for the genetic algorithms.
- `num.class` a number of the classes.
- `range.data.input` a matrix containing the ranges of the normalized input data.
- `p.dcare` a probability of "don't care" attributes occurred.
- `p.gccl` a probability of GCCL process occurred.

Details

This method is based on Ishibuchi's method using the hybridization of GCCL and the Pittsburgh approach for genetic fuzzy systems. The algorithm of this method is as follows.

- Step 1: Generate population where each individual in the population is a fuzzy rule set.
- Step 2: Calculate the fitness value of each rule set in the current population.
- Step 3: Generate new rule sets by the selection, crossover, and mutation in the same manner as the Pittsburgh-style algorithm. Then, apply iterations of the GCCL to each of the generated rule sets with a probability.
- Step 4: Add the best rule set in the current population to newly generated rule sets to form the next population.
- Step 5: Return to Step 2 if the prespecified stopping condition is not satisfied.

References

H. Ishibuchi, T. Yamamoto, and T. Nakashima, "Hybridization of fuzzy GBML approaches for pattern classification problems," IEEE Trans. on Systems, Man, and Cybernetics-Part B: Cybernetics, vol. 35, no. 2, pp. 359 - 365 (2005).

`FIR.DM` *FIR.DM model building*

Description

This is the internal function that implements the fuzzy inference rules by descent method (FIR.DM). It is used to solve regression tasks. Users do not need to call it directly, but just use `frbs.learn` and `predict`.

Usage

```
FIR.DM(data.train, num.labels, max.iter, step.size,
  type.tnorm = "MIN", type.snorm = "MAX",
  type.implication.func = "ZADEH")
```

Arguments

`data.train`	a matrix ($m \times n$) of normalized data for training, where m is the number of instances and n is the number of variables. The last column is the output variable. Note the data must be normalized between 0 and 1.
`num.labels`	a matrix ($1 \times n$) whose elements represent the number of labels (fuzzy terms), where n is the number of variables.
`max.iter`	the maximal number of iterations.
`step.size`	the step size of the descent method, between 0 and 1.
`type.tnorm`	the type of t-norm. For more detail, please have a look at `inference`.
`type.snorm`	the type of s-norm. For more detail, please have a look at `inference`.
`type.implication.func`	a value representing type of implication function. For more detail, please have a look at `WM`

Details

This method was proposed by H. Nomura, I. Hayashi, and N. Wakami. FIR.DM uses simplified fuzzy reasoning where the consequent part is a real number (a particular case within the Takagi Sugeno Kang model), while the membership function on the antecedent part is expressed by an isosceles triangle. So, in the learning phase, FIR.DM updates three parameters which are center and width of the triangular and a real number on the consequent part using a descent method.

References

H. Nomura, I. Hayashi and N. Wakami, "A learning method of fuzzy inference rules by descent method", IEEE International Conference on Fuzzy Systems, pp. 203 - 210 (1992).

See Also

`DM.update`, `frbs.learn`, and `predict`.

`FRBCS.CHI` *FRBCS.CHI model building*

Description

This is the internal function that implements the fuzzy rule-based classification system using Chi's technique (FRBCS.CHI). It is used to solve classification tasks. Users do not need to call it directly, but just use `frbs.learn` and `predict`. This method is suitable only for classification problems.

Usage

```
FRBCS.CHI(range.data, data.train, num.labels, num.class,
  type.mf = "TRIANGLE", type.tnorm = "MIN",
  type.snorm = "MAX", type.implication.func = "ZADEH")
```

Arguments

`range.data`	a matrix ($2 \times n$) containing the range of the normalized data, where n is the number of variables, and first and second rows are the minimum and maximum values, respectively.
`data.train`	a matrix ($m \times n$) of normalized data for the training process, where m is the number of instances and n is the number of variables; the last column is the output variable. Note the data must be normalized between 0 and 1.
`num.labels`	a matrix ($1 \times n$), whose elements represent the number of labels (linguistic terms); n is the number of variables.
`num.class`	an integer number representing the number of labels (linguistic terms).
`type.mf`	the type of the shape of the membership functions. See `fuzzifier`.
`type.tnorm`	the type of t-norm. See `inference`.
`type.snorm`	the type of s-norm. See `inference`.
`type.implication.func`	the type of implication function. See `WM`.

Details

This method was proposed by Z. Chi, H. Yan, and T. Pham that extends Wang and Mendel's method for tackling classification problems. Basically, the algorithm is quite similar as Wang and Mendel's technique. However, since it is based on the FRBCS model, Chi's method only takes class labels on each data to be consequent parts of fuzzy IF-THEN rules. In other words, we generate rules as in Wang and Mendel's technique (`WM`) and then we replace consequent parts with their classes. Regarding calculating degress of each rule, they are determined by antecedent parts of the rules. Redudant rules can be deleted by considering their degrees. Lastly, we obtain fuzzy IF-THEN rules based on the FRBCS model.

References

Z. Chi, H. Yan, T. Pham, "Fuzzy algorithms with applications to image processing and pattern recognition", World Scientific, Singapore (1996).

See Also

`FRBCS.eng`, `frbs.learn`, and `predict`

FRBCS.eng — *FRBCS: prediction phase*

Description

This function is the internal function of the fuzzy rule-based classification systems (FRBCS) to compute the predicted values.

Usage

```
FRBCS.eng(object, newdata)
```

Arguments

`object`	the `frbs-object`.
`newdata`	a matrix ($m \times n$) of data for the prediction process, where m is the number of instances and n is the number of input variables.

Value

A matrix of predicted values.

FRBCS.W — *FRBCS.W model building*

Description

This is the internal function that implements the fuzzy rule-based classification system with weight factor (FRBCS.W). It is used to solve classification tasks. Users do not need to call it directly, but just use `frbs.learn` and `predict`. This method is suitable only for classification problems.

Usage

```
FRBCS.W(range.data, data.train, num.labels, num.class,
  type.mf, type.tnorm = "MIN", type.snorm = "MAX",
  type.implication.func = "ZADEH")
```

Arguments

`range.data`	a matrix ($2 \times n$) containing the range of the normalized data, where n is the number of variables, and first and second rows are the minimum and maximum values, respectively.
`data.train`	a matrix ($m \times n$) of normalized data for the training process, where m is the number of instances and n is the number of variables; the last column is the output variable. Note the data must be normalized between 0 and 1.

num.labels	a matrix ($1 \times n$), whose elements represent the number of labels (linguistic terms); n is the number of variables.
num.class	an integer number representing the number of labels (linguistic terms).
type.mf	the type of the shape of the membership functions.
type.tnorm	the type of t-norm. See inference.
type.snorm	the type of s-norm. See inference.
type.implication.func	the type of implication function. See WM.

Details

This method is adopted from Hisao Ishibuchi and Tomoharu Nakashima's paper. Each fuzzy IF-THEN rule consists of antecedent linguistic values and a single consequent class with certainty grades (weights). The antecedent part is determined by a grid-type fuzzy partition from the training data. The consequent class is defined as the dominant class in the fuzzy subspace corresponding to the antecedent part of each fuzzy IF-THEN rule and the certainty grade is calculated from the ratio among the consequent class. A class of the new instance is determined by the consequent class of the rule with the maximal product of the compatibility grade and the certainty grade.

References

H. Ishibuchi and T. Nakashima, "Effect of rule weights in fuzzy rule-based classification systems", IEEE Transactions on Fuzzy Systems, vol. 1, pp. 59 - 64 (2001).

See Also

FRBCS.eng, frbs.learn, and predict

---

frbs.eng *The prediction phase*

Description

This function is one of the main internal functions of the package. It determines the values within the prediction phase.

Usage

```
frbs.eng(object, newdata)
```

Arguments

object	the frbs-object.
newdata	a matrix ($m \times n$) of data for the prediction process, where m is the number of instances and n is the number of input variables.

Details

This function involves four different processing steps on fuzzy rule-based systems. Firstly, the rulebase (see `rulebase`) validates the consistency of the fuzzy IF-THEN rules form. Then, the fuzzification (see `fuzzifier`) transforms crisp values into linguistic terms. Next, the inference calculates the degree of rule strengths using the t-norm and the s-norm. Finally, the defuzzification process calculates the results of the model using the Mamdani or the Takagi Sugeno Kang model.

Value

A list with the following items:

`rule`	the fuzzy IF-THEN rules
`varinp.mf`	a matrix to generate the shapes of the membership functions for the input variables
`MF`	a matrix of the degrees of the membership functions
`miu.rule`	a matrix of the degrees of the rules
`func.tsk`	a matrix of the Takagi Sugeno Kang model for the consequent part of the fuzzy IF-THEN rules
`predicted.val`	a matrix of the predicted values

See Also

`fuzzifier`, `rulebase`, `inference` and `defuzzifier`.

`frbs.gen` *The frbs model generator*

Description

The purpose of this function is to generate a FRBS model from user-given input without a learning process.

Usage

```
frbs.gen(range.data, num.fvalinput, names.varinput,
  num.fvaloutput = NULL, varout.mf = NULL,
  names.varoutput = NULL, rule, varinp.mf,
  type.model = "MAMDANI", type.defuz = "WAM",
  type.tnorm = "MIN", type.snorm = "MAX",
  func.tsk = NULL, colnames.var = NULL,
  type.implication.func = "ZADEH", name = "Sim-0")
```

Arguments

`range.data`
: a matrix ($2 \times n$) containing the range of the data, where n is the number of variables, and first and second rows are the minimum and maximum values, respectively.

`num.fvalinput`
: a matrix representing the number of linguistic terms of each input variables.
For example: `num.fvalinput <- matrix(c(3,2), nrow =   1)`
means that there are two variables where the first variable has three linguistic terms and the second one has two linguistic terms.

`varinp.mf`
: a matrix for constructing the shapes of the membership functions. See how to construct it in `fuzzifier`.

`names.varinput`
: a list giving names to the linguistic terms for input variables. See `rulebase`.

`num.fvaloutput`
: the number of linguistic terms of the output variable. This parameter is required for Mamdani model only.
For example: `num.fvaloutput <- matrix(3, nrow = 1)`
means there are 3 linguistic terms for the output variable.

`varout.mf`
: a matrix for constructing the membership functions of the output variable. The form is the same as for the `varinp.mf` parameter. This parameter is required for Mamdani model only. See `fuzzifier`.

`names.varoutput`
: a list giving names of the linguistic terms for the output variable. The form is the same as for the `names.varinput` parameter. This parameter is required for Mamdani model only. See `rulebase`.

`rule`
: a list of fuzzy IF-THEN rules. There are some types of rule structures, for example: Mamdani, Takagi Sugeno Kang, and fuzzy rule-based classification systems (FRBCS). It allows to involve linguistic hedge, negation operator, etc. For more detail, see `rulebase`.

`type.model`
: the type of the model. There are three types as follows.
- `MAMDANI` means we are using Mamdani model.
- `TSK` means we are using Takagi Sugeno Kang model.
- `FRBCS` means we are using fuzzy rule-based classification systems (FRBCS).

`type.defuz`
: the type of the defuzzification method. It is used in Mamdani model only. See `defuzzifier`.

`type.tnorm`
: the type of the t-norm method. See `inference`.

`type.snorm`
: the type of the s-norm method. See `inference`.

`func.tsk`
: a matrix of parameters of the function on the consequent part using the Takagi Sugeno Kang model. This parameter must be defined when we are using Takagi Sugeno Kang. See `rulebase`.

`colnames.var`
: a list of names of input and output variables.

`type.implication.func`
: a type of implication function. See `WM`.

`name`
: a name of the simulation.

Details

It can be used if rules have already been obtained manually, without employing the learning process. In the examples shown, we generate a fuzzy model using `frbs.gen` and generate the fuzzy rule-based systems step by step manually. Additionally, the examples show several scenarios as follows.

- Using `frbs.gen` for constructing Mamdani model on a regression task.
- Using `frbs.gen` for constructing Takagi Sugeno Kang model on a regression task.
- Constructing Mamdani model by executing internal functions such as `rulebase`, `fuzzifier`, `inference`, and `defuzzifier` for Mamdani model.
- Using `frbs.gen` for constructing fuzzy rule-based classification systems (FRBCS) model.

Value

The `frbs-object`.

Examples

```
#################################################
## 1. The following codes show how to generate a fuzzy model
## using the frbs.gen function for regression tasks.
## The following are three scenarios:
## 1a. Using Mamdani model
## 1b. Using Takagi Sugeno Kang model
## 1c. Using Mamdani model and internal functions: fuzzifier, etc.
## Note:
## In the examples, let us consider four input variabels and one output variable.
## Some variables could be shared together for other examples.
#################################################

## Define shape and parameters of membership functions of input variables.
## Please see fuzzifier function to construct the matrix.
varinp.mf <- matrix(c(2, 0, 20, 40, NA, 4, 20, 40, 60, 80, 3, 60, 80, 100, NA,
                      2, 0, 35, 75, NA, 3, 35, 75, 100, NA,
                      2, 0, 20, 40, NA, 1, 20, 50, 80, NA, 3, 60, 80, 100, NA,
                      2, 0, 20, 40, NA, 4, 20, 40, 60, 80, 3, 60, 80, 100, NA),
                      nrow = 5, byrow = FALSE)

## Define number of linguistic terms of input variables.
## Suppose, we have 3, 2, 3, and 3 numbers of linguistic terms
## for first, second, third and fourth variables, respectively.
num.fvalinput <- matrix(c(3, 2, 3, 3), nrow=1)

## Give the names of the linguistic terms of each input variables.
## It should be noted that the names of the linguistic terms must be unique,
## so we put a number for making it unique.
varinput.1 <- c("a1", "a2", "a3")
varinput.2 <- c("b1", "b2")
varinput.3 <- c("c1", "c2", "c3")
varinput.4 <- c("d1", "d2", "d3")
names.varinput <- c(varinput.1, varinput.2, varinput.3, varinput.4)
```

```
## Set interval of data.
range.data <- matrix(c(0,100, 0, 100, 0, 100, 0, 100, 0, 100), nrow=2)

## Define inference parameters.
type.defuz <- "WAM"
type.tnorm <- "MIN"
type.snorm <- "MAX"
type.implication.func <- "ZADEH"

## Give the name of simulation.
name <- "Sim-0"

## Provide new data for testing.
newdata<- matrix(c(25, 40, 35, 15, 45, 75, 78, 70), nrow= 2, byrow = TRUE)
## the names of variables
colnames.var <- c("input1", "input2", "input3", "input4", "output1")

####################################################################
## 1a. Using Mamdani Model
#####################################################################
## Define number of linguistic terms of output variable.
## In this case, we set the number of linguistic terms to 3.
num.fvaloutput <- matrix(c(3), nrow=1)

## Give the names of the linguistic terms of the output variable.
## Note: the names of the linguistic terms must be unique.
varoutput.1 <- c("e1", "e2", "e3")
names.varoutput <- c(varoutput.1)

## Define the shapes and parameters of the membership functions of the output variables.
varout.mf <- matrix(c(2, 0, 20, 40, NA, 4, 20, 40, 60, 80, 3, 60, 80, 100, NA),
                      nrow = 5, byrow = FALSE)

## Set type of model which is "MAMDANI" or "TSK" for Mamdani or
## Takagi Sugeno Kang model, respectively.
## In this case, we choose Mamdani model.
type.model <- "MAMDANI"

## Define the fuzzy IF-THEN rules;
## there are two kinds of model: Mamdani and Takagi Sugeno Kang model
## if we use the Mamdani model then the consequent part is a linguistic term,
## but if we use Takagi Sugeno Kang then we build a matrix representing
## linear equations in the consequent part.
## In this example we are using the Mamdani model
## (see the type.model parameter).
## Note:
## "a1", "and", "b1, "->", "e1" means that
## "IF inputvar.1 is a1 and inputvar.2 is b1 THEN outputvar.1 is e1"
## Make sure that each rule has a "->" sign.
rule <- matrix(c("a1","and","b1","and","c1","and","d1","->","e1",
                 "a2","and","b2","and","c2","and","d2", "->", "e2",
                 "a3","and","b2","and","c2","and","d1", "->", "e3"),
                 nrow=3, byrow=TRUE)
```

```
## Generate a fuzzy model with frbs.gen.
object <- frbs.gen(range.data, num.fvalinput, names.varinput,
                 num.fvaloutput, varout.mf, names.varoutput, rule,
                 varinp.mf, type.model, type.defuz, type.tnorm,
                 type.snorm, func.tsk = NULL, colnames.var, type.implication.func, name)

## Plot the membership function.
plotMF(object)

## Predicting using new data.
res <- predict(object, newdata)$predicted.val

#####################################################################
## 1b. Using Takagi Sugeno Kang (TSK) Model
#####################################################################
## Define "TSK" for Takagi Sugeno Kang model
type.model <- "TSK"

## Define linear equation for consequent parts.
func.tsk <- matrix(c(1, 1, 5, 2, 1, 3, 1, 0.5, 0.1, 2, 1, 3, 2, 2, 2), nrow=3, byrow=TRUE)

## Define the fuzzy IF-THEN rules;
## For TSK model, it isn't necessary to put linguistic term in consequent parts.
## Make sure that each rule has a "->" sign.
rule <- matrix(c("a1","and","b1","and","c1","and","d1","->",
                 "a2","and","b2","and","c2","and","d2", "->",
                 "a3","and","b2","and","c2","and","d1", "->"),
                 nrow=3, byrow=TRUE)

## Generate a fuzzy model with frbs.gen.
## It should be noted that for TSK model, we do not need to input:
## num.fvaloutput, varout.mf, names.varoutput, type.defuz.
object <- frbs.gen(range.data, num.fvalinput, names.varinput,
             num.fvaloutput = NULL, varout.mf = NULL, names.varoutput = NULL, rule,
varinp.mf, type.model, type.defuz = NULL, type.tnorm, type.snorm,
             func.tsk, colnames.var, type.implication.func, name)

## Plot the membership function.
plotMF(object)

## Predicting using new data.
res <- predict(object, newdata)$predicted.val

######################
## 1c. Using the same data as in the previous example, this example performs
## step by step of the generation of a fuzzy rule-based system
######################
## Using Mamdani model.
type.model <- "MAMDANI"

## Construct rules.
rule <- matrix(c("a1","and","b1","and","c1","and","d1","->","e1",
```

```
                 "a2","and","b2","and","c2","and","d2", "->", "e2",
                 "a3","and","b2","and","c2","and","d1", "->", "e3"),
                 nrow=3, byrow=TRUE)

## Check input data given by user.
rule <- rulebase(type.model, rule, func.tsk = NULL)

## Fuzzification Module:
## In this function, we convert crisp into linguistic values/terms
## based on the data and the parameters of the membership function.
## The output: a matrix representing the degree of the membership of the data
num.varinput <- ncol(num.fvalinput)
MF <- fuzzifier(newdata, num.varinput, num.fvalinput, varinp.mf)

## Inference Module:
## In this function, we will calculate the confidence factor on the antecedent for each rule
## considering t-norm and s-norm.
miu.rule <- inference(MF, rule, names.varinput, type.tnorm, type.snorm)

## Defuzzification Module.
## In this function, we calculate and convert the linguistic values back into crisp values.
range.output <- range.data[, ncol(range.data), drop = FALSE]
result <- defuzzifier(newdata, rule, range.output, names.varoutput,
                  varout.mf, miu.rule, type.defuz, type.model, func.tsk = NULL)


#################################################
## 2. The following codes show how to generate a fuzzy model
## using the frbs.gen function for classification tasks using Mamdani model.
#################################################
## define range of data.
## Note. we only define range of input data.
range.data.input <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1), nrow=2)

## Define shape and parameters of membership functions of input variables.
## Please see fuzzifier function to construct the matrix.
## In this case, we are using TRIANGLE for membership functions.
varinp.mf <- matrix(c(1, 0, 0, 0.5, NA, 1, 0, 0.5, 1, NA, 1, 0.5, 1, 1, NA,
                      1, 0, 0, 0.5, NA, 1, 0, 0.5, 1, NA, 1, 0.5, 1, 1, NA,
                      1, 0, 0, 0.5, NA, 1, 0, 0.5, 1, NA, 1, 0.5, 1, 1, NA,
                      1, 0, 0, 0.5, NA, 1, 0, 0.5, 1, NA, 1, 0.5, 1, 1, NA),
                      nrow = 5, byrow = FALSE)

## Define number of linguistic terms of input variables.
## Suppose, we have 3, 3, 3, and 3 numbers of linguistic terms
## for first up to fourth variables, respectively.
num.fvalinput <- matrix(c(3, 3, 3, 3), nrow=1)

## Give the names of the linguistic terms of each input variable.
## It should be noted that the names of the linguistic terms must be unique,
## so we put a number for making it unique.
varinput.1 <- c("v.1_a.1", "v.1_a.2", "v.1_a.3")
varinput.2 <- c("v.2_a.1", "v.2_a.2", "v.2_a.3")
```

```
varinput.3 <- c("v.3_a.1", "v.3_a.2", "v.3_a.3")
varinput.4 <- c("v.4_a.1", "v.4_a.2", "v.4_a.3")
names.varinput <- c(varinput.1, varinput.2, varinput.3, varinput.4)

## Provide inference parameters.
type.tnorm <- "MIN"
type.snorm <- "MAX"
type.implication.func <- "ZADEH"
type.model <- "FRBCS"

## Give the name of simulation.
name <- "Sim-0"

## Provide new data for testing.
newdata<- matrix(c(0.45, 0.5, 0.89, 0.44, 0.51, 0.99, 0.1, 0.98, 0.51,
                 0.56, 0.55, 0.5), nrow= 3, byrow = TRUE)

## the names of variables
colnames.var <- c("input1", "input2", "input3", "input4", "output1")

## Construct rules.
## Take into account in consequent part, which expresses classes.
rule <- matrix(c("v.1_a.2","and","v.2_a.2","and","v.3_a.3","and","v.4_a.2","->","3",
         "v.1_a.2","and","v.2_a.3","and","v.3_a.1","and","v.4_a.3","->","1",
         "v.1_a.2","and","v.2_a.2","and","v.3_a.2","and","v.4_a.2","->","2"),
         nrow=3, byrow=TRUE)

## Generate frbs object.
object <- frbs.gen(range.data = range.data.input, num.fvalinput,
             names.varinput, num.fvaloutput = NULL, varout.mf = NULL,
             names.varoutput = NULL, rule, varinp.mf, type.model,
             type.defuz = NULL, type.tnorm, type.snorm, func.tsk = NULL,
             colnames.var, type.implication.func, name)

## Plot the shape of membership functions.
plotMF(object)

## Predicting using new data.
res <- predict(object, newdata)
```

`frbs.learn` *The frbs model building function*

Description

This is one of the central functions of the package. This function is used to generate/learn the model from numerical data using fuzzy rule-based systems.

Usage

```
frbs.learn(data.train, range.data = NULL,
  method.type = c("WM"), control = list())
```

Arguments

data.train
: a data frame or matrix ($m \times n$) of data for the training process, where m is the number of instances and n is the number of variables; the last column is the output variable. It should be noted that the training data must be expressed in numbers (numerical data). And, especially for classification tasks, the last column representing class names/symbols isn't allowed to have values 0 (zero). In the other words, the categorical values 0 should be replaced with other values.

range.data
: a matrix ($2 \times n$) containing the range of the data, where n is the number of variables, and first and second rows are the minimum and maximum values, respectively. It should be noted that for `"FRBCS.W"`, `"FRBCS.CHI"`, `"GFS.GCCL"`, `"FH.GBML"`, and `"SLAVE"`, n represents the number of input variables only (without the output variable). It will be assigned as min/max of training data if it is omitted.

method.type
: this parameter determines the learning algorithm to be used. The following methods are implemented:

- `"WM"`: Wang and Mendel's technique to handle regression tasks. See `WM`;
- `"SBC"`: subtractive clustering method to handle regression tasks. See `SBC`;
- `"HYFIS"`: hybrid neural fuzzy inference systems to handle regression tasks. See `HyFIS`;
- `"ANFIS"`: adaptive neuro-fuzzy inference systems to handle regression tasks. See `ANFIS`;
- `"FRBCS.W"`: fuzzy rule-based classification systems with weight factor based on Ishibuchi's method to handle classification tasks. See `FRBCS.W`;
- `"FRBCS.CHI"`: fuzzy rule-based classification systems based on Chi's method to handle classification tasks. See `FRBCS.CHI`;
- `"DENFIS"`: dynamic evolving neuro-fuzzy inference systems to handle regression tasks. See `DENFIS`;
- `"FS.HGD"`: fuzzy system using heuristic and gradient descent method to handle regression tasks. See `FS.HGD`;
- `"FIR.DM"`: fuzzy inference rules by descent method to handle regression tasks. See `FIR.DM`;
- `"GFS.FR.MOGUL"`: genetic fuzzy systems for fuzzy rule learning based on the MOGUL methodology to handle regression tasks. See `GFS.FR.MOGUL`;
- `"GFS.THRIFT"`: Thrift's technique based on genetic algorithms to handle regression tasks. See `GFS.Thrift`;
- `"GFS.GCCL"`: Ishibuchi's method based on genetic cooperative-competitive learning to handle classification tasks. See `GFS.GCCL`;
- `"FH.GBML"`: Ishibuchi's method based on hybridization of genetic cooperative-competitive learning and Pittsburgh to handle classification tasks. See `FH.GBML`;
- `"SLAVE"`: structural learning algorithm on vague environment to handle classification tasks. See `SLAVE`;

- "GFS.LT.RS": genetic algorithm for lateral tuning and rule selection. See GFS.LT.RS

`control` a list containing all arguments, depending on the learning algorithm to use. The following list are parameters required for each methods, whereas their descriptions will be explained later on.

- WM:
  `list(num.labels, type.mf, type.tnorm, type.snorm, type.defuz, type.implication.func, name)`
- HYFIS:
  `list(num.labels, max.iter, step.size, type.tnorm, type.snorm, type.defuz, type.implication.func, name)`
- ANFIS and FIR.DM:
  `list(num.labels, max.iter, step.size, type.tnorm, type.snorm, type.implication.func , name)`
- SBC:
  `list(r.a, eps.high, eps.low, name)`
- FS.HGD:
  `list(num.labels, max.iter, step.size, alpha.heuristic, type.tnorm, type.snorm, type.implication.func, name)`
- FRBCS.W and FRBCS.CHI:
  `list(num.labels, type.mf, type.tnorm, type.snorm, type.implication.func, name)`
- DENFIS method:
  `list(Dthr, max.iter, step.size, d, name)`
- GFS.FR.MOGUL:
  `list(persen_cross, max.iter, max.gen, max.tune, persen_mutant, epsilon, name)`
- GFS.THRIFT method:
  `list(popu.size, num.labels, persen_cross, max.gen, persen_mutant, type.tnorm, type.snorm, type.defuz, type.implication.func, name)`
- GFS.GCCL:
  `list(popu.size, num.class, num.labels, persen_cross, max.gen, persen_mutant, name)`
- FH.GBML:
  `list(popu.size, max.num.rule, num.class, persen_cross, max.gen, persen_mutant, p.dcare, p.gccl, name)`
- SLAVE:
  `list(num.class, num.labels, persen_cross, max.iter, max.gen, persen_mutant, k.lower, k.upper, epsilon, name)`
- GFS.LT.RS:
  `list(popu.size, num.labels, persen_mutant, max.gen, mode.tuning, type.tnorm, type.snorm, type.implication.func, type.defuz, rule.selection, name)`

Description of the `control` Parameters

- `num.labels`: a positive integer to determine the number of labels (linguistic terms). The default value is 7.
- `type.mf`: the following type of the membership function. The default value is `GAUSSIAN`. For more detail, see `fuzzifier`.
  - `TRIANGLE`: it refers triangular shape.
  - `TRAPEZOID`: it refers trapezoid shape.
  - `GAUSSIAN`: it refers gaussian shape.
  - `SIGMOID`: it refers sigmoid.
  - `BELL`: it refers generalized bell.
- `type.defuz`: the type of the defuzzification method as follows. The default value is `WAM`. For more detail, see `defuzzifier`.
  - `WAM`: the weighted average method.
  - `FIRST.MAX`: the first maxima.
  - `LAST.MAX`: the last maxima.
  - `MEAN.MAX`: the mean maxima.
  - `COG`: the modified center of gravity (COG).
- `type.tnorm`: the type of conjunction operator (t-norm). The following are options of t-norm available. For more detail, please have a look at `inference`. The default value is `MIN`.
  - `MIN` means standard type (minimum).
  - `HAMACHER` means Hamacher product.
  - `YAGER` means Yager class (with tao = 1).
  - `PRODUCT` means product.
  - `BOUNDED` mean bounded product.
- `type.snorm`: the type of disjunction operator (s-norm). The following are options of s-norm available. For more detail, please have a look at `inference`. The default value is `MAX`.
  - `MAX` means standard type (maximum).
  - `HAMACHER` means Hamacher sum.
  - `YAGER` means Yager class (with tao = 1).
  - `SUM` means sum.
  - `BOUNDED` mean bounded sum.
- `type.implication.func`: the type of implication function. The following are options of implication function available: `DIENES_RESHER`, `LUKASIEWICZ`, `ZADEH`, `GOGUEN`, `GODEL`, `SHARP`, `MIZUMOTO`, `DUBOIS_PRADE`, and `MIN`. For more detail, please have a look at `WM`. The default value is `ZADEH`.
- `name`: a name for the model. The default value is `"sim-0"`.
- `max.iter`: a positive integer to determine the maximal number of iterations. The default value is 10.
- `step.size`: the step size of the gradient descent, a real number between 0 and 1. The default value is 0.01.
- `r.a`: a positive constant which is effectively the radius defining a neighborhood. The default value is 0.5.
- `eps.high`: an upper threshold value. The default value is 0.5.

- eps.low: a lower threshold value. The default value is 0.15.
- alpha.heuristic: a positive real number representing a heuristic value. The default value is 1.
- Dthr: the threshold value for the envolving clustering method (ECM), between 0 and 1. The default value is 0.1.
- d: a parameter for the width of the triangular membership function. The default value is 2.
- persen_cross: a probability of crossover. The default value is 0.6.
- max.gen: a positive integer to determine the maximal number of generations of the genetic algorithm. The default value is 10.
- max.tune: a positive integer to determine the maximal number of tuning iterations. The default value is 10.
- persen_mutant: a probability of mutation. The default value is 0.3.
- epsilon: a real number between 0 and 1 representing the level of generalization. A high epsilon can lead to overfitting. The default value is 0.9.
- popu.size: the size of the population which is generated in each generation. The default value is 10.
- max.num.rule: the maximum size of the rules. The default value is 5.
- num.class: the number of classes.
- p.dcare: a probability of "don't care" attributes. The default value is 0.5.
- p.gccl: a probability of the GCCL process. The default value is 0.5.
- k.lower: a lower bound of the noise threshold with interval between 0 and 1. The default value is 0.
- k.upper: an upper bound of the noise threshold with interval between 0 and 1. The default value is 1.
- mode.tuning: a type of lateral tuning which are "LOCAL" or "GLOBAL". The default value is "GLOBAL".
- rule.selection:a boolean value representing whether performs rule selection or not. The default value is "TRUE".

Details

This function makes accessible all learning methods that are implemented in this package. All of the methods use this function as interface for the learning stage, so users do not need to call other functions in the learning phase. In order to obtain good results, users need to adjust some parameters such as the number of labels, the type of the shape of the membership function, the maximal number of iterations, the step size of the gradient descent, or other method-dependent parameters which are collected in the control parameter. After creating the model using this function, it can be used to predict new data with predict.

Value

The frbs-object.

See Also

`predict` for the prediction phase, and the following main functions of each of the methods for theoretical background and references: `WM`, `SBC`, `HyFIS`, `ANFIS`, `FIR.DM`, `DENFIS`, `FS.HGD`, `FRBCS.W`, `FRBCS.CHI`, `GFS.FR.MOGUL`, `GFS.Thrift`, `GFS.GCCL`, `FH.GBML`, `GFS.LT.RS`, and `SLAVE`.

Examples

```
##################################
## I. Regression Problem
## Suppose data have two input variables and one output variable.
## We separate them into training, fitting, and testing data.
## data.train, data.fit, data.test, and range.data are inputs
## for all regression methods.
###################################
## Take into account that the simulation might take a long time
## depending on the hardware you are using. The chosen parameters
## may not be optimal.
## Data must be in data.frame or matrix form and the last column
## is the output variable/attribute.
## The training data must be expressed in numbers (numerical data).
data.train <- matrix(c(5.2, -8.1, 4.8, 8.8, -16.1, 4.1, 10.6, -7.8, 5.5, 10.4, -29.0,
                      5.0, 1.8, -19.2, 3.4, 12.7, -18.9, 3.4, 15.6, -10.6, 4.9, 1.9,
                      -25.0, 3.7, 2.2, -3.1, 3.9, 4.8, -7.8, 4.5, 7.9, -13.9, 4.8,
                      5.2, -4.5, 4.9, 0.9, -11.6, 3.0, 11.8, -2.1, 4.6, 7.9, -2.0,
                      4.8, 11.5, -9.0, 5.5, 10.6, -11.2, 4.5, 11.1, -6.1, 4.7, 12.8,
                      -1.0, 6.6, 11.3, -3.6, 5.1, 1.0, -8.2, 3.9, 14.5, -0.5, 5.7,
                      11.9, -2.0, 5.1, 8.1, -1.6, 5.2, 15.5, -0.7, 4.9, 12.4, -0.8,
                      5.2, 11.1, -16.8, 5.1, 5.1, -5.1, 4.6, 4.8, -9.5, 3.9, 13.2,
                      -0.7, 6.0, 9.9, -3.3, 4.9, 12.5, -13.6, 4.1, 8.9, -10.0,
                      4.9, 10.8, -13.5, 5.1), ncol = 3, byrow = TRUE)
colnames(data.train) <- c("inp.1", "inp.2", "out.1")

data.fit <- data.train[, -ncol(data.train)]

data.test <- matrix(c(10.5, -0.9, 5.8, -2.8, 8.5, -0.6, 13.8, -11.9, 9.8, -1.2, 11.0,
                     -14.3, 4.2, -17.0, 6.9, -3.3, 13.2, -1.9), ncol = 2, byrow = TRUE)

range.data <- matrix(apply(data.train, 2, range), nrow = 2)

#############################################################
## I.1 Example: Implementation of Wang & Mendel
#############################################################
method.type <- "WM"

## collect control parameters into a list
## num.labels = 3 means we define 3 as the number of linguistic terms
control.WM <- list(num.labels = 3, type.mf = "GAUSSIAN", type.tnorm = "MIN", type.snorm = "MAX",
type.defuz = "WAM", type.implication.func = "ZADEH", name = "Sim-0")

## generate the model and save it as object.WM
object.WM <- frbs.learn(data.train, range.data, method.type, control.WM)
```

```
#############################################################
## I.2 Example: Implementation of SBC
#############################################################
## Not run: method.type <- "SBC"
control.SBC <- list(r.a = 0.5, eps.high = 0.5, eps.low = 0.15, name = "Sim-0")

object.SBC <- frbs.learn(data.train, range.data, method.type, control.SBC)

#############################################################
## I.3 Example: Implementation of HYFIS
#############################################################
method.type <- "HYFIS"

control.HYFIS <- list(num.labels = 5, max.iter = 50, step.size = 0.01, type.tnorm = "MIN",
                      type.snorm = "MAX", type.defuz = "COG",
                      type.implication.func = "ZADEH", name = "Sim-0")

object.HYFIS <- frbs.learn(data.train, range.data, method.type, control.HYFIS)

#############################################################
## I.4 Example: Implementation of ANFIS
#############################################################
method.type <- "ANFIS"

control.ANFIS <- list(num.labels = 5, max.iter = 10, step.size = 0.01, type.tnorm = "MIN",
                    type.snorm = "MAX", type.implication.func = "ZADEH", name = "Sim-0")

object.ANFIS <- frbs.learn(data.train, range.data, method.type, control.ANFIS)

#############################################################
## I.5 Example: Implementation of DENFIS
#############################################################

control.DENFIS <- list(Dthr = 0.1, max.iter = 10, step.size = 0.001, d = 2,
                       name = "Sim-0")
method.type <- "DENFIS"

object.DENFIS <- frbs.learn(data.train, range.data, method.type, control.DENFIS)

#############################################################
## I.6 Example: Implementation of FIR.DM
#############################################################
method.type <- "FIR.DM"

control.DM <- list(num.labels = 5, max.iter = 10, step.size = 0.01, type.tnorm = "MIN",
                    type.snorm = "MAX", type.implication.func = "ZADEH", name = "Sim-0")
object.DM <- frbs.learn(data.train, range.data, method.type, control.DM)

#############################################################
## I.7 Example: Implementation of FS.HGD
#############################################################
method.type <- "FS.HGD"
```

```
control.HGD <- list(num.labels = 5, max.iter = 10, step.size = 0.01,
               alpha.heuristic = 1, type.tnorm = "MIN", type.snorm = "MAX",
               type.implication.func = "ZADEH", name = "Sim-0")
object.HGD <- frbs.learn(data.train, range.data, method.type, control.HGD)

#############################################################
## I.8 Example: Implementation of GFS.FR.MOGUL
#############################################################
method.type <- "GFS.FR.MOGUL"

control.GFS.FR.MOGUL <- list(persen_cross = 0.6,
                    max.iter = 5, max.gen = 2, max.tune = 2, persen_mutant = 0.3,
                    epsilon = 0.8, name="sim-0")
object.GFS.FR.MOGUL <- frbs.learn(data.train, range.data,
                       method.type, control.GFS.FR.MOGUL)

#############################################################
## I.9 Example: Implementation of Thrift's method (GFS.THRIFT)
#############################################################
method.type <- "GFS.THRIFT"

control.Thrift <- list(popu.size = 6, num.labels = 3, persen_cross = 1,
                      max.gen = 5, persen_mutant = 1, type.tnorm = "MIN",
                      type.snorm = "MAX", type.defuz = "COG",
                      type.implication.func = "ZADEH", name="sim-0")
object.Thrift <- frbs.learn(data.train, range.data, method.type, control.Thrift)

##############################################################
## I.10 Example: Implementation of
##      genetic for lateral tuning and rule selection (GFS.LT.RS)
#############################################################
## Set the method and its parameters
method.type <- "GFS.LT.RS"

control.lt.rs <- list(popu.size = 5, num.labels = 5, persen_mutant = 0.3,
             max.gen = 10, mode.tuning = "LOCAL", type.tnorm = "MIN", type.snorm = "MAX",
                type.implication.func = "ZADEH", type.defuz = "WAM",
                rule.selection = TRUE, name="sim-0")

## Generate fuzzy model
object.lt.rs <- frbs.learn(data.train, range.data, method.type, control.lt.rs)
## End(Not run)

#############################################################
## II. Classification Problems
#############################################################
## The iris dataset is shuffled and divided into training and
## testing data. Bad results in the predicted values may result
## from casual imbalanced classes in the training data.
## Take into account that the simulation may take a long time
## depending on the hardware you use.
## One may get better results with other parameters.
## Data are in data.frame or matrix form and the last column is
```

```
## the output variable/attribute
## The data must be expressed in numbers (numerical data).

data(iris)
irisShuffled <- iris[sample(nrow(iris)),]
irisShuffled[,5] <- unclass(irisShuffled[,5])
tra.iris <- irisShuffled[1:105,]
tst.iris <- irisShuffled[106:nrow(irisShuffled),1:4]
real.iris <- matrix(irisShuffled[106:nrow(irisShuffled),5], ncol = 1)

## Please take into account that the interval needed is the range of input data only.
range.data.input <- matrix(apply(iris[, -ncol(iris)], 2, range), nrow = 2)

#########################################################
## II.1 Example: Implementation of FRBCS with weighted factor based on Ishibuchi's method
###############################################################
## generate the model
method.type <- "FRBCS.W"
control <- list(num.labels = 3, type.mf = "TRIANGLE", type.tnorm = "MIN",
               type.snorm = "MAX", type.implication.func = "ZADEH", name = "sim-0")

object <- frbs.learn(tra.iris, range.data.input, method.type, control)

## conduct the prediction process
res.test <- predict(object, tst.iris)

#########################################################
## II.2 Example: Implementation of FRBCS based on Chi's method
###############################################################
## generate the model
## Not run: method.type <- "FRBCS.CHI"
control <- list(num.labels = 7, type.mf = "TRIANGLE", type.tnorm = "MIN",
               type.snorm = "MAX", type.implication.func = "ZADEH", name = "sim-0")

object <- frbs.learn(tra.iris, range.data.input, method.type, control)

## conduct the prediction process
res.test <- predict(object, tst.iris)

#########################################################
## II.3 The example: Implementation of GFS.GCCL
###############################################################
method.type <- "GFS.GCCL"

control <- list(popu.size = 5, num.class = 3, num.labels = 5, persen_cross = 0.9,
                    max.gen = 2, persen_mutant = 0.3,
                    name="sim-0")
## Training process
## The main result of the training is a rule database which is used later for prediction.
object <- frbs.learn(tra.iris, range.data.input, method.type, control)

## Prediction process
res.test <- predict(object, tst.iris)
```

```
#########################################################
## II.4 Example: Implementation of FH.GBML
###############################################################
method.type <- "FH.GBML"

control <- list(popu.size = 5, max.num.rule = 5, num.class = 3,
persen_cross = 0.9, max.gen = 2, persen_mutant = 0.3, p.dcare = 0.5,
             p.gccl = 1, name="sim-0")

## Training process
## The main result of the training is a rule database which is used later for prediction.
object <- frbs.learn(tra.iris, range.data.input, method.type, control)

## Prediction process
res.test <- predict(object, tst.iris)

#########################################################
## II.5 The example: Implementation of SLAVE
###############################################################
method.type <- "SLAVE"

control <- list(num.class = 3, num.labels = 5,
persen_cross = 0.9, max.iter = 5, max.gen = 3, persen_mutant = 0.3,
             k.lower = 0.25, k.upper = 0.75, epsilon = 0.1, name="sim-0")

## Training process
## The main result of the training is a rule database which is used later for prediction.
object <- frbs.learn(tra.iris, range.data.input, method.type, control)

## Prediction process
res.test <- predict(object, tst.iris)
## End(Not run)
```

---

frbsData *Data set of the package*

Description

The package includes embedded versions of the Mackey-Glass chaotic time series and the Gas Furnance dataset.

Details

Mackey-Glass chaotic time series

The Mackey-Glass chaotic time series is defined by the following delayed differential equation:

$d_x(t)/d_t = (a * x(t - \tau)/(1 + x(t - \tau)^1 0)) - b * x(t)$

For this dataset, we generated 1000 samples, with input parameters as follows:

- $a = 0.2$
- $b = 0.1$
- $\tau = 17$
- $x_0 = 1.2$
- $d_t = 1$

The dataset is embedded in the following way:

input variables: $x(t-18)$, $x(t-12)$, $x(t-6)$, $x(t)$

output variable: $x(t+6)$

Gas Furnance dataset

The Gas Furnance dataset is taken from Box and Jenkins. It consists of 292 consecutive values of methane at time $(t-4)$, and the CO2 produced in a furnance at time $(t-1)$ as input variables, with the produced CO2 at time (t) as an output variable. So, each training data point consists of $[u(t-4), y(t-1), y(t)]$, where u is methane and y is CO2.

References

Box, G. E. P., & Jenkins, G. M. "Time Series Analysis, forecasting and control", San Fransisco, CA: Holden Day (1970).

Mackey, M., & Glass, L., "Oscillation and chaos in physiological control systems", Science, vol. 197, pp. 287 - 289 (1977).

`frbsObjectFactory` *The object factory for frbs objects*

Description

This function creates objects of type `frbs`. Currently, its implementation is very basic and does no argument checking, as it is only used internally.

Usage

```
frbsObjectFactory(mod)
```

Arguments

`mod`	a list containing all the attributes for the object

Details

The members of the frbs object depend on the used learning method. The following list describes all of the members that can be present.

num.labels the number of linguistic terms for the variables

varout.mf a matrix to generate the shapes of the membership functions for the output variable. The first row represents the shape of the membership functions, the other rows contain the parameters that have been generated. Whether the values of parameters within the matrix are normalized to lie between 0 and 1 or not depends on the selected method.

rule the fuzzy IF-THEN rules; In the GFS.FR.MOGUL case, a rule refers to the parameter values of the membership function which represents the rule.

rule.data.num the fuzzy IF-THEN rules in integer format.

varinp.mf a matrix to generate the shapes of the membership functions for the input variables. The first row represents the shape of the membership functions, the other rows contain the non NA values representing the parameters related with their type of membership function. For example, TRAPEZOID, TRIANGLE, and GAUSSIAN have four, three, and two values as their parameters, respectively. Whether the values of parameters within the matrix are normalized to lie between 0 and 1 or not depends on the selected method.

type.model the type of model. Here, MAMDANI refers to Mamdani model, and TSK refers to Takagi Sugeno Kang model on the consequence part.

func.tsk a matrix of the Takagi Sugeno Kang model consequent part of the fuzzy IF-THEN rules.

class a matrix representing classes of FRBCS model

num.labels a number of linguistic terms on each variables/attributes.

type.defuz the type of the defuzzification method.

type.tnorm the type of the t-norm method.

type.snorm the type of the s-norm method.

type.mf the type of shapes of membership functions.

type.implication.func the type of the implication function.

method.type the type of the selected method.

name the name given to the model.

range.data.ori range of the original data (before normalization).

cls cluster centers.

Dthr the boundary parameter of the DENFIS method.

d the multiplier parameters of the DENFIS method.

r.a the neighborhood factor of SBC.

degree.rule certainty degree of rules.

rule.data.num a matrix representing the rules in integer form.

grade.cert grade of certainty for classification problems.

alpha.heuristic a parameter for the heuristic of the FS.HGD method.

var.mf.tune a matrix of parameters of membership function for lateral tuning.

mode.tuning a type of lateral tuning.

rule.selection a boolean of rule selection.

colnames.var the names of variables.

Value

an object of type `frbs`

`FS.HGD` *FS.HGD model building*

Description

This is the internal function that implements the simplified TSK fuzzy rule generation method using heuristics and gradient descent method (FS.HGD). It is used to solve regression tasks. Users do not need to call it directly, but just use `frbs.learn` and `predict`.

Usage

```
FS.HGD(data.train, num.labels, max.iter = 100,
  step.size = 0.01, alpha.heuristic = 1,
  type.tnorm = "MIN", type.snorm = "MAX",
  type.implication.func = "ZADEH")
```

Arguments

`data.train` a matrix ($m \times n$) of normalized data for the training process, where m is the number of instances and n is the number of variables; the last column is the output variable. Note the data must be normalized between 0 and 1.

`num.labels` a matrix ($1 \times n$), whose elements represent the number of labels (fuzzy terms); n is the number of variables.

`max.iter` maximal number of iterations.

`step.size` step size of the descent method.

`alpha.heuristic` a positive real number which is the heuristic parameter.

`type.tnorm` the type of t-norm. For more detail, please have a look at `inference`.

`type.snorm` the type of s-norm. For more detail, please have a look at `inference`.

`type.implication.func` a value representing type of implication function. For more detail, please have a look at `WM`.

Details

This method was proposed by K. Nozaki, H. Ishibuchi, and H. Tanaka. It uses fuzzy IF-THEN rules with nonfuzzy singletons (i.e. real numbers) in the consequent parts. The techniques of space partition are implemented to generate the antecedent part, while the initial consequent part of each rule is determined by the weighted mean value of the given training data. Then, the gradient descent method updates the value of the consequent part. Futhermore, the heuristic value given by the user affects the value of weight of each data.

References

H. Ishibuchi, K. Nozaki, H. Tanaka, Y. Hosaka and M. Matsuda, "Empirical study on learning in fuzzy systems by rice taste analysis", Fuzzy Set and Systems, vol. 64, no. 2, pp. 129 - 144 (1994).

See Also

`frbs.learn`, `predict`, and `HGD.update`

`fuzzifier` *Transform from crisp set into linguistic terms*

Description

Fuzzification refers to the process of transforming a crisp set into linguistic terms.

Usage

```
fuzzifier(data, num.varinput, num.labels.input,
  varinp.mf)
```

Arguments

`data` a matrix of data containing numerical elements.

`num.varinput` number of input variables.

`num.labels.input` the number of labels of the input variables.

`varinp.mf` a matrix containing the parameters to form the membership functions. The dimension of the matrix is $(5, n)$ where n is a multiplication the number of linguistic terms/labels and the number of input variables. The rows of the matrix represent: The first row is the type of membership function, where 1 means `TRIANGLE`, 2 means `TRAPEZOID` in left side, 3 means `TRAPEZOID` in right side, 4 means `TRAPEZOID` in the middle, 5 means `GAUSSIAN`, 6 means `SIGMOID`, and 7 means `BELL`. And, the second up to fifth row indicate the corner points to construct the functions.

- `TRIANGLE` has three parameters (a, b, c), where b is the center point of the `TRIANGLE`, and a and c are the left and right points, respectively.
- `TRAPEZOID` has four parameters (a, b, c, d).
- `GAUSSIAN` has two parameters ($mean$ and $variance$).
- `SIGMOID` has two parameters (γ and c).
- `BELL` has three parameters (a, b, c).

For example:

```
varinp.mf <-  matrix(c(2,1,3,2,3,0,30,60,0,40,20,50,80,
30,80,40,70,100,60,100,0,0,100,0,100), nrow=5,  byrow=TRUE)
```

Details

In this function, there are five shapes of membership functions implemented, namely `TRIANGLE`, `TRAPEZOID`, `GAUSSIAN`, `SIGMOID`, and `BELL`.

Value

A matrix of the degree of each linguistic terms based on the shape of the membership functions

See Also

`defuzzifier`, `rulebase`, and `inference`

`GFS.FR.MOGUL` *GFS.FR.MOGUL model building*

Description

This is the internal function that implements genetic fuzzy systems for fuzzy rule learning based on the MOGUL methodology (GFS.FR.MOGUL). It is used to solve regression tasks. Users do not need to call it directly, but just use `frbs.learn` and `predict`.

Usage

```
GFS.FR.MOGUL(data.train, persen_cross = 0.6,
  persen_mutant = 0.3, max.iter = 10, max.gen = 10,
  max.tune = 10, range.data.ori, epsilon = 0.4)
```

Arguments

`data.train`	a matrix ($m \times n$) of normalized data for the training process, where m is the number of instances and n is the number of variables; the last column is the output variable. Note the data must be normalized between 0 and 1.
`persen_cross`	a real number between 0 and 1 determining the probability of crossover.
`persen_mutant`	a real number between 0 and 1 determining the probability of mutation.
`max.iter`	the maximal number of iterations.
`max.gen`	the maximal number of generations of the genetic algorithm.
`max.tune`	the maximal number of tuning iterations.
`range.data.ori`	a matrix containing the ranges of the original data.
`epsilon`	a real number between 0 and 1 determining the boundary of covering factor.

Details

This method was proposed by Herrera et al. GFS.FR.MOGUL implements a genetic algorithm determining the structure of the fuzzy IF-THEN rules and the membership function parameters. There are two general types of fuzzy IF-THEN rules, namely the descriptive and the approximative/free semantic approaches. A descriptive approach means that the linguistic labels represent a real-world semantic; the linguistic labels are uniformly defined for all rules. In contrast, in the approximative approach there isn't any associated linguistic label. This method is based on the latter one. We model a fuzzy IF-THEN rule on a chromosome which consists of the parameter values of the membership function. So, every rule has its own membership function values. A population contains many such generated chromosomes, based on the iterative rule learning approach (IRL). IRL means that the chromosomes will be generated one by one, taking into account the fitness value and covering factor, until there are sufficient chromosomes in the population. After having obtained the population, the genetic algorithm is started, using the genetic operators selection, mutation, and crossover.

References

F. Herrera, M. Lozano, and J. L. Verdegay, "A learning process for fuzzy control rules using genetic algorithms", Fuzzy Sets and Systems, vol. 100, pp. 143 - 158 (1998).

O. Cordon, M. J. del Jesus, F. Herrera, M. Lozano, "MOGUL: A methodology to obtain genetic fuzzy rule-based systems under the iterative rule learning approach", International Journal of Intelligent Systems, vol. 14, pp. 1123 - 1153 (1999).

See Also

`GFS.FR.MOGUL.test`, `frbs.learn`, and `predict`

`GFS.FR.MOGUL.test` *GFS.FR.MOGUL: The prediction phase*

Description

This function is the internal function of the GFS.FR.MOGUL method to compute the predicted values.

Usage

```
GFS.FR.MOGUL.test(object, newdata)
```

Arguments

`object`	the `frbs-object`.
`newdata`	a matrix $(m \times n)$ of data for the prediction process, where m is the number of instances and n is the number of input variables.

Value

A matrix of predicted values.

GFS.GCCL *GFS.GCCL model building*

Description

This is the internal function that implements the Ishibuchi's method based on genetic cooperative-competitive learning (GFS.GCCL). It is used to handle classification tasks. Users do not need to call it directly, but just use `frbs.learn` and `predict`.

Usage

```
GFS.GCCL(data.train, popu.size = 10, range.data.input,
  num.labels, persen_cross = 0.6, persen_mutant = 0.3,
  max.gen = 10, range.data.ori)
```

Arguments

`data.train`	a matrix ($m \times n$) of normalized data for the training process, where m is the number of instances and n is the number of variables; the last column is the output variable. Note the data must be normalized between 0 and 1.
`popu.size`	the size of the population which is generated in each generation.
`range.data.input`	a matrix containing the ranges of the normalized input data.
`num.labels`	a matrix describing the number of linguistic terms.
`persen_cross`	a real number between 0 and 1 representing the probability of crossover.
`persen_mutant`	a real number between 0 and 1 representing the probability of mutation.
`max.gen`	the maximal number of generations for the genetic algorithm.
`range.data.ori`	a matrix containing the ranges of the input data.

Details

This method is based on Ishibuchi's method. In this method, a chromosome describes each linguistic IF-THEN rule using integer as its representation of the antecedent part. In the consequent part of the fuzzy rules, the heuristic method is applied to automatically generate the class. The evaluation is calculated for each rule which means that the performance is not based on the entire rule set. The outline of the method is as follows.

- Step 1: Generate an initial population of fuzzy IF-THEN rules.
- Step 2: Evaluate each fuzzy IF-THEN rule in the current population.
- Step 3: Generate new fuzzy IF-THEN rules by genetic operators.
- Step 4: Replace a part of the current population with the newly generated rules.
- Step 5: Terminate the algorithm if a stopping condition is satisfied, otherwise return to Step 2.

Additionally, to handle high dimensional data, this method uses "don't care" attributes on the antecedent fuzzy set.

References

H. Ishibuchi, T. Nakashima, and T. Murata, "Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems", IEEE trans. on Systems, Man, and Cybernetics - Part B: Sybernetics, vol. 29. no. 5, pp. 601 - 618 (1999).

`GFS.GCCL.eng` *GFS.GCCL.test: The prediction phase*

Description

This function is the internal function of the GFS.GCCL and FH.GBML method to compute the predicted values.

Usage

```
GFS.GCCL.eng(object, newdata)
```

Arguments

`object`	the `frbs-object`.
`newdata`	a matrix ($m \times n$) of data for the prediction process, where m is the number of instances and n is the number of input variables.

Value

A matrix of predicted values.

`GFS.LT.RS` *GFS.LT.RS model building*

Description

This is the internal function that implements genetic lateral tuning and rule selection of linguistic fuzzy systems (GFS.LT.RS). It is used to solve regression tasks. Users do not need to call it directly, but just use `frbs.learn` and `predict`.

Usage

```
GFS.LT.RS(data.train, popu.size = 10, range.data,
  num.labels, persen_mutant, max.gen = 10,
  mode.tuning = "GLOBAL", type.tnorm = "MIN",
  type.snorm = "MAX", type.implication.func = "ZADEH",
  type.defuz = "WAM", rule.selection = FALSE,
  range.data.ori)
```

Arguments

`data.train`	a matrix ($m \times n$) of normalized data for the training process, where m is the number of instances and n is the number of variables; the last column is the output variable. Note the data must be normalized between 0 and 1.
`popu.size`	the size of the population which is generated in each generation.
`range.data`	a matrix representing interval of data.
`num.labels`	a matrix representing the number of linguistic terms in each variables.
`persen_mutant`	a real number between 0 and 1 determining the probability of mutation.
`max.gen`	the maximal number of generations of the genetic algorithm.
`mode.tuning`	a type of tuning which are `"LOCAL"` or `"GLOBAL"`.
`type.tnorm`	a type of t-norm. See `inference`.
`type.snorm`	a type of s-norm. See `inference`.
`type.implication.func`	a type of implication function. See `WM`.
`type.defuz`	a type of defuzzification methods. See `defuzzifier`.
`rule.selection`	a boolean value representing whether performs rule selection or not.
`range.data.ori`	a matrix containing the ranges of the original data.

Details

This method was proposed by R. Alcala et al. GFS.LT.RS implements a evolutionary algorithm for postprocessing in constructing FRBS model. It uses a new rule representation model based on the linguistic 2-tupples representation that allows the lateral displacement of the labels. This function allows two different tuning which are global and local tuning.

Regarding with evolutionary algorithms, the following are main components:

- coding scheme and initial gene pool;
- chromosome evalution;
- crossover operator;
- restarting approach;
- evolutionary model;

In first time, population is constructed by Wang & Mendel's technique. Mean square error (MSE) is used to calculate chromosome evaluation. This method performs BLX-a in crossover process. Additionally, rule selection method is performed in order to minimize the number of rules.

References

R. Alcala, J. Alcala-Fdez, and F. Herrera, "A Proposal for the genetic lateral tuning of linguistic fuzzy systems and its interaction with rule selection", IEEE Trans. on Fuzzy Systems, Vol. 15, No. 4, pp. 616 - 635 (2007).

See Also

`GFS.LT.RS.test`, `frbs.learn`, and `predict`

GFS.LT.RS.test *GFS.LT.RS: The prediction phase*

Description

This function is the internal function of the GFS.LT.RS method to compute the predicted values.

Usage

```
GFS.LT.RS.test(object, newdata)
```

Arguments

object	the frbs-object.
newdata	a matrix ($m \times n$) of data for the prediction process, where m is the number of instances and n is the number of input variables.

Value

A matrix of predicted values.

GFS.Thrift *GFS.Thrift model building*

Description

This is the internal function that implements the Thrift's technique based on a genetic algorithm. It is used to tackle regression tasks. Users do not need to call it directly, but just use frbs.learn and predict.

Usage

```
GFS.Thrift(data.train, popu.size = 10, num.labels,
  persen_cross = 0.6, persen_mutant = 0.3, max.gen = 10,
  range.data.ori, type.defuz = "WAM", type.tnorm = "MIN",
  type.snorm = "MAX", type.mf = "TRIANGLE",
  type.implication.func = "ZADEH")
```

Arguments

`data.train`	a matrix ($m \times n$) of normalized data for the training process, where m is the number of instances and n is the number of variables; the last column is the output variable. Note the data must be normalized between 0 and 1.
`popu.size`	the size of the population which is generated in each generation.
`num.labels`	a matrix describing the number of linguistic terms.
`persen_cross`	a real number between 0 and 1 representing the probability of crossover.
`persen_mutant`	a real number between 0 and 1 representing the probability of mutation.
`max.gen`	the maximal number of generations for the genetic algorithm.
`range.data.ori`	a matrix containing the ranges of the original data.
`type.defuz`	the type of the defuzzification method. For more detail, see `defuzzifier`. The default value is `WAM`.
`type.tnorm`	the type of t-norm. For more detail, please have a look at `inference`.
`type.snorm`	the type of s-norm. For more detail, please have a look at `inference`.
`type.mf`	the type of shape of membership function. See `fuzzifier`.
`type.implication.func`	the type of implication function. See `WM`.

Details

This method was developed by Thrift using Mamdani's model as fuzzy IF-THEN rules. In this method, we consider a table as a genotype with alleles that are fuzzy set indicators over the output domain. The phenotype is produced by the behavior produced by the fuzzification, max-* composition, and defuzzification operations. A chromosome (genotype) is formed from the decision table by going rowwise and producing a string of numbers from the code set. Standard crossover and mutation operators can act on these string.

References

P. Thrift, "Fuzzy logic synthesis with genetic algorithms", In Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA91), San Diego (United States of America), pp. 509 - 513 (1991).

See Also

`GFS.Thrift.test`, `frbs.learn`, and `predict`

GFS.Thrift.test *GFS.Thrift: The prediction phase*

Description

This function is the internal function of the GFS.Thrift method to compute the predicted values.

Usage

```
GFS.Thrift.test(object, newdata)
```

Arguments

`object`	the `frbs-object`.
`newdata`	a matrix $(m \times n)$ of data for the prediction process, where m is the number of instances and n is the number of input variables.

Value

A matrix of predicted values.

HGD.update *FS.HGD updating function*

Description

The role of this function is to update parameters within the simplified TSK fuzzy rule generation method using heuristics and the gradient descent method (FS.HGD). This function is called by the main function of the FS.HGD method, see `FS.HGD`.

Usage

```
HGD.update(data.train, miu.rule, func.tsk, varinp.mf,
  step.size = 0.01, def)
```

Arguments

`data.train`	a matrix $(m \times n)$ of normalized data for the training process, where m is the number of instances and n is the number of variables; the last column is the output variable.
`miu.rule`	a matrix with the degrees of rules which is the result of the `inference`.
`func.tsk`	a matrix of parameters of the function on the consequent part using the Takagi Sugeno Kang model. See `rulebase`.
`varinp.mf`	a matrix of parameters of membership functions of the input variables.
`step.size`	a real number between 0 and 1 representing the step size of the gradient descent.
`def`	a matrix which is obtained by the `defuzzifier`.

See Also

FS.HGD

HyFIS *HyFIS model building*

Description

This is the internal function that implements the hybrid neural fuzzy inference system (HyFIS). It is used to solve regression tasks. Users do not need to call it directly, but just use frbs.learn and predict

Usage

```
HyFIS(data.train, num.labels, max.iter = 10,
  step.size = 0.01, type.tnorm = "MIN",
  type.snorm = "MAX", type.defuz = "COG",
  type.implication.func = "ZADEH")
```

Arguments

data.train	a matrix ($m \times n$) of normalized data for the training process, where m is the number of instances and n is the number of variables; the last column is the output variable. Note the data must be normalized between 0 and 1.
num.labels	a matrix ($1 \times n$), whose elements represent the number of labels (linguistic terms); n is the number of variables.
max.iter	the maximal number of iterations.
step.size	step size of the gradient descent method.
type.tnorm	the type of t-norm. For more detail, please have a look at inference.
type.snorm	the type of s-norm. For more detail, please have a look at inference.
type.defuz	the type of aggregation function. For more detail, please have a look at defuzzifier
type.implication.func	a value representing type of implication function. For more detail, please have a look at WM

Details

This method was proposed by J. Kim and N. Kasabov. There are two phases in this method for learning, namely the knowledge acquisition module and the structure and parameter learning. The knowledge acquition module uses the techniques of Wang and Mendel. The learning of structure and parameters is a supervised learning method using gradient descent-based learning algorithms. This function generates a model which consists of a rule database and parameters of the membership functions. The rules of HyFIS use the Mamdani model on the antecedent and consequent parts. Futhermore, HyFIS uses a Gaussian membership function. So, there are two kinds of parameters that are optimized, mean and variance of the Gaussian function.

References

J. Kim and N. Kasabov, "HyFIS: Adaptive neuro-fuzzy inference systems and their application to nonlinear dynamical systems", Neural Networks, vol. 12, no. 9, pp. 1301 - 1319 (1999).

See Also

`HyFIS.update`, `frbs.learn`, and `predict`.

`HyFIS.update` *HyFIS updating function*

Description

This function is called by `HyFIS` to update the parameters within the HyFIS method.

Usage

```
HyFIS.update(data.train, def, rule, names.varoutput,
  var.mf, miu.rule, num.labels, MF, step.size = 0.001,
  degree.rule)
```

Arguments

`data.train`	a matrix ($m \times n$) of normalized data for the training process, where m is the number of instances and n is the number of variables; the last column is the output variable.
`def`	matrix of defuzzification results. See `defuzzifier`.
`rule`	fuzzy IF-THEN rules. See `rulebase`.
`names.varoutput`	a list of names of the output variable.
`var.mf`	a matrix of parameters of the membership functions. Please see `fuzzifier`.
`miu.rule`	a matrix of degree of rules which is a result of the `inference`.
`num.labels`	a matrix ($1 \times n$) whose elements represent the number of labels (or linguistic terms), where n is the number of variables.
`MF`	a matrix of parameters of the membership functions which is a result of the `fuzzifier`.
`step.size`	a real number, the step size of the gradient descent.
`degree.rule`	a matrix of degrees of rules. See `frbs-object`.

See Also

`HyFIS`

inference *The process of fuzzy reasoning*

Description

Inference refers to the process of fuzzy reasoning.

Usage

```
inference(MF, rule, names.varinput, type.tnorm,
  type.snorm)
```

Arguments

`MF`	a matrix of the degrees of membership functions which is a result of the `fuzzifier`.
`rule`	a matrix or list of fuzzy IF-THEN rules. See `rulebase`.
`names.varinput`	a list of names of the input variables.
`type.tnorm`	a value which represents the type of t-norm to be used: • 1 or `MIN` means standard t-norm: $min(x1, x2)$. • 2 or `HAMACHER` means Hamacher product: $(x1 * x2)/(x1 + x2 - x1 * x2)$. • 3 or `YAGER` means Yager class: $1 - min(1, ((1 - x1) + (1 - x2)))$. • 4 or `PRODUCT` means product: $(x1 * x2)$. • 5 or `BOUNDED` means bounded product: $max(0, x1 + x2 - 1)$.
`type.snorm`	a value which represents the type of s-norm to be used: • 1 or `MAX` means standard s-norm: $max(x1, x2)$. • 2 or `HAMACHER` means Hamacher sum: $(x1 + x2 - 2x1 * x2)/1 - x1 * x2$. • 3 or `YAGER` means Yager class: $min(1, (x1 + x2))$. • 4 or `SUM` means sum: $(x1 + x2 - x1 * x2)$. • 5 or `BOUNDED` means bounded sum: $min(1, x1 + x2)$.

Details

In this function, fuzzy reasoning is conducted based on Mamdani and Takagi Sugeno Kang model. Furthermore, there are some formula for conjunction and disjunction operators.

The Mamdani model: A fuzzy system with, e.g., two inputs $x1$ and $x2$ (antecedents) and a single output y (consequent) is described by the following fuzzy IF-THEN rule:

`IF x1 is A1 and x2 is A2 THEN y is B`

where $A1$ and $A2$ are the fuzzy sets representing the antecent pairs and B is the fuzzy set representing the consequent.

The Takagi Sugeno Kang model: Suppose we have two inputs $x1$ and $x2$ and output y, then the fuzzy IF-THEN rule is as follows:

`IF x1 is A1 and x2 is A2 THEN y is y = f(x1, x2)` where $y = f(x1, x2)$ is a crisp function in the consequent part which is usually a polynomial function, and $A1$ and $A2$ are the fuzzy sets representing the antecent pairs.

Futhermore, this function has the following capabilities:

- It supports unary operators (not) and binary operators (`AND` and `OR`).
- It provides linguistic hedge (`extremely`, `very`, `somewhat`, `slightly`).
- there are several methods for the t-norm and s-norm.

Value

a matrix of the degrees of the rules.

See Also

`defuzzifier`, `rulebase`, and `fuzzifier`.

`norm.data` *The data normalization*

Description

This function is to transform from real-valued data into normalized data.

Usage

```
norm.data(dt.ori, range.data, min.scale = 0,
  max.scale = 1)
```

Arguments

`dt.ori`	a matrix ($n \times m$) of the original data.
`range.data`	a matrix ($2 \times n$) containing the range of the data, where n is the number of variables, and first and second rows are the minimum and maximum value, respectively.
`min.scale`	the minimum value within normalization.
`max.scale`	the maximum value within normalization.

Value

the normalized data

See Also

`denorm.data`

`plotMF` *The plotting function*

Description

This function can be used to plot the shapes of the membership functions.

Usage

```
plotMF(object)
```

Arguments

`object`	an `frbs-object` or a list of parameters to plot membership functions when we build the frbs model without learning. For plotting using the list, there are several parameters that must be inserted in params as follows. • `var.mf`: a matrix of membership function of input and output variables. Please see `fuzzifier`. • `range.data.ori`: a matrix $(2 \times n)$ containing the range of the data, where n is the number of variables, and first and second rows are the minimum and maximum values, respectively. • `num.labels`: the number of linguistic terms of the input and output variables. For example: `num.labels <- matrix(c(3, 3, 3), nrow = 1)` It means we have 3 linguistic values/labels for two input variables and one output variable. • `names.variables`: a list of names of variables. For example: `names.variables <- c("input1", "input2", "output1")`

Examples

```
## The following examples contain two different cases which are
## using an frbs-object and the manual way.
##
## 1. Plotting using frbs object.
data(iris)
irisShuffled <- iris[sample(nrow(iris)),]
irisShuffled[,5] <- unclass(irisShuffled[,5])
tra.iris <- irisShuffled[1:105,]
tst.iris <- irisShuffled[106:nrow(irisShuffled),1:4]
real.iris <- matrix(irisShuffled[106:nrow(irisShuffled),5], ncol = 1)

## Please take into account that the interval needed is the range of input data only.
range.data.input <- matrix(c(4.3, 7.9, 2.0, 4.4, 1.0, 6.9, 0.1, 2.5), nrow=2)

## generate the model
method.type <- "FRBCS.W"
control <- list(num.labels = 7, type.mf = 1)
```

```
## Not run: object <- frbs.learn(tra.iris, range.data.input, method.type, control)

## plot the frbs object
## Not run: plotMF(object)

## 2. Plotting using params.
## Define shape and parameters of membership functions of input variables.
## Please see the fuzzifier function of how to contruct the matrix.
varinp.mf <- matrix(c(2, 0, 20, 40, NA, 4, 20, 40, 60, 80, 3, 60, 80, 100, NA,
                      2, 0, 20, 40, NA, 4, 20, 40, 60, 80, 3, 60, 80, 100, NA,
                      2, 0, 20, 40, NA, 4, 20, 40, 60, 80, 3, 60, 80, 100, NA,
                      2, 0, 20, 40, NA, 4, 20, 40, 60, 80, 3, 60, 80, 100, NA),
                      nrow = 5, byrow = FALSE)

## Define the shapes and parameters of the membership functions of the output variables.
varout.mf <- matrix(c(2, 0, 20, 40, NA, 4, 20, 40, 60, 80, 3, 60, 80, 100, NA),
                      nrow = 5, byrow = FALSE)
var.mf <- cbind(varinp.mf, varout.mf)
range.data <- matrix(c(0,100, 0, 100, 0, 100, 0, 100, 0, 100), nrow=2)
num.labels <- matrix(c(3,3,3,3,3), nrow = 1)
names.variables <- c("input1", "input2", "input3", "input4", "output1")

## plot the membership function.
## Not run: plotMF(object = list(var.mf = var.mf, range.data.ori = range.data,
          num.labels = num.labels, names.variables = names.variables))
## End(Not run)
```

---

predict.frbs *The frbs prediction stage*

Description

This is the main function to obtain a final result as predicted values for all methods in this package. In order to get predicted values, this function is run using an `frbs-object`, which is typically generated using `frbs.learn`.

Usage

```
## S3 method for class 'frbs'
predict(object, newdata, ...)
```

Arguments

`object`	an `frbs-object`.
`newdata`	a data frame or matrix ($m \times n$) of data for the prediction process, where m is the number of instances and n is the number of input variables. It should be noted that the testing data must be expressed in numbers (numerical data).
`...`	the other parameters (not used)

Value

The predicted values.

See Also

frbs.learn and frbs.gen for learning and model generation, and the internal main functions of each method for the theory: WM, SBC, HyFIS, ANFIS, FIR.DM, DENFIS, FS.HGD, FRBCS.W, GFS.FR.MOGUL, GFS.Thrift, GFS.GCCL, FRBCS.CHI, FH.GBML, GFS.LT.RS, and SLAVE.

Examples

```
##################################
## I. Regression Problem
###################################
## In this example, we just show how to predict using Wang and Mendel's technique but
## users can do it in the same way for other methods.
data.train <- matrix(c(5.2, -8.1, 4.8, 8.8, -16.1, 4.1, 10.6, -7.8, 5.5, 10.4, -29.0,
                       5.0, 1.8, -19.2, 3.4, 12.7, -18.9, 3.4, 15.6, -10.6, 4.9, 1.9,
                       -25.0, 3.7, 2.2, -3.1, 3.9, 4.8, -7.8, 4.5, 7.9, -13.9, 4.8,
                       5.2, -4.5, 4.9, 0.9, -11.6, 3.0, 11.8, -2.1, 4.6, 7.9, -2.0,
                       4.8, 11.5, -9.0, 5.5, 10.6, -11.2, 4.5, 11.1, -6.1, 4.7, 12.8,
                       -1.0, 6.6, 11.3, -3.6, 5.1, 1.0, -8.2, 3.9, 14.5, -0.5, 5.7,
                       11.9, -2.0, 5.1, 8.1, -1.6, 5.2, 15.5, -0.7, 4.9, 12.4, -0.8,
                       5.2, 11.1, -16.8, 5.1, 5.1, -5.1, 4.6, 4.8, -9.5, 3.9, 13.2,
                       -0.7, 6.0, 9.9, -3.3, 4.9, 12.5, -13.6, 4.1, 8.9, -10.0,
                       4.9, 10.8, -13.5, 5.1), ncol = 3, byrow = TRUE)

data.fit <- matrix(c(10.5, -0.9, 5.2, 5.8, -2.8, 5.6, 8.5, -0.2, 5.3, 13.8, -11.9,
                     3.7, 9.8, -1.2, 4.8, 11.0, -14.3, 4.4, 4.2, -17.0, 5.1, 6.9,
                     -3.3, 5.1, 13.2, -1.9, 4.6), ncol = 3, byrow = TRUE)

newdata <- matrix(c(10.5, -0.9, 5.8, -2.8, 8.5, -0.2, 13.8, -11.9, 9.8, -1.2, 11.0,
                    -14.3, 4.2, -17.0, 6.9, -3.3, 13.2, -1.9), ncol = 2, byrow = TRUE)

range.data<-matrix(c(0.9, 15.6, -29, -0.2, 3, 6.6), ncol=3, byrow = FALSE)
##############################################################
## I.1 Example: Implementation of Wang & Mendel
##############################################################
method.type <- "WM"

## collect control parameters into a list
## num.labels = 3 means we define 3 as the number of linguistic terms
control.WM <- list(num.labels = 3, type.mf = "GAUSSIAN", type.tnorm = "MIN",
               type.snorm = "MAX", type.defuz = "WAM",
               type.implication.func = "ZADEH", name = "Sim-0")

## generate the model and save it as object.WM
object.WM <- frbs.learn(data.train, range.data, method.type, control.WM)

## the prediction process
## The following code can be used for all methods
res <- predict(object.WM, newdata)
```

`rulebase` *The rule checking function*

Description

This function checks the consistency of a rule definition (given by the user). The rulebase consists of several fuzzy IF-THEN rules. The rules could be in a list or matrix type. Generally, there are three types of rule structures which are rules based on Mamdani, Takagi Sugeno Kang and fuzzy rule-based classification systems (FRBCS).

Usage

```
rulebase(type.model, rule, func.tsk = NULL)
```

Arguments

`type.model`	a value determining the type of model to use. Here, `MAMDANI` and `TSK` mean Mamdani and Takagi Sugeno Kang model, respectively.
`rule`	a matrix or list of rules.
`func.tsk`	a matrix representing the consequent parts of rules in Takagi Sugeno Kang formulation.

Details

For rules of the Mamdani model, there are 2 parts in each rule, the antecedent and the consequent part, which are separated by "->".

For example: `r1 <- c("a1","and","b1","->", "c1")`

It means that "IF input.variable1 is a1 and input.variable2 is b1 THEN output.variable is c1"

Here, ("a1", "and", "b1") is the antecedent, with "a1" and "b1" being linguistic terms, and ("c1") is the consequent part.

A fuzzy IF-THEN rule base with several rules is defined in the following way:

```
r1 <- c("not a1","and","b1", "->", "c1")
r2 <- c("a2","or","b2", "->", "c2")
r3 <- c("a3","or","b2", "->", "c3")
rule <- list(r1,r2,r3)
```

For rules of the Takagi Sugeno Kang model, the rules are at first defined without the consequent part, e.g.:

```
r1 <- c("a1",1,"b1","->")
r2 <- c("a2",2,"b2", "->")
r3 <- c("a3","2","b2", "->")
rule <- list(r1,r2,r3)
```

The consequences are defined then as a matrix `fun_tsk`, which contains the linear equations of the consequences of the rules. The dimension of this matrix is [<number_of_rules>, <number_of_variables> + 1]. The matrix has one extra column for the constants. If there is no constant, a zero is put.

So, for example, if we have 3 rules and 2 linguistic variables (A, B), the matrix `fun_tsk` has dim(3,3), as in:

```
func.tsk <- matrix(c(1, 1, 5, 2, 1, 3, 1, 2, 2),  nrow=3, ncol=3, byrow = TRUE)
```

Furthermore, we can represent linguistic hedges within the rules. The kinds of hedges used are

- `"extremely"` reduces the truth value. For example, `"extremely a1"` means membership function $a1 = \mu(a1)^3$.
- `"very"` reduces the truth value. For example, `"very a1"` means membership function $a1 = \mu(a1)^2$.
- `"somewhat"` increases the truth value. For example, `"somewhat a1"` means membership function $a1 = \mu(a1)^0.5$.
- `"slightly"` increases the truth value. For example, `"slightly a1"` means membership function $a1 = \mu(a1)^0.33$

An example of fuzzy IF-THEN rules using linguistic hedge is:

```
r1 <- c("very a1","and","b1","->","c1")
r2 <- c("a2",2,"b2", "->", "c2")
r3 <- c("a3","2","slightly b2", "->", "c3")
rule <- list(r1,r2,r3)
```

Furthermore, the following is an example in order to give names to the linguistic terms in the input and output variables.

```
varinput.1 <- c("a1", "a2", "a3")
varinput.2 <- c("b1", "b2")
names.varinput <- c(varinput.1, varinput.2)
names.varoutput <- c("c1", "c2", "c3")
```

In case of FRBCS model, the structure of rules are quite similar with Takagi Sugeno Kang model. But, instead of using linear equation in consequent part, consequent parts in FRBCS are represented by class. For example, Take into account that consequent parts expresses classes.

```
rule<-matrix(c("v.1_a.2","and","v.2_a.2","and","v.3_a.3","and","v.4_a.2","->","3",
"v.1_a.2","and","v.2_a.3","and","v.3_a.1","and","v.4_a.3","->","1",
"v.1_a.2","and","v.2_a.2","and","v.3_a.2","and","v.4_a.2","->","2"),
nrow=3, byrow=TRUE)
```

Where, `"1"`, `"2"`, `"3"` represent class `1`, `2`, and `3`.

Note that the names of the linguistic terms must be unique and if we are using the learning methods, the fuzzy IF-THEN rules will be generated automatically as the outputs of `frbs.learn`.

Value

fuzzy IF-THEN rule base

See Also

`defuzzifier`, `inference`, and `fuzzifier`

SBC *The subtractive clustering and fuzzy c-means (SBC) model building*

Description

This is the internal function that implements a combination of the subtractive clustering method and fuzzy c-means. It is used to solve regression tasks. Users do not need to call it directly, but just use `frbs.learn` and `predict`

Usage

```
SBC(data.train, range.data.ori, r.a = 0.5,
  eps.high = 0.5, eps.low = 0.15)
```

Arguments

`data.train`	a matrix ($m \times n$) of data for the training process, where m is the number of instances and n is the number of variables; the last column is the output variable.
`range.data.ori`	a matrix ($2 \times n$) containing the range of the data, where n is the number of variables, and first and second rows are the minimum and maximum value, respectively.
`r.a`	the radius defining a neighborhood.
`eps.high`	an upper threshold value.
`eps.low`	a lower threshold value.

Details

This method was proposed by S. Chiu. For generating the rules in the learning phase, the subtractive clustering method is used to obtain the cluster centers. Subtractive clustering (SBC) is an extension of Yager and Filev's mountain method. SBC considers each data point as a potential cluster center by determining the potential of a data point as a function of its distances to all the other data points. A data point has a high potential value if that data point has many nearby neighbors. The highest potential is chosen as the cluster center and then the potential of each data point will be updated. The process of determining new clusters and updating potentials repeats until the remaining potential of all data points falls below some fraction of the potential of the first cluster center. After getting all the cluster centers from subtractive clustering, the cluster centers are optimized by fuzzy c-means.

References

R. Yager and D. Filev, "Generation of fuzzy rules by mountain clustering," J. of Intelligent and Fuzzy Systems, vol. 2, no. 3, pp. 209 - 219 (1994).

S. Chiu, "Method and software for extracting fuzzy classification rules by subtractive clustering", Fuzzy Information Processing Society, NAFIPS, pp. 461 - 465 (1996).

See Also

SBC.test, frbs.learn, and predict

SBC.test *SBC prediction phase*

Description

This function is the internal function of the SBC method to compute the predicted values.

Usage

```
SBC.test(object, newdata)
```

Arguments

object	the frbs-object.
newdata	a matrix ($m \times n$) of data for the prediction process, where m is the number of instances and n is the number of input variables.

See Also

SBC

SLAVE *SLAVE model building*

Description

This is the internal function that implements the structural learning algorithm on vague environment (SLAVE). It is used to handle classification tasks. Users do not need to call it directly, but just use frbs.learn and predict.

Usage

```
SLAVE(data.train, persen_cross = 0.6,
  persen_mutant = 0.3, max.iter = 10, max.gen = 10,
  num.labels, range.data.input, k.lower = 0.25,
  k.upper = 0.75, epsilon = 0.1)
```

Arguments

`data.train`	a matrix ($m \times n$) of normalized data for the training process, where m is the number of instances and n is the number of variables; The last column is the output variable. Note the data must be normalized between 0 and 1.
`persen_cross`	a real number between 0 and 1 representing the probability of crossover.
`persen_mutant`	a real number between 0 and 1 representing the probability of mutation.
`max.iter`	the maximal number of iterations.
`max.gen`	the maximal number of generations for the genetic algorithm.
`num.labels`	a number of the linguistic terms.
`range.data.input`	a matrix containing the ranges of the normalized input data.
`k.lower`	a lower bound of the noise threshold.
`k.upper`	an upper bound of the noise threshold.
`epsilon`	a value between 0 and 1 representing the covering factor.

Details

This method is adopted from A. Gonzalez and R. Perez's paper which is applied for classification problems. SLAVE is based on the iterative rule learning approach which means that we get only one fuzzy rule in each execution of the genetic algorithm. In order to eliminate the irrelevant variables in a rule, SLAVE has a structure composed of two parts: the first part is to represent the relevance of variables and the second one is to define values of the parameters. The following steps are conducted in order to obtain fuzzy rules:

- Step 1: Use the genetic algorithm process to obtain ONE RULE for the system.
- Step 2: Collect the rule into the final set of rules.
- Step 3: Check and penalize this rule.
- Step 4: If the stopping criteria is satisfied, the system returns the set of rules as solution. Otherwise, back to Step 1.

This method uses binary codes as representation of the population and applies the basic genetic operators, i.e., selection, crossover, and mutation on it. And, the best rule is obtained by calculating the degree of consistency and completeness.

References

A. Gonzalez, R. Perez, "Selection of relevant features in a fuzzy genetic learning algorithm", IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, vol. 31, no. 3, pp. 417 - 425 (2001).

`SLAVE.test` *SLAVE.test: The prediction phase*

Description

This function is the internal function of the SLAVE method to compute the predicted values.

Usage

```
SLAVE.test(object, newdata)
```

Arguments

`object`	the `frbs-object`.
`newdata`	a matrix ($m \times n$) of data for the prediction process, where m is the number of instances and n is the number of input variables.

Value

A matrix of predicted values.

`summary.frbs` *The summary function for frbs objects*

Description

This function enables the output of a summary of the `frbs-object`.

Usage

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

Arguments

`object`	the `frbs-object`
`...`	the other parameters (not used)

Details

This function displays several components of the object. The components of one particular method can be different from components of other methods. The following is a description of all components which might be printed.

- The name of the model: A name given by the user representing the name of the simulation or data or model.
- Model was trained using: It shows which method we have been used.
- The names of attributes: a list of names of training data.
- The interval of training data: It is a matrix representing the original interval of data where the first and second rows are minimum and maximum of data, respectively. The number of columns represents the number of variables.
- Type of FRBS model: a description expresses one of the following FRBS model available such as `"MAMDANI"`, `"TSK"`, `"FRBCS"`, `"CLUSTERING"`, `"APPROXIMATE"`, and `"2TUPPLE"`.
- Type of membership function: a description expresses one of the following shapes of membership functions: `"GAUSSIAN"`, code"TRIANGLE", `"TRAPEZOID"`, `"SIGMOID"`, and `"BELL"`.
- Type of t-norm method: a description expresses one of the following type of t-norm: `"MIN"`, `"PRODUCT"`, `"HAMACHER"`, `"YAGER"`, and `"BOUNDED"`.
- Type of s-norm method: a description expresses one of the following type of s-norm: `"MAX"`, `"SUM"`, `"HAMACHER"`, `"YAGER"`, and `"BOUNDED"`.
- Type of defuzzification technique: a description expresses one of the following types: `"WAM"`, `"FIRST_MAX"`, `"LAST_MAX"`, `"MEAN_MAX"`, and `"COG"`.
- Type of implication function: a description expresses one of the following types: `"DIENES_RESHER"`, `"LUKASIEWICZ"`, `"ZADEH"`, `"GOGUEN"`, `"GODEL"`, `"SHARP"`, `"MIZUMOTO"`, `"DUBOIS_PRADE"`, and `"MIN"`.
- The names of linguistic terms of the input variables: These names are generated automatically by frbs expressing all linguistic terms considered. Generally, these names are built by two parts which are the name of variables expressed by `"v"` and the name of linguistic terms of each variables represented by `"a"`. For example, `"v.1_a.1"` means the linguistic value `"a.1"` of the first variable (v.1). However, we provide different format if we set the number of linguistic terms (`num.labels`) to 3, 5, 7. For example, for the number of label 3, it will be `"small"`, `"medium"`, and `"large"`.
- The names of linguistic terms of the output variable: For the Mamdani model, since the frbs package only considers single output, the names of the linguistic terms for the output variable are simple and clear and start with `"c"`. However, for Takagi Sugeno Kang model and fuzzy rule-based classification systems, this component is always `NULL`.
- The parameter values of membership functions of the input variables (normalized): It is represented by a matrix ($5 \times n$) where n depends on the number of linguistic terms on the input variables and the first row of the matrix describes a type of membership function, and the rest of rows are their parameter values. For example, label `"v.1_a.2"` has value 4.0, 0.23, 0.43, 0.53, 0.73 on its column. It means that the label a.2 of variable v.1 has a parameter as follows. 4.0 on the first row shows `TRAPEZOID` shape in the middle position, while 0.23, 0.43, 0.53, and 0.73 are corner points of a `TRAPEZOID`. Furthermore, the following is the complete list of shapes of membership functions:

- `TRIANGLE`: 1 on the first row and rows 2, 3, and 4 represent corner points.
- `TRAPEZOID`: 2, 3, or 4 on the first row means they are `TRAPEZOID` in left, right and middle side, respectively, and rows 2, 3, 4, and 5 represent corner points. But for `TRAPEZOID` at left or right side the fifth row is `NA`.
- `GAUSSIAN`: 5 on the first row means it uses `GAUSSIAN` and second and third row represent mean and variance.
- `SIGMOID`: 6 on the first row and two parameters (gamma and c) on second and third rows.
- `BELL`: 7 on the first row and three parameters (a, b, c) on second, third, and fourth rows.

- The fuzzy IF-THEN rules: In this package, there are several models for representing fuzzy IF-THEN rules based on the method used.
  - Mamdani model: they are represented as a knowledge base containing two parts: antecedent and consequent parts which are separated by a sign "THEN", as for example in the following rule:
    `IF var.1 is v.1_a.1 and var.2 is v.2_a.2 THEN var.3   is c.2`
  - Takagi Sugeno Kang model: In this model, this component only represents the antecedent of rules while the consequent part will be represented by linear equations.
  - fuzzy rule-based classification systems (FRBCS): This model is quite similar to the Takagi Sugeno Kang model, but the consequent part expresses pre-defined classes instead of a simplify of linear equations.
  - approximate approach: Especially for `GFS.FR.MOGUL`, a matrix of parameters of membership functions is used to represent the fuzzy IF-THEN rules as well. The representation of rules and membership functions is a matrix $(n \times (p \times m))$ where n is the number of rules and m is the number of variables while p is the number of corner points of the membership function, if we are using `TRIANGLE` or `TRAPEZOID` then p = 3 or 4, respectively. For example, let us consider the triangular membership function and a number of variables of 3. The representation of rules and membership functions is as follows:
    `<<a11 a12 a13>> <<b11 b12 b13>> <<c11 c12 c13>>`.
- The linear equations on consequent parts of fuzzy IF-THEN rules: It is used in the Takagi Sugeno Kang model.
- The weight of the rules or the certainty factor: For the `FRBCS.W` method, this shows the weight related to the rules representing the ratio of dominance among the rules.
- The cluster centers: This component is used in clustering methods representing cluster centers.

`WM` *WM model building*

Description

This is the internal function that implements the model proposed by L. X. Wang and J. M. Mendel. It is used to solve regression task. Users do not need to call it directly, but just use `frbs.learn` and `predict`

Usage

```
WM(data.train, num.labels, type.mf = "GAUSSIAN",
  type.tnorm = "PRODUCT",
  type.implication.func = "ZADEH",
  classification = FALSE, range.data = NULL)
```

Arguments

`data.train` a matrix ($m \times n$) of normalized data for the training process, where m is the number of instances and n is the number of variables; the last column is the output variable. Note the data must be normalized between 0 and 1.

`num.labels` a matrix ($1 \times n$), whose elements represent the number of labels (linguistic terms); n is the number of variables.

`type.mf` the type of the membership function. See `frbs.learn`.

`type.tnorm` a value which represents the type of t-norm. See `inference`.

`type.implication.func` a value representing type of implication function. Let us consider a rule, $a \rightarrow b$,

- `DIENES_RESHER` means $(b > 1 - a?b : 1 - a)$.
- `LUKASIEWICZ` means $(b < a?1 - a + b : 1)$.
- `ZADEH` means $(a < 0.5||1 - a > b?1 - a : (a < b?a : b))$.
- `GOGUEN` means $(a < b?1 : b/a)$.
- `GODEL` means $(a <= b?1 : b)$.
- `SHARP` means $(a <= b?1 : 0)$.
- `MIZUMOTO` means $(1 - a + a * b)$.
- `DUBOIS_PRADE` means $(b == 0?1 - a : (a == 1?b : 1))$.
- `MIN` means $(a < b?a : b)$.

`classification` a boolean representing whether it is a classification problem or not.

`range.data` a matrix representing interval of data.

Details

The fuzzy rule-based system for learning from L. X. Wang and J. M. Mendel's paper is implemented in this function. For the learning process, there are four stages as follows:

- `Step 1:` Divide equally the input and output spaces of the given numerical data into fuzzy regions as the database. In this case, fuzzy regions refers to intervals for each linguistic term. Therefore, the length of fuzzy regions represents the number of linguistic terms. For example, the linguistic term "hot" has the fuzzy region $[1, 3]$. We can construct a triangular membership function having the corner points $a = 1$, $b = 2$, and $c = 3$ where b is a middle point that its degree of the membership function equals one.
- `Step 2:` Generate fuzzy IF-THEN rules covering the training data, using the database from Step 1. First, we calculate degrees of the membership function for all values in the training data. For each instance in the training data, we determine a linguistic term having a maximum degree in each variable. Then, we repeat the process for each instance in the training data to construct fuzzy rules covering the training data.

- `Step 3:` Determine a degree for each rule. Degrees of each rule are determined by aggregating the degree of membership functions in the antecedent and consequent parts. In this case, we are using the product aggregation operators.
- `Step 4:` Obtain a final rule base after deleting redundant rules. Considering degrees of rules, we can delete the redundant rules having lower degrees.

The outcome is a Mamdani model. In the prediction phase, there are four steps: fuzzification, checking the rules, inference, and defuzzification.

References

L. X. Wang and J. M. Mendel, "Generating fuzzy rule by learning from examples", IEEE Trans. Syst., Man, and Cybern., vol. 22, no. 6, pp. 1414 - 1427 (1992).

See Also

`frbs.learn`, `predict` and `frbs.eng`.

Index